CELIO HIRA

**ARQUIMEDIA: UMA PROPOSTA DE ARQUITETURA DE
SOFTWARE PARA TERMINAIS DE ACESSO À TV DIGITAL
INTERATIVA**

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

São Paulo

2008

CELIO HIRA

**ARQUIMEDIA: UMA PROPOSTA DE ARQUITETURA DE
SOFTWARE PARA TERMINAIS DE ACESSO À TV DIGITAL
INTERATIVA**

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

2008

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, de abril de 2008.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Hira, Celio

Arquimedia : uma proposta de arquitetura de software para terminais de acesso à TV digital interativa / C. Hira. -- ed.rev. -- São Paulo, 2008.

118 p.

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

1. Televisão digital 2. Televisão interativa 3. Arquitetura de software I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II. t.

AGRADECIMENTOS

À minha família, pela compreensão, afeto, apoio e incentivo.

Ao Prof. Dr. Marcelo Knörich Zuffo, pela orientação e por sempre acreditar no meu trabalho.

Ao Sr. Hilel Becher, gerente do Núcleo de Engenharia de Mídias do Laboratório de Sistemas Integráveis, por toda a ajuda e a colaboração prestada nesta dissertação.

A todos os meus professores, que contribuíram para a minha formação acadêmica.

E, finalmente, aos colegas Ciro de Melo, Diego Shimizu, Eduardo Carvalho, Fernando Teubl, Gil Barros, Hilton Fernandes, Karen Sawabe, Laisa Costa, Lucas Real, Luiz Cunha, Luiz Mellucci, Marcia Kondo, Maria Esther, Maria Francesca, Rafael Alonso, Rogério Nunes, Roseli Lopes e a todos os outros colegas com quem tive a oportunidade de trabalhar, que de uma forma ou de outra me ajudaram na realização deste trabalho.

FINANCIAMENTO

Este trabalho contou com o financiamento da Associação do Laboratório de Sistemas Integráveis Tecnológico (LSI-TEC), da Financiadora de Estudos e Projetos (FINEP) e da Gerência de Programas Educacionais da Intel.

RESUMO

Os sistemas de televisão digital terrestre têm se estabelecido e expandido em várias regiões do mundo. A expectativa é de que 1,4 bilhão de televisores analógicos sejam substituídas por televisores e receptores digitais, também conhecidos como terminais de acesso. Daí a importância destes dispositivos que recebem e decodificam o sinal digital. Diante da diversidade de plataformas de *hardware* de diferentes fabricantes e perfis de terminais de acesso, existe a necessidade de um alto investimento para criar e desenvolver aplicações e *software* intermediário (*Middleware*) na medida em que existe a necessidade de diferentes códigos-fontes para cada uma das plataformas. Além desta fragmentação de implementações por conta das várias plataformas de *hardware*, existe a demanda crescente por desempenho e por soluções com um conjunto maior de funções. Para endereçar a estas questões, existem organizações, iniciativas e tecnologias emergentes em sistemas embarcados multimídia, algumas delas baseadas em componentes de *software*, que estão consolidando técnicas que podem ser aplicadas à estrutura de *software* de um terminal de acesso, dentre as quais destacamos a UHAPI, o M3W, o OpenMAX, a IOS e o GStreamer. Esta dissertação apresenta uma proposta de arquitetura de *software* denominada Arquimedia com base em uma análise comparativa destes trabalhos. Foram realizados e analisados ensaios de desempenho parciais para a validação da proposta.

ABSTRACT

The terrestrial digital television systems have been established and expanded by the worldwide. It is expected that 1.4 billion analog TV sets will be replaced by set-top-boxes and digital receivers TV sets, also known as access terminals. This fact shows the relevance of these devices that receive and decode the digital signals. Given the diversity of the hardware platforms of different manufacturers and different set-top-boxes profiles, there is a need for a major investment to create and develop applications and Middleware because for each hardware platform, different source codes need to be developed. Beside the fragmentation of implementations resulted by the diversity of hardware platforms, there is an increasing demand for performance and solutions with more functions. To address these questions, there are organizations, initiatives and emerging technologies, some of them based on software components, that are consolidating techniques that can be applied to the software stack of a set-top-box, from among we emphasize the UHAPI, M3W, IOS and GStreamer. This master thesis deals a comparative analysis of these technologies and presents a set-top-box software architecture proposal called Arquimedia. Performance tests were made and analyzed to validate the proposal.

SUMÁRIO

CAPÍTULO 1	INTRODUÇÃO	1
1.1	Relevância	3
1.2	Motivação.....	5
1.3	Objetivos	6
1.4	Metodologia	7
1.5	Estrutura da Dissertação.....	7
CAPÍTULO 2	ARQUITETURAS DE <i>SOFTWARE</i> PARA TERMINAIS DE ACESSO	9
2.1	Arquitetura de <i>Software</i>	9
2.2	Arquitetura Tradicional de um Terminal de Acesso em TV Digital Interativa	10
2.2.1	Aplicações de TV Digital.....	11
2.2.2	<i>Middleware</i> de TV Digital	11
2.2.3	Sistemas Operacionais de TV Digital	13
2.3	Arquitetura do Terminal de Acesso de Referência	14
2.3.1	Principais Requisitos da Arquitetura do Terminal de Acesso de Referência	15
	Utilização de Alternativas Tecnológicas Baseadas em <i>Software</i> Aberto e Livre	16
	Tempo de inicialização e carga do sistema operacional	17
	Escalabilidade	17
	Requisitos de Tempo Real	18
	Tamanho do Sistema Operacional	19
2.3.2	IOS (<i>Interface for Operating Systems</i>)	19
	A área funcional IOS-BASE	22
	A área funcional IOS-NET.....	23
	A área funcional IOS-SEC	23
	A área funcional IOS-IOD	23
	A área funcional IOS-TV	24
	A área funcional IOS-GRA.....	25
	A área funcional IOS-SND	26
	A área funcional IOS-PWR.....	26
	A área funcional IOS-QOS	26
2.4	Trabalhos Correlatos	27
2.4.1	O Fórum CELF (Consumer Electronics Linux Forum)	27
2.4.2	Componentes de <i>Software</i>	30
	Modelos de componentes	32
	Arcabouço de componentes	33
2.4.3	A Tecnologia COM (<i>Component Object Model</i>).....	34
	Interfaces e Objetos COM.....	35
	Ponteiros de Interfaces e Interfaces.....	36
	O método IUnknown.....	36
2.4.4	O padrão aberto UHAPI.....	37
2.4.5	O padrão MPEG <i>Multimedia Middleware</i>	39
	APIs de funcionalidade Multimídia	41
	APIs de Gerenciamento.....	42

APIs de Operação.....	42
Abordagem utilizada	43
Modelo Conceitual do MPEG <i>Multimedia Middleware</i>	46
Relação entre UHAPI e MPEG <i>Multimedia Middleware</i> (M3W)	47
2.4.6 O arcabouço DirectShow	48
2.4.7 O arcabouço GStreamer	48
2.4.8 O grupo Khronos.....	50
A Interface de Programação OpenMAX AL	51
A Interface de Programação OpenMAX IL	51
A Interface de Programação OpenMAX DL	51
2.5 Conclusões	52
CAPÍTULO 3 ANÁLISE E COMPARAÇÃO DE TÉCNICAS, FERRAMENTAS E ARCABOUÇOS	54
3.1 Avaliação da IOS (<i>Interface for Operating System</i>)	55
3.1.1 Funcionalidades cobertas da IOS	55
3.1.2 <i>Design</i> de projeto da IOS	56
3.1.3 Escalabilidade da IOS	56
3.1.4 Usabilidade da IOS	57
3.2 Avaliação da UHAPI.....	57
3.2.1 Funcionalidades cobertas da UHAPI	57
3.2.2 <i>Design</i> de projeto da UHAPI	57
Modelo de <i>Threads</i> da UHAPI	62
Notificações de Eventos da UHAPI.....	63
Tratamento de Erros na UHAPI.....	64
3.2.3 Escalabilidade da UHAPI	65
3.2.4 Usabilidade da UHAPI.....	66
3.3 Avaliação do MPEG <i>Multimedia Middleware</i>	67
3.3.1 Funcionalidades cobertas do MPEG <i>Multimedia Middleware</i>	67
3.3.2 <i>Design</i> de projeto do MPEG <i>Multimedia Middleware</i>	68
3.3.3 Escalabilidade do MPEG <i>Multimedia Middleware</i>	68
3.3.4 Usabilidade do MPEG <i>Multimedia Middleware</i>	68
3.4 Avaliação da API OpenMAX do Grupo Khronos	69
3.4.1 Funcionalidades cobertas do OpenMAX AL.....	69
3.4.2 <i>Design</i> de projeto do OpenMAX	70
A abordagem adotada pela API OpenMAX AL	72
Modelo de <i>Threads</i> do OpenMAX	78
Notificações de Eventos do OpenMAX.....	79
Tratamento de erros do OpenMAX.....	79
3.4.3 Escalabilidade do OpenMAX AL	80
3.4.4 Usabilidade do OpenMAX AL	81
3.5 Avaliação do arcabouço GStreamer.....	82
3.5.1 Funcionalidades cobertas pelo arcabouço GStreamer	82
3.5.2 <i>Design</i> de projeto do arcabouço GStreamer.....	82
Modelo de <i>Threads</i> do GStreamer	85
Notificações de Eventos do GStreamer.....	85
Tratamento de Erros do GStreamer.....	85
3.5.3 Escalabilidade do arcabouço GStreamer.....	86
3.5.4 Usabilidade do arcabouço GStreamer.....	86

3.6	Pontos fortes e Pontos fracos das abordagens em componentes de <i>software</i>	87
3.7	Conclusões	89
CAPÍTULO 4 PROPOSTA DA ARQUITETURA ARQUIMEDIA		92
4.1	Apresentação da proposta da Arquitetura de <i>Software</i> Arquimedia	92
4.2	Ensaio do GStreamer como arcabouço multimídia	94
4.3	Ensaio da utilização de APIs multimídia baseadas em componentes	99
4.3.1	Aplicação construída sobre a UHAPI	99
4.3.2	Aplicação construída sobre a IOS	100
4.4	Conclusões	106
CAPÍTULO 5 CONSIDERAÇÕES FINAIS		107
5.1	Conclusões	107
5.2	Contribuições	108
5.3	Trabalhos Futuros	109
REFERÊNCIAS		110
APÊNDICES		116

LISTA DE FIGURAS

Figura 1 – Esquema simplificado de um sistema completo de Televisão Digital. Extraído de (LSI-TEC, 2005)	1
Figura 2 – Foco deste trabalho – as camadas de <i>software</i> intermediário (destacadas em verde) entre as Aplicações e o <i>Middleware</i> e o <i>hardware</i> do terminal de acesso. ...	6
Figura 3 – Arquitetura típica da pilha de <i>software</i> de terminais de acesso utilizada em países que já adotaram a TV Digital. Modificada a partir de (CESAR, 2005).....	10
Figura 4 – Arquitetura de <i>Software</i> do Terminal de Acesso de Referência (LSI-TEC, 2005)	15
Figura 5 – Comparação entre a arquitetura de <i>software</i> tradicional (utilizada em regiões que já adotaram a TV Digital como EUA, Japão e Europa) e a arquitetura de <i>software</i> do Terminal de Acesso de Referência (a ser adotada no Brasil)	20
Figura 6 – Áreas funcionais da IOS. Extraído de (LSI-TEC, 2005).....	21
Figura 7 – Relacionamento entre componentes, modelo de componentes e arcabouço de componentes. Extraído de (SPAGNOLI; BECKER, 2003)	33
Figura 8 – Cliente usando um objeto COM através de um ponteiro para uma interface. Modificado a partir de (SEI, 2007).....	35
Figura 9 – Posição da UHAPI na pilha de <i>software</i> . Modificado a partir de (VUGTS, 2005)	38
Figura 10 – Camadas Arquiteturais que compõem o Sistema segundo a abordagem do MPEG <i>Multimedia Middleware</i> (GELISSEN, 2004).	44
Figura 11 – Modelo Conceitual do MPEG <i>Multimedia Middleware</i> (M3W). Modificado a partir de (ISO/IEC, 2004).....	46
Figura 12 – Exemplo de grafo construído. Modificado a partir de (DIRECTSHOW, 2008).	48
Figura 13 – Estrutura de <i>software</i> para o GStreamer.....	49
Figura 14 – Exemplo de <i>pipeline</i> do GStreamer.....	49
Figura 15 – Estrutura de <i>software</i> para o OpenMAX. Modificada a partir de (KHRONOS, 2007a).....	52
Figura 16 – Relacionamento entre os trabalhos detalhados no Capítulo 2.....	53
Figura 17 – Ilustração de um Componente Lógico da UHAPI. Modificado a partir de (UHAPI Forum, 2004).....	58
Figura 18 – Relação entre componentes lógicos e componentes de implementação. Modificado a partir de (UHAPI Forum, 2004).....	59
Figura 19 – Ilustrando o conceito de Caso de Uso da UHAPI. Modificada a partir de (UHAPI Forum, 2004).	59
Figura 20 – Ilustração dos principais elementos da UHAPI. Extraído de (UHAPI Forum, 2004).	60
Figura 21 – Formato típico dos nós de interface COM (LÜDERS, 2003).	61
Figura 22 – Componente Lógico definido por um fabricante. Modificado a partir de (VUGTS, 2005).	66
Figura 23 – Componente Lógico definido pela UHAPI com a adição de uma interface definida por um fabricante. Modificado a partir de (VUGTS, 2005).	66
Figura 24 – Recomendações de APIs do MPEG <i>Multimedia Middleware</i>	67
Figura 25 – Relacionamento entre OpenMAX AL, OpenMAX IL e OpenMAX DL. Modificado a partir de (KHRONOS, 2007c).....	71

Figura 26 – Diagrama de Estados de um Objeto para o OpenMAX AL. Modificado a partir de (KHRONOS, 2007b).	73
Figura 27 – Ilustração de uma interface do OpenMAX AL (KHRONOS, 2007b)	74
Figura 28 – Utilização de métodos pelo OpenMAX AL	74
Figura 29 – Aplicação OpenMAX AL, o objeto Engine e abstração de objetos. Modificado a partir de (KHRONOS, 2007b).	75
Figura 30 – Ilustração de um elemento de origem. Modificado a partir de (RUBY-GNOME2, 2007).	82
Figura 31 – Ilustração de um filtro. Modificado a partir de (RUBY-GNOME2, 2007).	83
Figura 32 – Ilustração de um filtro com mais de um pino de saída. Modificado a partir de (RUBY-GNOME2, 2007).	84
Figura 33 – Ilustração de um elemento de entrada. Modificado a partir de (RUBY-GNOME2, 2007).	84
Figura 34 – Arquitetura de <i>software</i> proposta	92
Figura 35 – Trecho de código em que é criado o Gerenciador de Conexões	99
Figura 36 – Trecho de código exemplificando a seleção de um caso de uso	100
Figura 37 – Trecho de código em que inicializa-se a libavcodec	101
Figura 38 – Trecho de código para a abertura do arquivo com os fluxos de áudio e vídeo	101
Figura 39 – Trecho de código para recuperar informações dos fluxos audiovisuais....	102
Figura 40 – Trecho de código para obter um ponteiro para o contexto dos codecs	102
Figura 41 – Trecho de código para encontrar o codec para decodificação.....	102
Figura 42 – Trecho de código para inicializar as bibliotecas FusionSound e DirectFB	103
Figura 43 – Trecho de código para o disparo das <i>threads</i> de renderização e decodificação	103
Figura 44 – Trecho de código para liberar os elementos inicializados.....	103
Figura 45 – Função usada para renderizar fluxos de vídeo	105
Figura 46 – Função para configurar a saída de áudio	105

LISTA DE TABELAS

Tabela 1 – Partes da ISO/IEC 23004 - MPEG <i>Multimedia Middleware</i>	47
Tabela 2 – Descrição geral das funções cobertas pela IOS.....	56
Tabela 3 – Principais pontos das tecnologias IOS, UHAPI, M3W, OpenMAX e GStreamer	90
Tabela 4 – Porcentagem (%) de CPU utilizada na decodificação e exibição de arquivos com fluxos de vídeo utilizando o Ffplay e a aplicação GStreamer	96
Tabela 5 – Porcentagem (%) de memória utilizada na decodificação e exibição de arquivos com fluxos de áudio e vídeo utilizando o ffplay e a aplicação GStreamer	97

LISTA DE ABREVIATURAS E SIGLAS

ACAP	<i>Advanced Common Application Platform</i>
ALSA	<i>Advanced Linux Sound Architecture</i>
API	<i>Application Programming Interface</i>
ARIB	<i>Association of Radio Industries and Businesses</i>
ASIC	<i>Application-Specific Integrated Circuit</i>
ATSC	<i>Advanced Television Systems Committee</i>
AVC	<i>Advanced Video Coding</i>
BML	<i>Broadcast Markup Language</i>
CELF	<i>Consumer Electronics Linux Forum</i>
COM	<i>Component Object Model</i>
CORBA	<i>Common Object Request Broker Architecture</i>
CPU	<i>Central Processing Unit</i>
DASE	<i>Digital TV Application Software Environment</i>
DCOM	<i>Distributed Component Object Model</i>
DRM	<i>Digital Rights Management</i>
DSP	<i>Digital Signal Processor</i>
DVB	<i>Digital Video Broadcasting</i>
EPG	<i>Electronic Program Guides</i>
FFT	<i>Fast Fourier Transform</i>
HAL	<i>Hardware Abstraction Layer</i>
HTML	<i>Hyper Text Markup Language</i>
ID	Identificador
IEC	<i>International Electrotechnical Commission</i>

IOS	<i>Interface for Operating Systems</i> (Interface do Sistema Operacional)
ISDB	<i>Integrated Services Digital Broadcasting</i>
ISO	<i>International Organization for Standardization</i>
ITU	<i>International Telecommunication Union</i>
JPEG	<i>Joint Photographic Experts Group</i>
MHP	<i>Multimedia Home Platform</i>
MMS	<i>Multimedia Messaging Service</i>
MPEG	<i>Motion Picture Experts Group</i>
M3W	<i>MPEG Multimedia Middleware</i>
NCL	<i>Nested Context Language</i>
OCAP	<i>OpenCable Application Platform</i>
OMG	<i>Object Management Group</i>
PC	<i>Personal Computer</i>
PVR	<i>Personal Video Recorder</i>
ROM	<i>Read-Only Memory</i>
SBTVD	Sistema Brasileiro de Televisão Digital
SMS	<i>Short Message Service</i>
SOTAR	Sistema Operacional do Terminal de Acesso de Referência
TAR	Terminal de Acesso de Referência
TV	Televisão
UHAPI	<i>Universal Home API</i>
URL	<i>Uniform Resource Locator</i>
VTABLE	<i>Virtual Table</i>

CAPÍTULO 1

INTRODUÇÃO

Milhões de pessoas na América do Norte, Europa e Japão já possuem acesso à tecnologia de televisão digital. Esta tecnologia tem se disseminado em várias regiões do mundo, a partir da definição dos sistemas e padrões de TV digital desde o início da década de 1990.

Para os usuários desta nova tecnologia, a transição da TV analógica para a TV digital vai propiciar a transmissão de sinais de áudio e vídeo de alta qualidade, sem interferências e distorções, em função da maior robustez na transmissão e na recepção do sinal de TV Digital. A introdução da TV Digital permitirá também a interatividade com a oferta de serviços e aplicações ao telespectador, além de um maior número de canais (MORRIS; SMITH-CHAIGNEAU, 2005).

Figura 1 – Esquema simplificado de um sistema completo de Televisão Digital.
Extraído de (LSI-TEC, 2005)

De modo geral, um sistema de televisão digital pode ser dividido em três elementos: a geração do sinal, a transmissão do sinal e a recepção do sinal. A geração do sinal é realizada normalmente dentro de uma emissora de TV e é responsável por gerar fluxos de áudio, vídeo e dados, que serão multiplexados no tempo e, em seguida, modulados. Já a transmissão do sinal engloba a transmissão de informações entre emissora e usuários. E, finalmente, o sistema de recepção do sinal é composto por

uma antena, um televisor e um terminal de acesso (podendo estar embutido ao televisor). A Figura 1 apresenta um esquema simplificado de um sistema completo de televisão digital, indicando como estes componentes estão relacionados e a inserção do terminal de acesso nesta cadeia.

O **terminal de acesso** tem como função receber e decodificar o sinal audiovisual codificado na forma digital, além de possibilitar a recepção de dados oferecendo interatividade ao usuário. Inicialmente, o terminal de acesso deverá ser adquirido separadamente por parte dos consumidores, mas, futuramente, com os avanços da microeletrônica e a consolidação destas tecnologias, as TVs deverão embutir os terminais de acesso.

Baseado em dados da ITU (*International Telecommunications Union*) do final do ano 2000, existem cerca de 1,4 bilhão de televisores analógicos em todo o mundo. A expectativa é que todos estes dispositivos sejam substituídos por televisores digitais e terminais de acesso, criando negócios que irão movimentar bilhões de dólares por ano nos mercados de equipamentos de radiodifusão, eletrônicos de consumo, computadores e indústrias de semicondutores nos próximos vinte anos (WU, Y; HIRAKAWA, S.; REIMERS, U. H.; WHITAKER, J, 2006).

Particularmente no Brasil, o processo de implantação da TV Digital foi motivado pela publicação do Decreto N^o 4.901 (BRASIL, 2003) pelo governo brasileiro em 2003. Este Decreto instituiu o Sistema Brasileiro de Televisão Digital (SBTVD). A partir de então, o governo brasileiro financiou o projeto de desenvolvimento da Arquitetura do Terminal de Acesso de Referência. Este projeto, executado ao longo do ano de 2005 e coordenado pelo Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo, contou com a participação de várias universidades brasileiras e empresas privadas em pesquisas para a definição do modelo de referência do sistema brasileiro de televisão digital e do padrão de televisão digital a ser adotado no país. Estas pesquisas foram consolidadas, integradas e apresentadas numa sessão pública realizada no prédio da Administração da Escola Politécnica da Universidade de São Paulo em 9 de dezembro de 2005.

Em 2006, através da publicação do Decreto N^o 5.820 (BRASIL, 2006), em 29 de junho, o governo brasileiro determinou que o Brasil deve adotar, como base, o

padrão de sinais da tecnologia japonesa ISDB (*Integrated Services Digital Broadcasting*), que será expandida incluindo as inúmeras inovações propostas pelos pesquisadores brasileiros. Assim, neste decreto foram definidas todas as bases tecnológicas do SBTVD, que então foram objeto de padronização pela indústria, radiodifusores e universidades.

Os resultados dos trabalhos realizados ao longo de 2005 e a publicação do Decreto em 2006 (BRASIL, 2006) foram utilizados para a definição das especificações das normas do padrão brasileiro. O SBTVD em sua forma padronizada foi lançado comercialmente por dez emissoras em 2 de dezembro de 2007 em São Paulo e, até dezembro de 2009, todas as capitais do país deverão receber o sinal digital no padrão SBTVD.

1.1 Relevância

No processo de definição e consolidação das normas do SBTVD, foi dada uma especial atenção às questões da TV Digital interativa prevista nos dois decretos presidenciais, Decreto 4.901 (BRASIL, 2003) e Decreto 5.820 (BRASIL, 2006). Desta forma o recurso da interatividade prevê terminais de acesso mais avançados, cuja arquitetura da plataforma de circuitaria (*hardware*) se aproxima de um computador convencional, com seus elementos de processamento de alto desempenho, memória e armazenamento de massa, além dos componentes e circuitos dedicados à sintonização de canais e decodificação multimídia. A diversidade dos circuitos do terminal de acesso, levou o autor a investigar com mais atenção os aspectos de *software* levando a considerar a pesquisa e proposta de uma arquitetura de *software* do terminal de acesso de referência.

Um dos principais requisitos da arquitetura de *software* do Terminal de Acesso de Referência é a sua capacidade de possibilitar implementações em uma grande variedade de plataformas de circuitaria de terminais de acesso.

Os terminais de acesso podem ser classificados em quatro grandes categorias quanto à interatividade (LEMO; FERNANDES; SILVEIRA, 2004; SOUZA; OLIVEIRA, 2005):

- Não interativo: esta categoria é considerada a mais simples pois apenas transforma o sinal modulado recebido em sinais de áudio e vídeo. De maneira simplificada, as etapas envolvidas são: sintonização, demodulação, decodificação de canal, demultiplexação, decodificação e exibição de áudio, vídeo e dados e tratamento do sinal de saída;
- Interatividade local: esta categoria é considerada a mais básica das categorias de interatividade. A antena doméstica continua recebendo fluxos de áudio e vídeo, enquanto o terminal de acesso armazena o fluxo de dados. As aplicações que estão sendo executadas permitem ao telespectador a interação propriamente dita. Porém o telespectador não consegue realizar o envio de dados em direção ao emissor, pois não possui um canal de retorno. Como exemplo de aplicações para este nível de interatividade, podem-se citar a configuração de legendas, jogos residentes e guias eletrônicos de programação, também conhecidos como EPGs, do inglês Electronic Program Guide;
- Interatividade intermitente: a antena doméstica continua exercendo a função de receptora de fluxos de áudio e vídeo. Já o terminal de acesso apresenta uma mudança significativa para o aumento da interatividade: o canal de retorno. Este canal permite que o telespectador transmita fluxos de dados ao servidor, mas este não pode enviar respostas ao telespectador. Por isso, o canal de retorno é considerado não dedicado. Sendo uma comunicação de dados unidirecional, esta categoria de interatividade é muito utilizada em aplicações como votações, pesquisas de opinião, entre outras;
- Interatividade permanente: esta categoria é uma evolução da interatividade intermitente, na qual a comunicação de dados deixa de ser unidirecional para se tornar bidirecional, existindo para isso um canal de retorno dedicado ao terminal de acesso. Através desse nível de interatividade, é possível ter o acesso às funções básicas de um computador conectado à Internet e usufruir aplicações como navegação, e-mail, chat, competições interativas (ex: jogos multi-usuário em tempo real), compras, homebanking, educação à distância, entre outras.

Como pode ser observado, os terminais de acesso podem seguir diferentes perfis de *hardware* e de recursos disponíveis, desde o perfil mais simples possuindo somente o necessário para decodificar o sinal audiovisual na forma digital, como é o terminal não-interativo, até o perfil mais avançado, oferecendo recursos sofisticados de interatividade ao usuário final.

Esta diversidade de plataformas de *hardware* de diferentes fabricantes faz com que os códigos-fontes dos *softwares* para os terminais de acesso sejam diferentes para cada uma destas plataformas, aumentando o investimento para criar e desenvolver aplicativos e *middleware*.

Além da questão relacionada à fragmentação das plataformas de *hardware*, existe uma demanda crescente por desempenho e por soluções capazes de executar um número maior de funções.

O terminal de acesso mais básico do SBTVD, no caso o não interativo, possui demanda por requisitos de desempenho elevado para o processamento de serviços multimídia com base no estado da arte das tecnologias disponíveis.

1.2 Motivação

Durante as pesquisas realizadas ao longo do SBTVD, foi possível perceber que a especificação de um conjunto comum de interfaces que possa ser utilizada por múltiplas plataformas pode reduzir a fragmentação de implementações de *software* e acelerar o desenvolvimento e disponibilidade de aplicações. Este conjunto de interfaces de *software* escolhido no projeto SBTVD foi denominado de IOS (*Interface for Operating System*).

Existem algumas iniciativas recentes de consórcios da indústria como o *Consumer Electronics Linux Forum* (*Consumer Electronics Linux Forum*, 2007) e a UHAPI Forum (UHAPI Forum, 2004) e o Grupo Khronos (KHRONOS, 2007a) que também objetivam reduzir a fragmentação de implementações de *software* através da padronização das interfaces disponibilizadas pelas plataformas de *hardware*. Como apresentado ao longo deste trabalho, uma grande parte destes esforços da indústria utiliza-se de abordagens baseadas em componentes de *software*.

1.3 Objetivos

O objetivo principal do presente trabalho é propor uma nova arquitetura de *software* para terminais de acesso aos serviços de TV Digital interativa através da pesquisa, comparação e análise de técnicas, ferramentas e arcabouços utilizados em arquiteturas de *software* de sistemas embarcados multimídia considerando o contexto do SBTVD. Atribuiu-se o nome Arquimedia a esta arquitetura proposta, sendo uma composição do nome do famoso filósofo Arquimedes, com o termo multimídia.

Ao contrário da abordagem utilizada pela IOS na especificação inicial do SBTVD, a proposta do Arquimedia considera uma abordagem baseada em componentes de *software*. Sendo assim, foi realizado um levantamento das vantagens e desvantagens da abordagem de componentes de *software* para a área de TV Digital.

Neste trabalho o foco de atenção são as camadas de abstração intermediárias entre as camadas superiores (aplicações e *middleware*) e a plataforma de *hardware* do terminal de acesso, conforme apresentado na Figura 2.

Figura 2 – Foco deste trabalho – as camadas de *software* intermediário (destacadas em verde) entre as Aplicações e o *Middleware* e o *hardware* do terminal de acesso.

1.4 Metodologia

Para o desenvolvimento desta dissertação, foi utilizada uma metodologia cujas fases estão definidas a seguir:

- **Pesquisa:** fase de seleção de informações de tecnologias no estado da arte na área de sistemas embarcados multimídia aplicados à arquitetura de *software* de um Terminal de Acesso;
- **Análise das tecnologias:** fase que engloba as atividades de caracterização das tecnologias selecionadas na fase de Pesquisa. Cada uma destas tecnologias foi analisada em termos de funcionalidades cobertas, *design* de projeto, usabilidade e escalabilidade. Além disso, foi elaborada uma comparação com a descrição dos pontos fortes e fracos que as abordagens em componentes de *software* trazem em relação às abordagens que não utilizam componentes de *software*;
- **Proposição de uma Arquitetura:** fase que apresenta a proposta de uma arquitetura que represente o sistema em estudo. Aqui foram analisadas as várias propostas de arquitetura e, segundo as análises da fase anterior, apresentada a recomendação de uma arquitetura específica.

1.5 Estrutura da Dissertação

O texto encontra-se organizado em cinco capítulos.

O primeiro capítulo oferece uma visão inicial sobre os assuntos estudados, sobre a motivação e a importância do trabalho.

O segundo capítulo apresenta as bases teóricas deste trabalho, fornecendo ao leitor uma visão geral dos diversos conceitos abordados ao longo do desenvolvimento da dissertação.

O terceiro capítulo apresenta as análises realizadas com as tecnologias descritas no segundo capítulo com o objetivo de apresentar as principais características das tecnologias que podem ser incorporadas na proposição de uma arquitetura de *software* para terminais de acesso à TV Digital no Brasil;

O quarto capítulo traz a proposta da arquitetura de *software* com base nas análises do terceiro capítulo.

Por fim, o quinto e último capítulo contém as considerações finais do trabalho com as principais conclusões, contribuições e indicações de possíveis trabalhos futuros.

CAPÍTULO 2

ARQUITETURAS DE SOFTWARE PARA TERMINAIS DE ACESSO

O objetivo deste capítulo é apresentar uma revisão da literatura bem como aprofundar o entendimento sobre o delineamento de Arquiteturas de *Software* para terminais de acesso à TV Digital Interativa. Pretende-se neste capítulo fornecer os subsídios para a proposta da arquitetura Arquimedia, que será apresentada nos próximos capítulos.

Primeiramente, apresenta-se a definição de arquitetura de *software*. Em seguida, são descritas a arquitetura de *software* de terminais de acesso mais comumente utilizada nos países que já possuem a tecnologia de TV Digital e a proposta inicial da Arquitetura do Terminal de Acesso de Referência desenvolvida no Brasil. Por fim, apresentam-se tecnologias em pesquisa por instituições que estão concentrando esforços na consolidação de técnicas aplicáveis à arquitetura de *software* de um terminal de acesso.

2.1 Arquitetura de *Software*

A **arquitetura de *software*** de um sistema é a sua estrutura ou o seu conjunto de estruturas, que engloba os elementos de *software*, as propriedades destes elementos que são visíveis externamente e os relacionamentos entre estes elementos. As propriedades visíveis externamente referem-se às premissas que os outros elementos podem fazer de um determinado elemento, tais como serviços fornecidos, características de desempenho, tratamento de erros e uso de recursos compartilhados (SEI, 2006).

A arquitetura de *software* oferece informações sobre como os elementos se relacionam uns com os outros. Elementos interagem uns com os outros por meio de interfaces que oferecem detalhes sobre um elemento em partes públicas ou privadas. A arquitetura de *software* se preocupa com o lado público desta divisão. Por outro

lado, detalhes privados de elementos, que são tratados somente com uma implementação interna, não são arquiteturais (SEI, 2006).

Apesar da arquitetura de *software* tender a focar em informação estrutural, o comportamento de cada elemento é parte da arquitetura de *software* já que pode ser observado do ponto de vista de outro elemento. Este comportamento é o que permite elementos interagirem entre si, o que é claramente parte da arquitetura de *software* (SEI, 2006).

2.2 Arquitetura Tradicional de um Terminal de Acesso em TV Digital Interativa

Tradicionalmente, a Arquitetura de *Software* de um terminal de acesso em TV Digital Interativa é composta por uma camada de Aplicações, que são construídas sobre a camada de *Middleware*, que por sua vez acessam o sistema operacional. Este último gerencia os recursos de *hardware* do terminal de acesso. A Figura 3 ilustra a arquitetura típica da pilha de camadas de *software* de um terminal de acesso em países que já adotaram a TV Digital (CESAR, 2005).

Figura 3 – Arquitetura típica da pilha de *software* de terminais de acesso utilizada em países que já adotaram a TV Digital. Modificada a partir de (CESAR, 2005).

2.2.1 Aplicações de TV Digital

As aplicações de TV Digital são definidas como uma camada de *software* visível ao usuário, que faz a interação direta com o mesmo, sendo suportada pelas camadas inferiores (SOUZA; ELIAS, 2003). Oferecem serviços específicos ao usuário tais como educação à distância, governo eletrônico, telemedicina, dentre outros.

Para que possam ser executadas em múltiplas plataformas, as aplicações são desenvolvidas sobre uma camada de *software* intermediário conhecida como **Middleware**. De forma geral, pode-se dizer que os padrões de *middleware* definem dois tipos de aplicações a serem executadas em terminais de acesso: as aplicações declarativas e as aplicações procedurais (LEITE et. al., 2005).

As **aplicações procedurais** são formadas por comandos interpretados no terminal de acesso e traduzidos em linguagem de máquina. Desta forma, o terminal de acesso necessita da presença de um interpretador ou uma máquina virtual. A maioria dos *middlewares* utiliza a linguagem Java e como interpretador a JVM (*Java Virtual Machine*).

Já as **aplicações declarativas** são voltadas à apresentação de documentos hipermídia e, portanto, são mais simples, demandando menos recursos de processamento e memória. A maioria dos *middlewares* utiliza alguma linguagem baseada em HTML (*Hyper Text Markup Language*). Existem vários exemplo de linguagens para aplicações declarativas como o XML (*Extensible Markup Language*), o BML (*Broadcast Markup Language*) e o NCL (*Nested Context Language*).

As aplicações podem ser residentes, ou seja, armazenadas na memória ROM (ou FLASH ou outra memória não-volátil) do terminal de acesso. Os jogos e os guias eletrônicos de programação são exemplos deste tipo de aplicações.

2.2.2 Middleware de TV Digital

O *Middleware* de TV Digital é uma camada de *software* que serve como interface para camadas inferiores (como o sistema operacional) e superiores (aplicações), garantindo a comunicação entre as camadas de forma transparente. O

Middleware viabiliza e controla a interação dos usuários com os fluxos de multimídia e dados provenientes do sistema operacional (SOUZA; ELIAS, 2003).

Para muitos pesquisadores e desenvolvedores, o termo *middleware* é hoje sinônimo de *middleware* baseado em objetos, como o CORBA da OMG e o DCOM da Microsoft. Entretanto, o *middleware* baseado em objetos é somente um tipo de *middleware* junto a outros como os *middlewarees* baseados em eventos e orientados a mensagens. Uma definição mais básica de *middleware* de Geoff Coulson (PLAGEMANN, 2002) seria:

A função do *middleware* é facilitar a tarefa de projetar, programar e gerenciar aplicações distribuídas provendo um ambiente de programação distribuído simples, consistente e integrado. Essencialmente, o *middleware* é uma camada de *software*, ou plataforma que abstrai toda a complexidade e heterogeneidade do ambiente distribuído base com suas várias tecnologias de rede, arquiteturas de máquina, sistemas operacionais e linguagens de programação.

Em outras palavras, a abstração de toda a complexidade e heterogeneidade é o objetivo principal do *middleware*. Neste contexto, é importante notar que aplicações multimídia aumentam o problema da heterogeneidade, já que existem muitos formatos de codificação para vídeo, áudio, gráficos, etc. (PLAGEMANN, 2002)

Atualmente, existem várias alternativas para o *middleware* para TV Digital. Algumas delas são proprietárias como o MediaHighway e o OpenTV. Outras são padrões abertos e variam de acordo com a região (CESAR, 2005). Dentre os *middlewarees* mais conhecidos, podem ser elencados:

- MHP: definido pelo DVB;
- *DTV Application Software Environment* (DASE): definido pelo ATSC;
- *OpenCable Application Platform* (OCAP): definido pela CableLabs;
- *Advanced Common Application Platform* (ACAP): definido pelo ATSC;
- ARIB STD-B23/STD-B24: definido pelo ARIB.

Para o SBTVD, foi proposta uma alternativa de *middleware* conhecida como Ginga (GINGA, 2007). O Ginga foi dividido em duas partes: o Ginga-J, voltado para aplicações procedurais, e o Ginga-NCL, voltado às aplicações declarativas.

2.2.3 Sistemas Operacionais de TV Digital

Para controlar os recursos de *hardware* de um sistema embarcado como é um terminal de acesso, é necessária uma série de características que não são encontradas em um sistema operacional convencional.

Um **sistema embarcado** pode ser definido como qualquer sistema computacional ou dispositivo de computação que executa uma função dedicada ou é projetada para uso com uma aplicação de *software* específica (LATURA, 2002).

Atualmente, existem diversos aspectos no projeto de sistemas embarcados que seguem algumas tendências desafiantes (ANDREWS, 2005):

- **Complexidade:** níveis maiores de funcionalidade em conjunto com código legado e falta de abstração. Em dispositivos eletrônicos de consumo, esta complexidade é derivada de sistemas com a convergência de requisitos de armazenamento, conectividade, e integração aumentada de funcionalidade;
- **Flexibilidade:** mudanças posteriores, *download* de *software*, reuso;
- **Dependabilidade:** o nível de integridade nos casos de falha e não-falha tem aumentado. Isto é resultado não somente do medo da queda nas vendas como também devido a pressões oriundas da legislação;
- **Conectividade:** no nível de sistemas, existe uma pressão cada vez maior para que os sistemas trabalhem em conjunto, como, por exemplo, telefones móveis se comunicando com *notebooks*;
- **Modularidade:** necessidade para facilitar a manutibilidade, mas também para suportar o desenvolvimento concorrente;
- **Manutibilidade:** existe uma tendência em deixar o desenvolvimento monolítico devido a dificuldade de mudanças e não suportar estratégias de reuso;
- **Capacidade de atualização:** existe uma necessidade em possibilitar atualizações. Estas atualizações devem ser executadas tanto por especialistas como por leigos;

- **Tamanho e energia:** existe uma pressão por dispositivos cada vez menores que possam ser energizados por baterias por um longo período de tempo.

Os aspectos citados anteriormente serão considerados, na medida do possível, na proposta da arquitetura Arquimedia.

2.3 Arquitetura do Terminal de Acesso de Referência

A Arquitetura do Terminal de Acesso de Referência do SBTVD foi concebida para contemplar:

- **Flexibilidade**, ou seja, a possibilidade de implementações que variam em custo, em complexidade e aplicações;
- **Inclusão digital**, através da definição de uma arquitetura mínima para garantir baixo custo e acesso à informação e aos serviços para a população marginalizada e;
- **Escalabilidade**, ou seja, capacidade de incorporação de novas tecnologias e funcionalidades.

Tendo em vista tais desafios, a Arquitetura de Referência inclui a proposta de uma arquitetura de *software*. Esta arquitetura foi desenvolvida para oferecer recomendações a fabricantes de terminais de acesso e desenvolvedores de *software* embarcado, de forma a oferecer às aplicações de tempo real o uso inteligente dos recursos de *hardware* do terminal de acesso.

A Arquitetura do Terminal de Acesso de Referência difere das arquiteturas tradicionais, no qual o *middleware* tem a função de abstrair a plataforma. Nesta arquitetura, é proposta uma interface disponibilizada pelo sistema operacional que abstrai a implementação da plataforma para o *Middleware*. Esta interface é denominada de **Interface do Sistema Operacional** cujo acrônimo é IOS (do inglês *Interface for Operating System*).

A arquitetura de *software* proposta para o SBTVD está apresentada na Figura 4 (LSI-TEC, 2005):

Figura 4 – Arquitetura de *Software* do Terminal de Acesso de Referência (LSI-TEC, 2005)

2.3.1 Principais Requisitos da Arquitetura do Terminal de Acesso de Referência

Alguns requisitos apontados estão relacionados às ferramentas ou aos mecanismos computacionais de suporte estrutural, não estando diretamente relacionados aos recursos essenciais para os terminais de acesso, descritos anteriormente. Já outros requisitos estão voltados para suas funcionalidades primordiais como recepção de sinal digital, decodificação audiovisual e apresentação do conteúdo. Desta forma, os principais requisitos foram expandidos, conforme apresentado abaixo (LSI-TEC, 2005):

- Utilização de alternativas tecnológicas baseadas em *software* aberto e livre;
- Tempo de inicialização e carga do sistema operacional mínimo;
- Escalabilidade;
- Requisitos de Tempo Real;
- Tamanho do Sistema Operacional.

Estes cinco requisitos serão detalhados nas próximas seções.

Utilização de Alternativas Tecnológicas Baseadas em *Software* Aberto e Livre

Um requisito considerado no desenvolvimento da arquitetura de *software* de Referência foi o uso de alternativas tecnológicas baseadas em *software* aberto e livre, sendo aliado à atual política do governo brasileiro de apoio ao *software* livre.

Segundo a *Free Software Foundation* (FSF), o *software* livre tem como premissas básicas:

- Liberdade para execução do *software* para qualquer propósito;
- Liberdade para estudo do funcionamento do *software* e modificação do mesmo para atender as necessidades de cada usuário;
- Liberdade para redistribuição de cópias como forma de ajudar cada usuário na realização de suas atividades;
- Liberdade para liberar as modificações realizadas para beneficiar a comunidade dos usuários.

O conceito de *software* livre é comumente tido como sinônimo de código-aberto. Embora o *software* livre implique necessariamente na abertura do seu código-fonte, tornando-se efetiva a liberdade de modificá-lo, nem todo o programa de código-aberto é livre. Há exemplos concretos em que, embora o código-fonte esteja disponível para exame, não se concede ao usuário a liberdade de uso, cópia, modificação e/ou distribuição. O *software* livre não é necessariamente gratuito, podendo haver o pagamento de determinado valor (taxa, preço, etc) (CASTRO, 2007).

Assim, algumas características favoráveis ao uso do *software* livre são o custo e a independência em relação a um único mantenedor. O impacto do custo do terminal de acesso na digitalização de um sistema de TV é bastante agressivo devido ao número de unidades necessárias (uma para cada TV no sistema analógico). Considerando que o modelo de *software* proprietário geralmente requer o pagamento de *royalties*, o uso de alternativas tecnológicas livres torna-se bastante atrativo economicamente, desonerando os fabricantes de aparelhos e usuários finais. Além da flexibilidade proporcionada pelo fato de poder personalizar o sistema de acordo com

as necessidades, o *software* livre possibilita a criação de comunidades de desenvolvedores.

Tempo de inicialização e carga do sistema operacional

Um outro requisito importante levado em consideração é o tempo de inicialização. A importância deste trabalho está relacionada ao fato de que os usuários esperam que os televisores façam a exibição de imagens logo após ligá-lo. Um sistema de televisão analógico geralmente não demora mais do que quatro segundos até mostrar alguma imagem ao usuário.

Escalabilidade

A escalabilidade é um atributo desejável de um sistema. O conceito conota a habilidade de um sistema em acomodar um número crescente de elementos ou objetos, de forma a processar maiores volumes de trabalho, e/ou para ser suscetível a um crescimento do sistema. Uma escalabilidade pobre pode resultar em um baixo desempenho do sistema, necessitando uma reengenharia ou duplicação de sistemas.

A escalabilidade estrutural é a habilidade do sistema em expandir em uma determinada dimensão sem maiores modificações em sua arquitetura. A escalabilidade de um sistema sujeita a demandas crescentes é crucial para o seu sucesso a longo prazo. Ao mesmo tempo, o conceito de escalabilidade e o entendimento dos fatores que podem melhorar ou piorar o sistema são vagos ou até mesmo subjetivos.

Muitos projetistas de sistemas e analistas de desempenho têm uma idéia intuitiva para escalabilidade, mas os fatores determinantes nem sempre são claros. Eles podem variar de um sistema para outro. Um autor não identificado atribuiu uma definição para escalabilidade em (WHATIS, 2007). São citadas duas convenções: (1) a habilidade de uma aplicação computacional ou produto (*hardware* ou *software*) em cumprir suas funções ao mudar o seu tamanho ou volume para atender às necessidades do usuário; e (2) a habilidade de não somente cumprir suas funções no caso do sistema ser reescalado, mas tirar proveito disso, por exemplo, caso um sistema seja movido de um sistema operacional menor para um maior ou de um ambiente com um único processador para um com vários processadores.

A habilidade para reescalar um sistema ou um de seus componentes pode depender dos tipos de estruturas de dados e algoritmos usados para implementá-los ou dos mecanismos que estes componentes utilizam para se comunicar uns com os outros.

As estruturas de dados suportam funções particulares de um sistema. Os algoritmos podem ser usados para procurar estas estruturas, para agendar atividades ou acessar recursos, para coordenar interações entre processos, ou para atualizar múltiplas cópias de dados em diferentes lugares. As estruturas de dados afetam não somente a quantidade de espaço requerido para executar uma função em particular, mas também o tempo (BONDI, 2000).

Existem diferentes tipos de escalabilidade como:

- **Escalabilidade espacial:** compatibilidade entre modelos de terminais de acesso distintos;
- **Escalabilidade temporal:** compatibilidade de aplicativos entre gerações de terminais de acesso distintas.

A escalabilidade é um requisito fundamental dada a variedade de terminais de acesso. Como os terminais de acesso seguem perfis de *hardware* de diferentes custos, nem todo componente de *hardware* presente em um terminal de perfil avançado estará presente no terminal de perfil básico. Isto exige que as APIs suportem as diferentes configurações de um terminal de acesso.

Idealmente, devem-se adotar procedimentos básicos para a verificação da presença de dispositivos de *hardware*. Caso o *hardware* não esteja presente, a abstração provida pela API deve permitir que uma camada de *software* responda pela funcionalidade requerida. Esta metodologia possibilitaria a utilização de soluções híbridas, com diferentes funcionalidades nas camadas de *hardware* e *software*.

Requisitos de Tempo Real

Em um sistema de tempo real, os requisitos relacionados às restrições de tempo devem ser atendidos pois, caso contrário, ocorrerão falhas no sistema. Os terminais de acesso devem ser capazes de processar vários fluxos de áudio, vídeo e dados em tempo real para exibir o sinal de televisão ao usuário de forma suave.

Como exemplo de um sistema de tempo real para TV Digital, pode-se considerar um terminal de acesso reproduzindo um filme, no qual os fluxos de áudio e vídeo possuem requisitos de tempo real já que devem ser decodificados e mostrados ao usuário de forma sincronizada. As informações de cada quadro do filme passam por uma série de transformações antes de serem exibidos na tela. O *hardware* que controla o sinal de televisão precisa receber cada quadro numa taxa fixa de frequência para apresentar cada imagem na sequência. Se um processo nesta cadeia não puder ser executado devido à sobrecarga do sistema ou problemas de escalonamento, um quadro pode ser perdido, podendo resultar na aparência de sobressaltos nas imagens apresentadas ao usuário e em atrasos significativos entre a recepção e a exibição do serviço.

Tamanho do Sistema Operacional

Outra característica com relação direta com o custo do terminal de acesso são as limitações típicas de um eletrônico de consumo, como a necessidade de recursos de processamento e memória. Desta maneira o sistema operacional desenvolvido deve ser minimalista, apresentando apenas ferramentas específicas para as funcionalidades projetadas para o terminal de acesso.

2.3.2 IOS (*Interface for Operating Systems*)

Um dos resultados gerados pelo projeto do SBTVD foi o desenvolvimento do Sistema Operacional do Terminal de Acesso de Referência (SOTAR) - considerando os requisitos da Arquitetura do Terminal de Acesso de Referência, o uso de *software* livre, a flexibilidade, a escalabilidade e os requisitos gerados pelo projeto concorrente do desenvolvimento do *middleware* de referência do SBTVD (Sistema Brasileiro de TV Digital).

Foram analisados vários sistemas operacionais sob licença de uso livre e em desenvolvimento ativo como FreeBSD, NetBSD, OpenBSD, eCos, Fiasco e Linux. O sistema operacional Linux foi escolhido como base tendo em vista os requisitos de *software* livre, escalabilidade e maior quantidade e disponibilização de ferramentas multimídia (HIRA et al, 2007).

O SOTAR disponibiliza uma interface de *software* denominada de Interface do Sistema Operacional cuja sigla é IOS (do termo em inglês *Interface for Operating Systems*) com o objetivo de oferecer as funções para a comunicação entre o *software* em execução e o *hardware*, assim como as demais abstrações normalmente definidas internamente em um sistema operacional convencional. Portanto, a Interface do Sistema Operacional (IOS) é uma camada de abstração que torna a implementação da plataforma transparente para o *middleware*. Funciona como referência para os desenvolvedores das camadas superiores de *software*. Como observado na Figura 5, a inserção da camada de IOS é uma das principais diferenças entre a Arquitetura Tradicional utilizada nos países que já adotaram a TV Digital e a Arquitetura do Terminal de Acesso de Referência do SBTVD. A ampla adoção da IOS permitirá que as aplicações residentes e o *middleware* sejam portados com maior facilidade para várias plataformas distintas, favorecendo os pequenos fabricantes.

Figura 5 – Comparação entre a arquitetura de *software* tradicional (utilizada em regiões que já adotaram a TV Digital como EUA, Japão e Europa) e a arquitetura de *software* do Terminal de Acesso de Referência (a ser adotada no Brasil)

Fazem parte da IOS as chamadas de sistema, funções de bibliotecas básicas, estruturas de dados e variáveis compartilhadas que viabilizam o acesso a tais funcionalidades.

No intuito de facilitar a compreensão e a organização, a IOS foi especificada em módulos denominados de **áreas funcionais**. Cada área funcional está relacionada a um conjunto específico de funções afins, conforme apresentado na Figura 6.

Figura 6 – Áreas funcionais da IOS. Extraído de (LSI-TEC, 2005)

A área funcional IOS-BASE

A IOS-BASE provê um conjunto mínimo de funções para a gerência de processos, acesso a sistemas de arquivos presentes em memória de massa ou memória volátil, gerenciamento de memória principal, comunicação básica entre processos locais e acesso a recursos como o relógio do sistema. Assim, a IOS-BASE garante acesso ao grupo de *Hardware* Básico de Computação, composto por:

- Unidade de Processamento Central (CPU);
- Memória principal;
- Memória de massa ou secundária;
- Relógio do sistema.

De forma geral, a IOS-BASE possui funções para:

- chamadas de sistema (*System Calls*);
- E/S padrão;
- tratamento de sinais para permitir a configuração e manipulação de sinais que devem ser feitos pelos processos;
- localização, permitindo a identificação de processos;
- comunicação entre processos;
- manipulação de relógio;
- construção de aplicações baseadas na linguagem C padrão;
- controle do carregamento dinâmico de código-objeto a ser agregado ao processo requisitante;
- tratamento de algumas das estruturas internas do sistema operacional por aplicações com requisitos de tempo real (exemplos: temporizadores, regiões de memória compartilhada, comunicação entre processos, entre outros);
- autenticação de usuários e grupos cadastrados no sistema;
- obtenção de dados sobre o estado do sistema.

A especificação POSIX (POSIX, 2003) para sistemas operacionais foi o padrão adotado como base para a IOS-BASE. Suas funções e chamadas de sistema são providas pela biblioteca Glibc (GLIBC, 2007).

A área funcional IOS-NET

A IOS-NET oferece as chamadas para o estabelecimento de um canal de comunicação entre processos por meio de uma rede de computadores, geralmente estabelecida através de um canal de retorno. Fornece funcionalidades para controle do *hardware* para o estabelecimento de conexões, tais como discagem (como em modems ligados à rede pública de telefonia). Assim, garante acesso ao grupo de *Hardware* de Comunicação em Rede, composto pelo canal de retorno.

As funções da área funcional IOS-NET seguem, em sua maioria, a especificação BSD Sockets, conforme documentado pelo padrão POSIX.

A IOS-NET oferece funções para comunicação em rede baseada em sockets e funções para configurar e verificar o estado do canal de retorno baseadas no *software* Wvdial (WVDIAL, 2007).

A área funcional IOS-SEC

A IOS-SEC oferece uma interface de segurança para autenticação e sigilo na troca de informações. Garante acesso ao grupo de *Hardware* de Segurança, composto por *SmartCards* e pelo TPM (*Trusted Platform Module*).

Grande parte das funções oferecidas por esta interface segue a API disponibilizada pelo projeto NSS (*Network Securities Services*) (NSS, 2007).

A área funcional IOS-IOD

A IOS-IOD oferece interfaces para gerenciar a comunicação com dispositivos de Entrada e Saída que são operadas pelo usuário. Assim, garante acesso ao grupo de *Hardware* de Interação com o usuário, composto por controle remoto, teclado, *joystick* e portas de entrada/saída.

Desta forma, a IOS-IOD provê uma interface de programação que abrange um conjunto completo de funções para calibração, obtenção de características, configuração e acesso ao dispositivo de controle direcional (*joystick*), ao controle remoto sem fio e ao painel frontal.

As funções disponibilizadas pela IOS-IOD foram desenvolvidas pelo consórcio ao longo do projeto (LSI-TEC, 2005).

A área funcional IOS-TV

A IOS-TV fornece funções para a seleção de canais de TV, acesso a dados sobre programação e dados multiplexados, entre outros. Assim, garante acesso ao grupo de *Hardware* Especializado de TV, composto por:

- *Front End* (composto por sintonizador, demodulador e decodificador de canais);
- Demultiplexador de fluxos de transporte;
- Decodificador de Áudio;
- Decodificador de Vídeo.

As funções da IOS-TV utilizam, em alguns casos, funcionalidades disponibilizadas pelos padrões Video4Linux2 (DIRKS, B.; VERKUIL, H.; RUBLI, M., 2006) e LinuxTV (LINUXTV, 2008).

A interface de programação do sintonizador da IOS-TV provê um conjunto completo de funções para a configuração e acesso ao dispositivo de demodulação e sintonização do sinal de TV Digital. São disponibilizadas funções para a mudança de frequência e para a identificação dos tipos de sinais captados bem como a qualidade destes sinais.

Já a interface de programação de Acesso Condicional da IOS-TV provê um conjunto de funções para a configuração do dispositivo de controle de acesso, responsável por efetuar o desembaralhamento (também conhecido como *descrambling*) do fluxo de transporte para, em seguida, entregá-lo ao demultiplexador. São disponibilizadas funções para a consulta de características presentes no *hardware* e para o fornecimento de chaves criptográficas por *software*.

A interface de programação do Demultiplexador da IOS-TV, por sua vez, provê um conjunto completo de funções para a configuração e obtenção de dados do dispositivo de demultiplexação de fluxos de transporte, e portanto, responsável por entregar os fluxos elementares a cada um dos dispositivos interessados, entre eles os

decodificadores de áudio e vídeo, além do próprio *software* em execução no terminal de acesso.

E, por fim, a interface de programação dos decodificadores da IOS-TV provê um conjunto completo de funções para a configuração dos dispositivos de decodificadores de fluxos elementares de áudio e vídeo, responsáveis por entregar aos dispositivos de saída (alto-falantes e TV) os sinais decodificados.

A área funcional IOS-GRA

A IOS-GRA é constituída por um conjunto de funções que realizam operações gráficas, possibilitando que as aplicações se comuniquem com o *Hardware* de Vídeo através de uma API. Assim, garante acesso ao grupo de *Hardware* de Exibição de Gráficos, composto por um controlador de vídeo.

As funções da IOS-GRA seguem a especificação da interface DirectFB.

O DirectFB é uma biblioteca de espaço de usuário que provê funcionalidades gráficas, suportando as seguintes operações (HUNDT, 2004):

- Traçado e preenchimento de retângulos;
- Traçado e preenchimento de triângulos;
- Traçado de linhas;
- Operações de *blitting*, que consiste em mover blocos de bits da memória principal para a região de memória do display;
- Renderização de fonte;
- Gerenciamento de janelas;
- Operações de *Color keying*;
- Manipulação de eventos de entrada;
- Mixagem de camadas.

Caso exista suporte a aceleração gráfica, a maioria das operações listadas anteriormente é executada por *hardware*. Caso contrário, são executadas por *software* de forma automática.

Esta biblioteca também possui chamadas para a manipulação de superfícies e janelas, bem como comandos de composição de janelas em uma mesma superfície e manipulação de *buffer* duplo ou triplo para a saída de vídeo.

A área funcional IOS-SND

A IOS-SND é responsável por oferecer funções básicas de acesso e controle de mistura do áudio dos aplicativos com o áudio da programação de TV. Assim, estas funções garantem acesso ao grupo de *Hardware* de Processamento de Áudio, composto pelo controlador de áudio.

As funções oferecidas por esta interface seguem, na sua maioria, as APIs disponibilizadas pelo FusionSound (FUSIONSOUND, 2007).

O FusionSound é uma biblioteca de renderização de áudio que segue algumas das principais características da biblioteca DirectFB, com um conjunto simples de chamadas para executar as funções essenciais no que se refere à saída de áudio de um terminal de acesso.

Essa biblioteca disponibiliza chamadas para manipulação de fontes distintas de áudio, além de suportar mixagem de som, ajuste de frequência, dentre outras.

A biblioteca FusionSound depende de parte da infra-estrutura interna do DirectFB.

A área funcional IOS-PWR

A IOS-PWR disponibiliza, para aplicações privilegiadas, um conjunto de funções para as ações de controle de energia que vão desde a desativação de dispositivos sem utilização até o desligamento do sistema. Assim, garante acesso ao grupo de *Hardware* de Controle de Energia, composto pelo sistema básico de E/S (BIOS) compatível com gerenciamento de energia. Para prover este tipo de serviço, o *hardware* deve ser compatível com algum mecanismo de gerenciamento de energia.

O consórcio responsável pelo SOTAR desenvolveu as funções disponibilizadas pela IOS-PWR (LSI-TEC, 2005).

A área funcional IOS-QOS

A IOS-QOS estabelece funções para a negociação de qualidade de serviço envolvendo recursos como CPU e canal de retorno, visando garantir a qualidade de

operação de aplicações com fortes requisitos de processamento e/ou comunicação. Dessa forma, tais recursos passam a ser compartilhados com maior justiça e menor probabilidade de ocorrência de sobrecargas, as quais levariam à violação da qualidade de operação das várias aplicações concorrentes. Trata-se de um grupo adicional, sem vínculo específico a um bloco de *hardware*, mas fundamental para o uso de mecanismos de reserva de recursos como processamento e rede em aplicações de tempo real. Esta IOS segue o padrão conhecido como QoSOS (MORENO et al., 2003).

2.4 Trabalhos Correlatos

Esta seção tem por objetivo apresentar alguns dos principais trabalhos sendo realizados por organizações emergentes visando desenvolver tecnologias que possam ser aplicadas em sistemas embarcados multimídia.

Os trabalhos aqui apresentados foram escolhidos de acordo com as suas relações com os principais requisitos da Arquitetura do Terminal de Acesso de Referência.

2.4.1 O Fórum CELF (Consumer Electronics Linux Forum)

O CELF (*Consumer Electronics Linux Forum*) é uma organização estabelecida para disseminar a utilização e oferecer oportunidades para discussões técnicas relacionadas ao Linux como plataforma de código-aberto para eletrônicos de consumo em geral (Consumer Electronics Linux Forum, 2007).

Este Fórum foi fundado em 2003 por oito empresas: Matsushita, Sony, Hitachi, NEC, Philips, Samsung, Sharp e Toshiba. Atualmente conta com mais de cinquenta sócios-membros.

Para a melhoria do Linux para eletrônicos de consumo, o CELF tem duas principais contribuições: a publicação de especificações e o ajuste de mudanças ao Linux aprovado pelo Fórum.

O propósito desta organização não é competir com outras organizações ou sistemas operacionais e, sim, melhorar o Linux para um uso em particular, criando uma estrutura para um esforço em conjunto.

Apesar do CELF não desenvolver código diretamente, existem Grupos de Trabalho estabelecidos para criar especificações e revisar implementações. Enquanto as especificações são desenvolvidas como parte da atividade do Fórum, o código é desenvolvido por entidades (tanto membros como não-membros) em atividades disjuntas do Fórum, e submetidas ao Fórum para aprovação e inclusão na Árvore de diretórios do CELF com o código-fonte (Consumer Electronics Linux Forum, 2007). São oito os Grupos de Trabalho:

1. O Grupo de Trabalho de Áudio, Vídeo e Processamento Gráfico

Este Grupo de Trabalho estabelece requisitos para as APIs de processamento gráfico e exibição de vídeo do ponto de vista do *Kernel*. Após uma avaliação das APIs existentes, o Grupo de Trabalho recomenda algumas soluções (Consumer Electronics Linux Forum, 2007).

Tais recomendações estão descritas numa especificação denominada de CELF AVG (Consumer Electronics Linux Forum, 2006) cuja última versão foi ratificada em Julho de 2006.

De forma geral, esta especificação faz a recomendação da utilização das API de quatro grandes projetos existentes:

- DirectFB (HUNDT, 2004), uma biblioteca gráfica para o sistema operacional Linux, de código-aberto, que fornece a abstração do *hardware* gráfico;
- OpenGL ES (KHRONOS, 2007a), uma API que fornece funcionalidades para jogos tridimensionais;
- ALSA (ALSA, 2007), um projeto de código-aberto que fornece um suporte eficiente para todas as interfaces de áudio;
- UHAPI (Universal Home API, 2007), uma família de APIs de funcionalidade multimídia que será detalhada mais adiante.

2. O Grupo de Trabalho de Tempo de inicialização

As especificações deste grupo de trabalho têm como objetivo minimizar o tempo necessário para inicializar o *kernel* Linux, estabelecendo requisitos relacionados

ao *firmware*, *kernel* do sistema operacional e espaço de usuário (Consumer Electronics Linux Forum, 2007).

Este trabalho é extremamente importante, já que a expectativa dos usuários é que o produto fique disponível para uso imediatamente após ligá-lo. Geralmente, sistemas operacionais como o Windows e o Linux utilizado em *desktops* e servidores demoram de trinta a sessenta segundos aproximadamente para ficar disponível (Consumer Electronics Linux Forum, 2007).

3. O Grupo de Trabalho de Televisão Digital

Os resultados dos trabalhos do grupo de Televisão Digital estão incluídos na especificação do CELF AVG.

4. O Grupo de Trabalho de Telefonia Móvel

Este Grupo de Trabalho tem como objetivo formalizar requisitos para componentes de sistema, além de identificar projetos de código-aberto existentes e necessários que fornecem componentes ou capacidades para telefones móveis baseados em Linux (Consumer Electronics Linux Forum, 2007).

5. O Grupo de Trabalho de Gerenciamento de Energia

Este Grupo de Trabalho tem como objetivo projetar uma arquitetura de gerenciamento de energia para o Linux, através da otimização do uso de energia no nível de sistema, envolvendo interações entre aplicação, sistema operacional e *hardware*. Além disso, o Grupo de Trabalho também deseja fornecer um arcabouço de gerenciamento de energia para várias plataformas de *hardware* (Consumer Electronics Linux Forum, 2007).

6. O Grupo de Trabalho de Tempo Real

Este Grupo de Trabalho tem como objetivo melhorar as capacidades de tempo real do Sistema Operacional Linux, tendo como o escopo o *Kernel* bem como bibliotecas no nível de usuário que coletivamente fornecem o ambiente em tempo de execução com características adequadas de tempo real (Consumer Electronics Linux Forum, 2007).

7. O Grupo de Trabalho de Tamanho do Sistema

Este Grupo de Trabalho tem como objetivo minimizar a memória requerida para operar um Sistema Operacional Linux, estabelecendo requisitos e recomendações técnicas (Consumer Electronics Linux Forum, 2007).

8. O Grupo de Trabalho de Segurança

Este Grupo de Trabalho tem como objetivo melhorar as capacidades de segurança do Sistema Operacional Linux (Consumer Electronics Linux Forum, 2007).

Pode-se perceber que vários dos Grupos de Trabalho do CELF como o AVG, o de tempo de inicialização, o de Televisão Digital, o de tempo real e o de tamanho do sistema possuem relação direta com os principais requisitos da Arquitetura do Terminal de Acesso de Referência.

2.4.2 Componentes de *Software*

O Desenvolvimento Baseado em Componentes surgiu como uma nova perspectiva para o desenvolvimento de *software*, já que, até então, a grande maioria dos produtos de *software* em sistemas embarcados multimídia eram baseados no desenvolvimento de blocos monolíticos, formados por um grande número de partes inter-relacionadas, os quais estavam implícitos na maior parte das vezes. Neste cenário, o objetivo é dividir estes blocos monolíticos em componentes interoperáveis, reduzindo a complexidade e o custo de desenvolvimento. Além disso, é preciso reconhecer o fato de que muitos sistemas apresentam componentes semelhantes ou mesmo idênticos, os quais podem ser reutilizados (SAMETINGER, 1997).

Existem diversas definições de componentes na literatura, mas neste trabalho, serão utilizadas as definições de SAMETINGER e de SZYPERSKI.

Em (SAMETINGER, 1997), os componentes são definidos como unidades reutilizáveis e autocontidas, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras em conformidade com um dado modelo de arquitetura de *software*, documentação apropriada e um grau de reutilização definido.

Outra definição de componente é feita em (SZYPERSKI, 1999): “o Componente de *Software* é uma unidade de composição com interfaces contratualmente especificadas e apenas com dependências de contexto explícitas. O componente de *software* pode ser usado independentemente e combinado com outras partes”.

Assim, utilizando estas duas definições, os componentes, nesta dissertação, possuem as seguintes características:

- **Autocontido:** refere-se à característica dos componentes de poderem ser reutilizáveis sem a necessidade de incluir ou depender de outros componentes. Caso exista alguma dependência, então todo o conjunto deve ser visto como o componente reutilizável (SPAGNOLI; BECKER, 2003);
- **Identificável:** os componentes devem ser facilmente identificáveis, ou seja, devem estar contidos em um único lugar ao invés de espalhados e misturados com outros artefatos de *software* ou documentação. Os componentes são tratados como artefatos por poderem assumir uma variedade de diferentes formas como, por exemplo, código-fonte, documentação e código executável (SPAGNOLI; BECKER, 2003);
- **Função específica:** os componentes possuem uma funcionalidade clara e específica que realizam e/ou descrevem. Podem realizar funções ou ser simplesmente descrições de funcionalidades. Com isso se deseja também caracterizar como componente toda a documentação do ciclo de vida do *software*, embora esta não inclua a codificação propriamente dita de funcionalidades (SPAGNOLI; BECKER, 2003);
- **Interface:** componentes devem ter uma interface clara, que indica como podem ser reusados e conectados a outros componentes, e devem ocultar os detalhes que não são necessários para o reuso (SPAGNOLI; BECKER, 2003);
- **Documentação:** a existência de documentação é indispensável para o reuso. O tipo de componente e sua complexidade irão indicar a conveniência do tipo de documentação (SPAGNOLI; BECKER, 2003);

- **Condição de reuso:** componentes devem ser mantidos de modo a preservar o reuso sistemático e a condição de reuso compreende diferentes informações como, por exemplo, quem é o proprietário, quem deve ser contatado em caso de problema, qual é a situação de qualidade, entre outras (SPAGNOLI; BECKER, 2003);
- **Unidade de atuação independente:** isto exige que o componente seja separado do ambiente e dos demais componentes. Também não deve existir a perspectiva de se ter acesso a detalhes de construção do componente. Logo, o componente precisa encapsular sua implementação e interagir com o ambiente através de interfaces bem definidas. Para que o componente possa ser combinado com outros componentes ele precisa ser suficientemente autocontido (SPAGNOLI; BECKER, 2003);
- **Dependências de contexto:** a necessidade dos componentes de especificarem suas dependências de contexto diz respeito à necessidade deles indicarem o que o ambiente em que serão acionados deve prover para que eles funcionem (SPAGNOLI; BECKER, 2003);
- **Conformidade com um modelo:** o componente deve estar em conformidade a algum modelo de componentes padronizado. Este modelo pode definir interfaces, meta-dados, documentação e composição de componentes (SPAGNOLI; BECKER, 2003).

Modelos de componentes

Um modelo de componentes representa um elemento da arquitetura do sistema na qual são definidos os padrões e convenções impostas aos componentes do sistema, de modo a descrever a função de cada um e como eles interagem entre si. Com isso, busca-se expressar restrições de projeto arquitetural ou global (BACHMANN, 2002).

Existe certa confusão relacionada aos termos modelo de componentes e arcabouço de componentes. Existe um relativo consenso na literatura quanto ao uso do nome modelo de componentes para identificar o conjunto de padrões e convenções com as quais os componentes devem estar em conformidade. Entretanto, a infra-estrutura

que dá suporte a estes modelos é encontrada na literatura como arcabouço de componentes (SPAGNOLI; BECKER, 2003), como será visto a seguir.

Arcabouço de componentes

Segundo (BACHMANN, 2002), o arcabouço de componentes representa a base sobre a qual estes padrões e convenções do modelo de componentes são empregados. Com isso, arcabouço e modelo de componentes podem ser considerados dois conceitos complementares e fortemente relacionados. As definições estabelecidas pelo modelo de componentes devem ser suportadas pelo arcabouço, bem como o arcabouço deve respeitar e regular as definições estabelecidas pelo modelo de componentes. Além disso, o uso do arcabouço de componentes possibilita que os desenvolvedores de componentes e aplicações não precisem se preocupar em implementar suas aplicações em inúmeros serviços complexos para troca de mensagens, passagens de dados e ligação dos componentes.

O relacionamento entre componentes, modelo e arcabouço de componentes pode ser identificado na Figura 7, adaptada de (CRNKOVIC; LARSSON, 2002).

Figura 7 – Relacionamento entre componentes, modelo de componentes e arcabouço de componentes. Extraído de (SPAGNOLI; BECKER, 2003)

Como mostra a Figura 7, o arcabouço de componentes oferece a infra-estrutura de suporte a comunicação e ligação dos componentes, através do fornecimento de serviços de coordenação. Já o modelo de componentes identifica as definições

quanto a tipos de componentes, formas de interação e recursos necessários. Os componentes definidos para executarem determinadas funcionalidades devem estar em conformidade com as definições do modelo de componentes, pertencendo a um dos possíveis tipos e respeitando as formas de interação definidas, além de se utilizarem dos serviços disponibilizados. Por fim, tem-se que a utilização dos componentes caracteriza a ligação entre o modelo e arcabouço de componentes (SPAGNOLI; BECKER, 2003).

2.4.3 A Tecnologia COM (*Component Object Model*)

O COM (*Component Object Model*) é uma tecnologia desenvolvida pela Microsoft com o objetivo de criar componentes de *software* binários que podem interagir. É a tecnologia base para outras da Microsoft como o OLE e o ActiveX.

Não se trata de uma linguagem orientada a objeto, mas um padrão. O COM especifica um modelo de objetos e requisitos de programação que possibilitam que objetos COM possam interagir com outros objetos. Estes objetos podem estar dentro do mesmo processo, em outros processos ou podem estar em máquinas remotas. O COM é referido como um padrão binário – um padrão que se aplica depois de um programa ter sido traduzido para a linguagem de máquina binária. (MICROSOFT, 2007).

O COM define uma maneira padrão de dispor em ordem tabelas de funções virtuais na memória (vtable) e para chamar tais funções através destas vtables (CARDOSO; ESTEVES, 2002).

O único requisito de linguagem para o COM é que o código deve ser gerado em uma linguagem que possa criar estruturas de ponteiros e, tanto explícita ou implicitamente, chamadas de funções através de ponteiros. Linguagens orientadas a objetos como o C++ e SmallTalk fornecem mecanismos que simplificam a implementação de objetos COM, mas linguagens como o C, Pascal e Java podem criar e usar objetos COM (MICROSOFT, 2007).

Em geral, o objeto de *software* é constituído de um conjunto de dados e funções que manipulam dados. Um objeto COM é um objeto no qual o acesso aos dados é obtido exclusivamente através de um ou mais conjuntos de funções relacionadas. Estes

conjuntos de funções são chamados de interfaces, e as funções de uma interface são chamados de métodos. Além disso, o COM requer que a única maneira de se obter acesso aos métodos de uma interface é através de um ponteiro para a interface.

Além de especificar o padrão de objetos binários básico, o COM define certas interfaces básicas que fornecem funções comuns para todas as tecnologias baseadas no COM, e fornece algumas funções de API que todos os componentes requerem.

Interfaces e Objetos COM

Os componentes em conformidade com a especificação do padrão binário COM, também conhecidos como objetos COM, fornecem serviços que podem ser invocados por clientes. Os serviços implementados por objetos COM são expostos através de um conjunto de interfaces que representam o único ponto de contato entre clientes e o objeto, como pode ser observado na Figura 8.

O COM define uma estrutura binária para a interface entre o cliente e o objeto. Esta estrutura binária provê a base para a interoperabilidade entre componentes de *software* escritos em linguagens arbitrárias. Enquanto o compilador pode reduzir as estruturas de linguagem para esta representação binária, a linguagem de implementação para clientes e objetos COM não é significativa. Assim, os objetos COM e clientes podem ser codificados em qualquer linguagem que suporte a estrutura binária do COM da Microsoft.

Figura 8 – Cliente usando um objeto COM através de um ponteiro para uma interface. Modificado a partir de (SEI, 2007)

O objeto COM pode suportar qualquer número de interfaces. Uma interface fornece uma coleção agrupada de métodos relacionados.

Os objetos e interfaces COM são especificados usando a linguagem de descrição de interface (IDL) da Microsoft.

Estas interfaces são consideradas logicamente imutáveis. Uma vez que a interface é definida, não deve ser alterada. Novos métodos não devem ser adicionados e métodos existentes não devem ser alterados. Esta restrição sobre as interfaces não é obrigatória, mas é uma regra recomendada aos desenvolvedores de componentes. Aderindo a esta restrição, diminui-se o risco de incompatibilidade de versões. Caso uma interface nunca mude, os clientes dependentes desta interface podem confiar em um conjunto consistente de serviços. Se uma nova funcionalidade tiver de ser adicionada a um componente, pode-se expor através de uma interface diferente. (SEI, 2007)

Ponteiros de Interfaces e Interfaces

Uma instância de uma implementação de interface é um ponteiro para um vetor de ponteiros de métodos, isto é, uma tabela de funções que faz referência a uma implementação de todos os métodos especificados na interface. Objetos com múltiplas interfaces podem fornecer ponteiros para mais de uma tabela de funções (MICROSOFT, 2007).

Cada interface, considerada um contrato imutável de um grupo funcional de métodos, é referida em tempo de execução com um IID (*Interface Identifier*) único global. Este IID, que é uma instância específica de um GUID (*Globally Unique Identifier*) suportado pelo COM, permite que o cliente requisiute um objeto precisamente, sem *overhead* desnecessário e sem confusão que possa vir a acontecer num sistema na qual existem múltiplas versões da mesma interface com o mesmo nome.

O método IUnknown

O COM define uma interface especial denominada **IUnknown**. Todos os objetos componentes implementam esta interface, e sendo assim, todas as interfaces COM derivam de **IUnknown**.

A interface **IUnknown** possui três métodos: **QueryInterface**, **AddRef** e **Release**.

AddRef e **Release** são métodos usados para a contagem de referências. O método **AddRef** de um objeto é chamado quando um outro objeto está usando a interface. Já o método **Release** é chamado quando o objeto não necessita mais do uso desta interface. Enquanto o contador de referências não for nulo, o objeto deve permanecer na memória. Quando o contador de referências se torna igual a zero, o objeto pode ser liberado da memória de forma segura já que nenhum outro objeto faz referências a ele.

O método **QueryInterface** é o mecanismo que permite aos clientes descobrirem dinamicamente (em tempo de execução) se uma interface é suportada ou não por um objeto. Ao mesmo tempo, é o mecanismo que o cliente do objeto usa para obter um ponteiro de interface de um objeto. Quando uma aplicação necessita usar alguma função de um objeto, faz uma chamada ao método **QueryInterface** deste objeto, requisitando um ponteiro para a interface que implementa a função desejada. Se o objeto suporta esta interface, será retornado o ponteiro de interface apropriado e um código de sucesso. Se o objeto não suporta a interface requisitada, então, um valor de erro será retornado. A aplicação irá examinar o código de retorno e em função da resposta tomar as ações adequadas.

2.4.4 O padrão aberto UHAPI

A UHAPI (*Universal Home API*) é uma solução padrão aberta com o objetivo de facilitar o desenvolvimento de aplicações para dispositivos de eletrônicos de consumo. Esta solução é resultado da iniciativa da UHAPI Forum, que é uma organização sem fins lucrativos criada em Novembro de 2004 por Samsung, HP, Philips e DTVIA (*Digital TV Industry Alliance of China*). Assim, a UHAPI Forum fornece à indústria o suporte para a implementação da especificação da UHAPI, tentando reduzir o tempo de desenvolvimento e custos, além de aumentar a interoperabilidade entre produtos, sistemas e *softwares* de eletrônicos de consumo.

Para tanto, a UHAPI Forum objetiva padronizar um conjunto de APIs abertas para uma variedade de aplicações como Televisão Digital e Analógica, terminais de acesso, DVD *players* e outros dispositivos de áudio e vídeo. Desta forma, a adoção

da UHAPI irá ajudar a reduzir o custo e a complexidade de portar aplicações para novos processadores e arquiteturas, acelerando o desenvolvimento de aplicações.

A UHAPI consiste de especificações de uma família de APIs para funcionalidades de áudio e vídeo, sendo projetada tendo o *middleware* em mente. Desta forma, suporta um grande conjunto de *middlewares* (VUGTS, 2005).

A Figura 9 ilustra a posição da UHAPI na pilha de camadas de *software* de produtos de eletrônicos de consumo.

A programação utilizando a UHAPI é baseada em interfaces: um conjunto consistente e coerente de interfaces bem definidas, tanto sintaticamente quanto semanticamente. Estas interfaces são independentes de processador e sistema operacional e permitem liberdade na implementação, suportando fluxos em *hardware* e *software*, de tal modo que a implementação do componente de *software* não é exposta.

Utiliza mecanismos padronizados para notificações, tratamento de erros e gerenciamento de conexões, além de um pequeno subconjunto de mecanismos semelhantes ao COM como o **IUnknown** (**QueryInterface**, **AddRef** e **Release**), **vtables** e **GUIDs**, que são adequadas para sistemas embarcados de tempo real.

Figura 9 – Posição da UHAPI na pilha de *software*. Modificado a partir de (VUGTS, 2005)

Para ilustrar a importância deste trabalho, o conjunto de APIs da UHAPI, além de ser uma das APIs sendo recomendada pelo CELF, está sendo proposto para padronização pelo MPEG. Este padrão irá ser referido como a ISO/IEC 23004: M3W (MPEG *Multimedia Middleware*), também conhecido por MPEG-E.

2.4.5 O padrão MPEG *Multimedia Middleware*

As atividades do MPEG têm sido organizadas em volta de grandes projetos, tipicamente direcionados por maiores domínios de aplicação, ou funcionalidades. Esta abordagem levou ao seguinte conjunto de padrões MPEG (PEREIRA, 2006):

- ISO/IEC 11172 (MPEG-1), “Codificação de imagens em movimento e informação de áudio associada em até 1,5 Mbit/s”, direcionado principalmente ao armazenamento digital em CD-ROM;
- ISO/IEC 13818 (MPEG-2), “Codificação genérica de imagens em movimento e informação de áudio associada”, principalmente direcionado à TV Digital e ao armazenamento digital;
- ISO/IEC 14496 (MPEG-4), “Codificação de Objetos Audiovisuais”, fornecendo novas funcionalidades baseadas em objetos, integração natural e sintética, novas formas de interação, etc;
- ISO/IEC 15938 (MPEG-7), “Interface de Descrição de Conteúdo Multimídia”, oferecendo capacidades de descrição de conteúdo multimídia para uma grande variedade de aplicações;
- ISO/IEC 21000 (MPEG-21), “Arcabouço Multimídia”, provendo um arcabouço de integração para os padrões MPEG anteriores e outras tecnologias como o IPMP (*Intellectual Property Management and Protection*), e Adaptação de Conteúdo.

Mais recentemente, em Julho de 2005, o MPEG reconheceu que a abordagem de padronização utilizada anteriormente baseada em grandes projetos tais como os mencionados acima não era mais adequada, e, ao invés disso, diversos padrões menos abrangentes deveriam ser desenvolvidos. Além disso, a maioria destes padrões não estavam diretamente relacionados a nenhum dos grandes projetos do MPEG já existentes. Desta forma, um novo tipo de estrutura de padrões MPEG foi

criado para complementar o conjunto disponível de grandes projetos (PEREIRA, 2006):

- ISO/IEC 23000 (MPEG-A), “Formatos de Aplicação Multimídia”, definindo formatos de arquivos dirigidos por aplicações entre os padrões MPEG;
- ISO/IEC 23001, “Tecnologias de Sistemas MPEG”, definindo ferramentas de sistema que podem ser usados através de todos os grandes padrões MPEG;
- ISO/IEC 23002, “Tecnologias visuais MPEG”, definindo ferramentas de codificação visual que podem ser usados entre todos os grandes padrões MPEG;
- ISO/IEC 23003, “Tecnologias de áudio MPEG”, definindo ferramentas de codificação de áudio que podem ser usadas entre todos os grandes padrões MPEG;
- ISO/IEC 23004 (M3W), “MPEG *Multimedia Middleware*”, melhorando a portabilidade e a interoperabilidade de aplicações através da especificação de um conjunto de APIs dedicadas a funções multimídia bem como o oferecimento de uma maneira padronizada de implementação destas APIs. Este padrão também é conhecido por MPEG-E.

No contexto desta dissertação, será dado foco na ISO/IEC 23004, que irá proporcionar (GELISSEN, 2004):

- Aos desenvolvedores de aplicação desenvolver rapidamente e fazer facilmente a manutenção de aplicações multimídia;
- Aos provedores de aplicação e serviço limitar o custo do provisionamento de aplicações;
- Aos fabricantes prover um ambiente de desenvolvimento de aplicações interoperáveis.

Com o intuito de oferecer qualidade de serviço às aplicações multimídias, o MPEG *Multimedia Middleware* (M3W) define, além de APIs multimídia, mecanismos para atender propriedades não-funcionais como disponibilidade, segurança e mecanismos para gerenciamento do sistema.

Para satisfazer os objetivos citados anteriormente, serão requeridas as seguintes APIs (GELISSEN, 2004):

- APIs funcionais para permitir às aplicações acessar as funções multimídia;
- APIs para gerenciar o comportamento da execução como por exemplo: recursos (memória, energia, processador, uso da rede) de funções multimídia;
- APIs para gerenciar o isolamento (por exemplo: proteção de memória) de funções multimídia;
- APIs para gerenciar o ciclo de vida de componentes do MPEG *Multimedia Middleware* (M3W) (por exemplo: identificação, *download*, instalação, ativação, desativação, desinstalação).

Em geral, estas APIs têm como objetivos (GELISSEN, 2004):

- Permitir a identificação de componentes necessários por uma aplicação, além de permitir a localização, a recuperação, o teste de consistência, o registro, a instanciação e a remoção de componentes. Todas estas atividades juntas constituem o gerenciamento do ciclo de vida dos componentes;
- Permitir mecanismos de gerenciamento de recursos: avaliação dos recursos atualmente disponíveis no dispositivo em relação aos recursos requeridos pela aplicação e o contexto da operação do dispositivo bem como do usuário. Estes mecanismos juntos permitem uma política de gerenciamento de recursos (por exemplo: prioridade, degradação adequada da aplicação em execução, etc.);
- Permitir mecanismos de gerenciamento de falhas para fazer o tratamento de exceções, prevenir a propagação de falhas de *software* e o isolamento de componentes de *software* que excedem o espaço de execução alocados a eles.

APIs de funcionalidade Multimídia

As APIs multimídia têm a intenção de disponibilizar o acesso requerido das aplicações às funcionalidades multimídia fornecidas pelos componentes. Uma lista não exaustiva de tais APIs pode ser encontrada abaixo:

- Serviços de processamento de mídia (incluindo codificação, decodificação e transcodificação);
- Serviços de distribuição de mídia (através de arquivos, fluxos e mensagens);
- Serviços DRM;
- Acesso a Metadados; Edição de Metadados; Procura de Metadados.

APIs de Gerenciamento

Para permitir o tratamento e o gerenciamento de componentes, são requeridos procedimentos para lidar com processos essenciais como localização de serviço, decisão/validação, *download*, instanciação e desinstalação/remoção. O processo de decisão, por exemplo, tem de validar se o sistema resultante depois da instanciação da nova, atualizada ou estendida funcionalidade ainda provê um sistema válido em relação aos requisitos base. A composição do serviço incluindo uma análise profunda de todos os possíveis efeitos colaterais serve como base para este processo. Estes procedimentos e processos de decisão, e a API de gerenciamento associada, precisam ser suplementadas com modelos de negócio e regras para permitir que a troca de um componente aconteça.

APIs de Operação

As APIs de operação levam em conta as funcionalidades extras requisitadas que são essenciais para a execução confiável da aplicação que requisita os serviços dos componentes, que por sua vez, requisitam serviços da plataforma. Neste contexto, os componentes fazem a mediação entre a funcionalidade requisitada pela aplicação (baseada em requisitos e premissas) e o que é possível na plataforma (baseada em conhecimento e fatos). Estas funcionalidades extras incluem, por exemplo (GELISSEN, 2004):

- Gerenciamento de recursos (ciclos de processamento, memória, largura de banda, energia, estado real da interface de usuário, etc.);
- Operação em tempo real e garantias (por exemplo para permitir acesso a escalonadores).

Em geral estes requisitos requerem uma abordagem de duas camadas em relação ao *middleware*, a camada superior sendo a interface para a aplicação, a camada inferior sendo a interface para o sistema operacional (abstraido) e serviços de sistema para permitir para o monitoramento e controle das funcionalidades extras mencionadas acima. Tais APIs de Operação estão relacionadas ao:

- gerenciamento de recursos (energia, processador, memória);
- gerenciamento de falha ou exceção.

Abordagem utilizada

Tradicionalmente os dispositivos de eletrônicos de consumo, uma das principais áreas de desenvolvimento dos padrões MPEG, não possuíam *software* e todas as funcionalidades eram implementadas em *hardware*. De 1980 em diante o *software* foi introduzido em partes desses sistemas começando na parte de controle. Gradualmente, cada vez mais módulos de *software* foram sendo introduzidos em diversas partes desses sistemas. Ao mesmo tempo, as partes que diferenciam os produtos entre os diferentes fabricantes eram cada vez mais obtidas de fabricantes externos. Nos últimos anos, o papel dos diferentes agentes neste campo tem mudado, os fabricantes de semicondutores estão focados em fornecer uma plataforma genérica que pode servir um grande mercado e os fabricantes de eletrônicos de consumo estão focados em oferecer aos seus clientes funcionalidades e serviços que diferenciam seus produtos do que os seus competidores oferecem. Todos esses desenvolvimentos têm feito dois agentes tradicionais (fabricantes de semicondutores e eletrônicos de consumo) menos dependente um do outro e foi aberta a oportunidade para novos agentes neste campo para o chamado *middleware* que faz a ponte entre a plataforma oferecida pelos fabricantes de semicondutores e as aplicações que os fabricantes de eletrônicos de consumo oferecem aos seus clientes (GELISSEN, 2004) .

Esta situação não é diferente dos desenvolvimentos na indústria de semicondutores alguns anos atrás que resultaram no estabelecimento de um mercado de módulos de terceiros que são integrados nos produtos finais das companhias de semicondutores. Numa situação de mercado como esboçado acima os papéis e responsabilidades de diferentes provedores precisam ser claros para evitar a sobreposição ou a falta de

funcionalidades. A Figura 10 ilustra os diferentes papéis dos três atores que juntos compõem o sistema.

Figura 10 – Camadas Arquiteturais que compõem o Sistema segundo a abordagem do MPEG *Multimedia Middleware* (GELISSEN, 2004).

Várias corporações proponentes de padrões de-facto da indústria e/ou organizações de padronização internacional estão ativos nas camadas de interface entre as três áreas arquiteturais acima mencionadas por diferentes razões:

- A interface da camada de aplicação (API entre o *middleware* e a aplicação) objetiva a unificação de interfaces para permitir que os serviços operem numa larga base instalada de dispositivos;
- A interface da camada de plataforma (API entre a plataforma e o *middleware*) objetiva a unificação dessas interfaces para permitir que as plataformas sejam usadas numa larga variedade de aplicações. Existem muitas iniciativas como o CELF e a UHAPI da Samsung e Philips.

Uma lista não-exaustiva de exemplos de entidades que poderiam ser posicionadas nestas três camadas de sistema são:

- Camada de Aplicação
 - EPG
 - Programa de Controle do DVD

- Programa de Controle da TV
- Gerenciamento de armazenamento
- Camada do *Middleware*
 - Suporte a gerenciamento de conteúdo
 - PVR
 - Decodificador e codificador de mídia
 - Filtros de mídia
- Camada de Plataforma
 - Sistema Operacional
 - *Streaming (Software)*
 - Serviços de Sistema
 - *Drivers* de dispositivos
 - Decodificador e codificador de mídia

Vários exemplos mencionados já são resultado das atividades de padronização internacional ou de-facto da indústria e são como tais potencialmente disponíveis de múltiplas fontes. Além disso, deve ser notado que em relação aos codificadores e decodificadores de mídia, posicionado tanto na camada do *middleware* como na camada de plataforma nas listas acima, para o futuro próximo estas funções provavelmente ainda dependem muito dos ASICs (*Application-Specific Integrated Circuit*) e como tais estão posicionados na camada de plataforma. É esperado entretanto que haja uma mudança gradual dessas funções em direção ao *software* em processadores de mídia programáveis e/ou dispositivos de computação reconfigurável e tais funções sejam parte da camada de *middleware*. O posicionamento destas funções nos dispositivos atuais irá, entretanto, sempre ser um *trade-off* econômico e a mudança para os processadores de mídia programáveis e dispositivos de computação reconfigurável só irá acontecer quando a aplicação necessitar (devido a mudança no mix da aplicação requerendo múltiplos

decodificadores em diferentes configurações), ou quando se tornarem mais barato. O último caso pode parecer estranho mas poderia acontecer no caso dos dispositivos de computação reconfigurável.

A abordagem utilizada pelo MPEG *Multimedia Middleware* (M3W) é a utilização de componentes de *software* para encapsular funcionalidade. Esta abordagem foi estendida para suportar uma grande variedade de propriedades não-funcionais.

Modelo Conceitual do MPEG *Multimedia Middleware*

O MPEG *Multimedia Middleware* (M3W) utiliza o modelo conceitual ilustrado na Figura 11.

**Figura 11 – Modelo Conceitual do MPEG *Multimedia Middleware* (M3W).
Modificado a partir de (ISO/IEC, 2004)**

Um sistema consiste de uma plataforma com um conjunto de *softwares* que oferecem funções a um usuário. Uma aplicação é a interface primária com o usuário.

Componentes e serviços suportam as aplicações provendo uma camada extra de funcionalidade sobre a plataforma. Aplicações usam as funcionalidades providas pelos componentes e serviços. Um componente é uma unidade de desenvolvimento e pode conter serviços. Os serviços são uma unidade de execução e oferecem funcionalidade a aplicações – instâncias de serviços podem ou não ter estados, e, caso não tenha, são similares a bibliotecas (ISO/IEC, 2004).

Relação entre UHAPI e MPEG *Multimedia Middleware* (M3W)

A ISO/IEC 23004 é constituída das seguintes partes, como mostra a Tabela 1:

Tabela 1 – Partes da ISO/IEC 23004 - MPEG *Multimedia Middleware*

MPEG <i>Multimedia Middleware</i>		
ISO/IEC 23004 – 1	Parte 1 – Arquitetura	Define a arquitetura da tecnologia M3W.
ISO/IEC 23004 – 2	Parte 2 – API Multimídia	Define a API multimídia do M3W.
ISO/IEC 23004 – 3	Parte 3 – Modelo de Componentes	Define o modelo de componentes do M3W.
ISO/IEC 23004 – 4	Parte 4 – Gerenciamento de recursos	Define o arcabouço de gerenciamento de recursos e qualidade do M3W.
ISO/IEC 23004 – 5	Parte 5 – <i>Download</i> / Distribuição	Define a arquitetura de <i>download</i> do M3W.
ISO/IEC 23004 – 6	Parte 6 – Gerenciamento de Falhas	Define a arquitetura de gerenciamento de falhas do M3W.
ISO/IEC 23004 – 7	Parte 7 – Gerenciamento de Integridade	Define a arquitetura de gerenciamento de integridade do M3W.
ISO/IEC 23004 – 8	Parte 8 – <i>Software</i> de Referência e Conformidade	Provê uma implementação de referência para as partes 1 a 7 deste padrão.

Como pode ser observado, a parte 2 (API Multimídia) do M3W corresponde à família de APIs definida pela UHAPI.

2.4.6 O arcabouço DirectShow

O DirectShow é um arcabouço multimídia produzido pela Microsoft para desenvolvedores de *software* executarem várias operações com arquivos de mídia.

Baseado no arcabouço COM (*Component Object Model*) da Microsoft, o DirectShow fornece uma interface comum para mídia através de muitas das linguagens de programação da Microsoft.

O arcabouço DirectShow divide o processamento de tarefas multimídia em um conjunto de passos conhecidos como filtros. Cada filtro representa um estágio no processamento dos dados, por exemplo, leitura de um arquivo ou câmera, decodificação, transformação ou renderização. Os filtros possuem um número de pinos de entrada e saída, que representam pontos de conexão. Desta forma, os filtros podem se conectar entre si. Estes filtros podem ser conectados de muitas formas diferentes para tarefas diferentes para criar um grafo de filtros. Os desenvolvedores podem adicionar alguns filtros – possivelmente personalizados – ao grafo, e então renderizar o arquivo, a URL (*Uniform Resource Locator*), ou câmera. Durante o processo de renderização, o grafo procura, no Registro do Windows, por filtros registrados e faz a construção do grafo de filtros baseado na localização fornecida. Assim, os filtros são conectados, e, à requisição do usuário (reproduzir, parar, pausar, etc), o grafo é executado.

A Figura 12 mostra o exemplo de um grafo construído utilizando o arcabouço DirectShow.

Figura 12 – Exemplo de grafo construído. Modificado a partir de (DIRECTSHOW, 2008).

2.4.7 O arcabouço GStreamer

O GStreamer (GSTREAMER, 2007) é um arcabouço multimídia, em análise pelo CELF, que permite a criação de aplicações de fluxos multimídia. A Figura 13 mostra a estrutura de *software* para uma aplicação que utiliza o GStreamer para fazer o processamento de fluxos multimídia.

Sendo um arcabouço multimídia, o GStreamer pode vir a ser um elemento de infraestrutura que ofereça suporte a modelos de componentes como os da UHAPI e o M3W.

Figura 13 – Estrutura de *software* para o GStreamer

Fundamentalmente, o GStreamer incorpora várias das idéias do DirectShow, sendo um arcabouço baseado em *plugins* que irão prover vários codecs e outras funcionalidades. Os *plugins* podem ser ligados e organizados em uma *pipeline*. Esta *pipeline* define o fluxo de dados através da construção de um grafo composto por *plugins* de processamento, conforme ilustrado pela Figura 14.

Figura 14 – Exemplo de *pipeline* do GStreamer

Cada *plugin* é um bloco de processamento visto como uma caixa preta, não importando se a implementação está em *software* ou *hardware*.

Uma das grandes responsáveis no consumo de processamento são as cópias de memória no fluxo de dados. No GStreamer, estas cópias são evitadas pois os dados passam entre os blocos através de ponteiros de regiões de memória compartilhada.

A exemplo de outros trabalhos correlatos apresentados neste capítulo como o OpenMAX, a UHAPI e o M3W, o GStreamer também é uma solução de *software* aberto e livre.

2.4.8 O grupo Khronos

Fundado em 2000 por companhias como a 3Dlabs, ATI, Discreet, Evans & Sutherland, Intel, NVIDIA, SGI e Sun Microsystems, o grupo Khronos é um consórcio fundado com o objetivo de criar APIs abertas e padronizadas para permitir a reprodução e a distribuição de conteúdo multimídia numa grande variedade de plataformas e dispositivos (KHRONOS, 2007a).

Atualmente existe uma variedade de soluções desenvolvidas, cada uma projetada para acelerar aplicações multimídia como os processadores com extensões multimídia específicas, arquiteturas com múltiplos processadores e decodificadores de vídeo usando *hardware* dedicado. Um dos maiores desafios com todas estas variantes arquiteturais é desenvolver um código-fonte eficiente capaz de explorar todo o potencial da plataforma de *hardware*. A consequência disso é que alguns trechos de código são escritos em *assembly* e a proliferação de diferentes soluções de *hardware* faz com que o *software* tenha que ser reescrito e otimizado para cada plataforma (KHRONOS, 2007a).

Com base neste fato, o grupo Khronos criou uma das suas principais APIs: o OpenMAX.

O OpenMAX é uma API livre de *royalties*, multiplataforma, com o objetivo de definir um conjunto de padrões de APIs abertas para aplicações multimídia, reduzindo custos e a complexidade de portar *softwares* multimídia para novos processadores e arquiteturas.

Assim, especificações de codecs multimídia, bibliotecas gráficas e outras funções para vídeo, áudio e imagens podem possibilitar aos desenvolvedores de *software* a diferenciação de seus produtos ao invés de implementar novamente funcionalidades multimídia comuns.

O OpenMAX define três camadas (KHRONOS, 2007a): o OpenMAX AL, o OpenMAX IL e o OpenMAX DL, que serão detalhadas nos subtópicos a seguir.

A Interface de Programação OpenMAX AL

A API OpenMAX AL (*Application Layer*) define um conjunto de APIs para fornecer uma interface padronizada entre uma aplicação e o *middleware* multimídia, de forma que o *middleware* multimídia forneça os serviços necessários para executar a funcionalidade esperada da API. Conforme pode ser observado na Figura 15, as aplicações multimídia podem ser escritas de forma portátil, independentemente da plataforma de suporte.

A Interface de Programação OpenMAX IL

A API OpenMAX IL (*Integration Layer*) oferece uma interface de baixo nível para codecs de áudio e vídeo usados em dispositivos embarcados e/ou móveis. Possibilita às aplicações e arcabouços multimídia a habilidade de fazer interface com codecs multimídia e componentes suportados (isto é, pinos de entrada e pinos de saída) de maneira unificada. Os codecs podem ser qualquer combinação de *hardware* ou *software* e são completamente transparentes para usuário. O principal objetivo do OpenMAX IL é fornecer codecs com um grau de abstração do sistema. Conforme pode ser observado na Figura 15, os componentes multimídia podem ser integrados em grafos flexíveis para processamento de fluxos multimídia.

A Interface de Programação OpenMAX DL

A API OpenMAX DL (*Development Layer*) contém um conjunto de funções de áudio, vídeo e imagens que podem ser implementadas e otimizadas em novos processadores pelos fabricantes dos mesmos. Inclui funções de processamento de sinais como FFTs e filtros, primitivas de processamento de imagens, primitivas de processamento de vídeo para possibilitar implementações otimizadas de codecs como MPEG-4, H.264, MP3, AAC e JPEG. Conforme pode ser observado na Figura 15, os componentes multimídia podem ser escritos usando primitivas para portabilidade entre diversas arquiteturas.

Figura 15 – Estrutura de *software* para o OpenMAX. Modificada a partir de (KHRONOS, 2007a)

2.5 Conclusões

Neste capítulo foi apresentado uma revisão da literatura sobre arquiteturas de *software* de terminais de acesso, oferecendo ao leitor os subsídios necessários para o entendimento do desenvolvimento deste trabalho. Foram apresentadas a arquitetura de *software* tradicional de terminais de acesso e a arquitetura do Terminal de Acesso de Referência do SBTVD, bem como as diferenças entre ambas. Adicionalmente, também foram apresentadas alguns trabalhos correlatos na área de sistemas embarcados multimídia, como são os casos do CELF, da UHAPI, do M3W e do OpenMAX, que foram selecionados de acordo com os principais requisitos da Arquitetura do Terminal de Acesso de Referência. A Figura 16 ilustra como se relacionam os trabalhos detalhados neste capítulo, de forma a elucidar onde estão inseridos.

Figura 16 – Relacionamento entre os trabalhos detalhados no Capítulo 2

Como pode ser observado na Figura 16, as camadas de Aplicações e do *Middleware* utilizam o conjunto de APIs padronizadas que abstraem os recursos da plataforma. Esta camada de abstração para as camadas superiores é alvo das especificações da IOS, do OpenMAX AL, do M3W e da UHAPI, sendo que esta última também faz parte do padrão M3W.

Para implementar as funções especificadas pela IOS, OpenMAX AL, M3W ou UHAPI, está uma camada composta pelas bibliotecas de baixo nível e pelo núcleo do sistema operacional. Dentro desta camada, existe a possibilidade de utilização de arcabouços multimídia baseados em componentes como o GStreamer e o DirectShow. Estes arcabouços multimídia, por sua vez, podem utilizar as interfaces de programação definidas pelo OpenMAX IL que abstraem os codecs do sistema. Já os componentes que implementam as funções do OpenMAX IL podem fazer uso da API OpenMAX DL, que contém diversas primitivas de processamento muito utilizadas nos processos de codificação e decodificação de áudio, vídeo e imagens.

CAPÍTULO 3

ANÁLISE E COMPARAÇÃO DE TÉCNICAS, FERRAMENTAS E ARCABOUÇOS

Nesse capítulo, faz-se uma análise das tecnologias descritas no Capítulo 2 que podem ser utilizadas para compor a arquitetura de *software* de um terminal de acesso. Além disso, faz-se um levantamento comparação dos pontos fortes e dos pontos fracos das abordagens baseadas em componentes de *software*. Desta forma, as seguintes tecnologias são analisadas:

- A IOS (*Interface for Operating Systems*);
- O padrão aberto UHAPI;
- O padrão MPEG *Multimedia Middleware* (M3W);
- As APIs do grupo Khronos;
- O arcabouço GStreamer.

Apesar de existir diversas soluções em *software*, ainda é preciso entender os pontos fortes e fracos de cada uma delas. É importante entender o que estas tecnologias têm a oferecer de forma a analisar como a arquitetura de *software* do SBTVD se relaciona com estes trabalhos e como a arquitetura proposta Arquimedia pode aproveitar os esforços destes grupos.

Cada uma destas tecnologias será avaliada nos seguintes aspectos:

- **Funcionalidades cobertas:** Este aspecto tem por objetivo esclarecer quais são as funcionalidades cobertas por estas tecnologias. Algumas tecnologias podem cobrir uma determinada funcionalidade, enquanto outras não. Ou então, algumas podem ser mais completas em relação a uma determinada funcionalidade, ao passo que outras podem ser mais restritas;
- **Design de projeto:** este aspecto tem como objetivo oferecer uma visão da abordagem utilizada na concepção destas tecnologias, já que cada uma possui as suas próprias prioridades e peculiaridades. Por exemplo, algumas

tecnologias podem utilizar uma abordagem baseada em componentes de *software*, enquanto outras não. Sendo assim, a idéia é analisar cada tecnologia apresentando os principais conceitos utilizados e o funcionamento de pontos importantes como o modelo de *threads*, o mecanismo de notificações de eventos e o tratamento de erros;

- **Escalabilidade:** Sendo este aspecto um dos principais requisitos da Arquitetura do Terminal de Acesso de Referência, são analisadas os mecanismos que proporcionam escalabilidade em cada um dos trabalhos mencionados, apresentando a capacidade de expandir a funcionalidade oferecida ou cumprir suas funções de acordo com o perfil do terminal de acesso;
- **Usabilidade:** A tarefa de projetar APIs fácil utilização é essencial para aumentar a velocidade no desenvolvimento de produtos. A usabilidade de APIs está associada à medida de seu grau de compreensão e de conveniência, além do seu tempo de aprendizado. Uma boa documentação pode ser essencial para contribuir para aumentar o grau de compreensão e diminuir o tempo de aprendizado (BORE, C; BORE, S, 2005).

3.1 Avaliação da IOS (*Interface for Operating System*)

Nas próximas seções será realizada uma avaliação da IOS (*Interface for Operating System*), considerando os aspectos discutidos no Capítulo 2 e apresentadas na introdução deste capítulo.

3.1.1 Funcionalidades cobertas da IOS

Conforme dito anteriormente, a IOS define nove áreas funcionais como mostra a Tabela 2.

Tabela 2 – Descrição geral das funções cobertas pela IOS

Área Funcional	Descrição
IOS-PWR	Controle de Energia
IOS-SND	Áudio
IOS-GRA	Gráfico
IOS-TV	<i>Hardware</i> de TV
IOS-SEC	Segurança
IOS-QOS	Qualidade de Serviço
IOS-NET	Comunicação em Rede
IOS-IOD	Interação com o usuário
IOS-BASE	<i>Hardware</i> Básico de Computação

3.1.2 Design de projeto da IOS

A IOS não foi projetada como sendo uma tecnologia baseada em componentes de *software*. A principal contribuição da IOS está no conjunto de interfaces de *software* recomendado. Para cada área funcional, foi escolhida e/ou desenvolvida uma interface de *software* que fosse adequada para sistemas embarcados multimídia. Desta forma, a IOS procura fazer recomendações a fabricantes e desenvolvedores de acordo com os principais requisitos da Arquitetura do Terminal de Acesso de Referência.

3.1.3 Escalabilidade da IOS

Na IOS, a escalabilidade está no fato de poder adicionar ou remover áreas-funcionais.

A aplicação escrita sobre o conjunto de interfaces de *software* definido pela IOS está sujeito às modificações de funções de cada uma das áreas funcionais. Desta forma, caso o desenvolvedor faça uso de uma determinada função de uma área-funcional e seja lançada uma nova versão desta área-funcional na qual esta função não esteja mais presente, a aplicação deverá sofrer alterações caso seja necessário utilizar as novas funções incluídas nesta nova versão.

3.1.4 Usabilidade da IOS

A IOS forma um conjunto de APIs bastante heterogêneo. Algumas áreas funcionais como a IOS-GRA, IOS-SND, IOS-TV, IOS-SEC, IOS-NET e IOS-BASE foram escolhidas a partir de projetos já existentes. Outras como a IOS-PWR, IOS-QOS e IOS-IOD foram desenvolvidas pelo consórcio de universidades que participou do desenvolvimento da Arquitetura do Terminal de Acesso de Referência.

Cada área funcional possui uma filosofia diferente e força com que o desenvolvedor tenha que aprender o funcionamento de cada uma delas separadamente, o que dificulta o processo de desenvolvimento.

Este processo de aprendizagem inclui muitos experimentos para descobrir como funciona uma determinada API. Isto devido principalmente à falta de uma documentação adequada. E como cada área funcional possui uma filosofia de programação diferente, o tempo gasto na aprendizagem do funcionamento de uma área funcional geralmente não será aproveitado para entender o funcionamento de outra área funcional.

3.2 Avaliação da UHAPI

3.2.1 Funcionalidades cobertas da UHAPI

A UHAPI define uma família de APIs para funcionalidade de áudio e vídeo, ou seja, estão cobertas apenas funcionalidades multimídia. Em comparação com as áreas-funcionais da IOS estariam cobertas apenas as funcionalidades previstas para a IOS-SND, IOS-GRA e IOS-TV.

3.2.2 *Design* de projeto da UHAPI

A UHAPI tenta reproduzir as características de fluxo multimídia simplificando a criação de sistemas de *software* para uma série de transformações chamadas de **componentes lógicos** e a sequência na qual estes componentes são conectados.

As APIs do padrão UHAPI são constituídas de componentes lógicos.

As funções executadas por cada componente lógico em um fluxo de dados em particular são determinadas por um ou mais atributos que são controlados por uma

interface. Seguindo este raciocínio, um componente lógico pode ser visto como uma caixa preta com vários atributos usados para configurar as funções a serem executadas quando os dados forem processados pelo mesmo.

Assim, o componente lógico pode ser definido como caixa preta que executa transformações no fluxo de dados. Cada componente lógico possui um ou mais papéis. O papel é uma classe abstrata em termos UML, ou seja, que descreve um comportamento sem referenciar uma implementação particular. Usualmente, um papel fornece uma ou mais interfaces de controle e é cliente de uma ou mais interfaces de notificação. A Figura 17 mostra o esquema de um componente lógico.

Figura 17 – Ilustração de um Componente Lógico da UHAPI. Modificado a partir de (UHAPI Forum, 2004)

A UHAPI define um documento de especificação para cada componente lógico.

Os componentes de implementação devem implementar componentes lógicos. O implementador tem liberdade para escolher a arquitetura dos componentes de implementação.

A implementação pode, por exemplo, usar fluxos em *hardware*, fluxos em *software* ou uma combinação. Além disso, não é necessário que a relação entre componente lógico e componente de implementação seja de um para um. A Figura 18 ilustra isso.

Figura 18 – Relação entre componentes lógicos e componentes de implementação. Modificado a partir de (UHAPI Forum, 2004)

Ao conjunto predefinido de componentes lógicos interconectados para executar uma aplicação em particular dá-se o nome de **Caso de Uso** (vide Figura 19). O programador irá controlar as ações e as seqüências de conexões dos componentes lógicos no nível de casos de uso.

Figura 19 – Ilustrando o conceito de Caso de Uso da UHAPI. Modificada a partir de (UHAPI Forum, 2004).

As **interfaces** são conjuntos de funções que controlam o que os componentes lógicos executam. Os programadores conseguem enxergar somente a interface, mas não detalhes da implementação. Desta forma, os programadores podem utilizar a mesma programação para várias plataformas, sendo que em cada plataforma, o componente lógico pode executar uma determinada funcionalidade de forma diferente (UHAPI Forum, 2004).

As interfaces estão agrupadas em componentes lógicos. Cada componente lógico é uma coleção de interfaces que fornece acesso a uma funcionalidade.

Os programadores da UHAPI controlam as ações e as conexões dos componentes no nível de casos de uso. Para seleccionar casos de uso apropriados para uma aplicação, o programador utiliza um componente especial denominado de **Gerenciador de Conexões**. Para cada caso de uso seleccionado, o Gerenciador de Conexões utiliza todos os componentes necessários para o caso de uso referenciado, faz as conexões entre os componentes e os configura.

Uma **Plataforma** UHAPI pode ser implementada usando qualquer combinação de *hardware* e/ou *software* para tratar funções de fluxo multimídia para um produto de eletrónico de consumo em particular. A plataforma pode ser parte de um sistema maior ou implementada por várias partes distribuídas do sistema. O importante é que os desenvolvedores não precisam saber como a plataforma é implementada, só precisam saber como usar a UHAPI.

Cada plataforma define o que está implementado pelos casos de uso oferecidos, pelos componentes lógicos nos casos de uso, conexões entre componentes, e qualquer especialização de componentes lógicos.

A aplicação controla qual caso de uso está seleccionado num determinado instante.

A Figura 20 mostra como se relacionam os conceitos de componente lógico, caso de uso, interface, gerenciador de conexões e plataforma.

Figura 20 – Ilustração dos principais elementos da UHAPI. Extraído de (UHAPI Forum, 2004).

As APIs da UHAPI são baseadas em um subconjunto do COM. O princípio chave do COM e de outros modelos de componentes são as interfaces.

Interfaces são orientadas a objeto no sentido de que suas operações devem ser implementadas por uma classe e passada uma referência para uma instância particular desta classe quando invocada. Entretanto, o código que usa o componente não se refere diretamente a qualquer objeto. Ao invés disso, as operações de uma interface suportada por um objeto são invocadas através do que é conhecido como ponteiro para interface. O conceito conhecido como interface de navegação torna possível para o usuário obter um ponteiro para qualquer interface suportada pelo objeto.

O COM também define um formato de tempo de execução para ponteiros de interface. O que um ponteiro de interface realmente referencia é o nó de interface, que por sua vez, contém um ponteiro para uma tabela de ponteiros de função, chamada de VTABLE (*Virtual Table*), conforme a Figura 21. O uso de VTABLES é idêntico a maneira que muitos compiladores C++ implementam funções virtuais. Este *overhead* de tempo é muito modesto. O *overhead* de memória também pode ser aceitável, exceto talvez para os sistemas embarcados de recursos mais restritos (LÜDERS, 2003).

Figura 21 – Formato típico dos nós de interface COM (LÜDERS, 2003).

Para a maioria de sistemas de tempo real, uma preocupação mais séria do que o modesto *overhead* é que a navegação através de interfaces introduz uma possível fonte de erros em tempo de execução. Se o usuário requisitar um objeto para um ponteiro para uma interface que o objeto não suporta, isto não será detectado em tempo de compilação. Pode ser argumentado, de fato, que essa é a principal diferença entre navegação através de interfaces e herança de interfaces na tradicional

programação orientada a objeto. Este pode ser visto como um preço necessário para pagar pela desejável dependência reduzida em tempo de compilação entre componentes.

Conforme dito anteriormente, o código de um componente COM é implementado em classes. O COM define uma política para instanciação, que tem a intenção de assegurar que diferentes componentes possam ser instalados num sistema em diferentes instantes. Quando um componente está instalado, informações sobre o mesmo precisam estar registrados no sistema.

Assim, a criação de uma instância envolve a busca de informação sobre classes registradas e a possibilidade de carregamento de código. Isto leva a um perceptível *overhead* quando comparado a instanciação no C++. Além disso, este *overhead* irá variar, dependendo se o código que implementa a classe já foi ou não carregada. Esta variabilidade pode ser eliminada, entretanto, projetando o *software* de forma que todos os componentes que possam ser usados sejam carregados no início do programa.

Um benefício do COM na sua maneira de criar instâncias é que o código que implementa o componente pode ser construído independentemente de qualquer código que utiliza o componente. Como a instanciação envolve a transmissão da identidade da classe desejada como parâmetro para a operação do sistema, isto é uma possível fonte de erros em tempo de execução, que não está presente durante a instanciação na programação orientada a objeto tradicional, já que a tentativa para instanciar a classe que não existe resulta em erro de compilação neste caso (LÜDERS, 2003).

Modelo de *Threads* da UHAPI

Para a UHAPI, uma determinada função pode ser:

- *Thread-safe*: indica que uma função pode ser acessada concorrentemente por múltiplas *threads*;
- *Single-threaded*: indica que a função pode ser acessada por no máximo uma *thread* por vez.

Pode-se especificar se uma entidade é *thread-safe* ou *single-threaded* no nível de plataforma, componente lógico, papel, interface e função.

Ao decidir se uma entidade mencionada anteriormente deve ser *thread-safe* ou *single-threaded*, existem dois aspectos importantes a serem considerados. Uma é a facilidade de implementação e a facilidade de teste, que são vantagens oferecidas por uma interface *single-threaded*. O outro aspecto é a facilidade de uso, que é a vantagem oferecida por uma interface *thread-safe*.

As APIs da UHAPI tentam encontrar o melhor balanceamento entre ambos. A aplicação cliente geralmente pode resolver problemas de sincronização de uma melhor forma do que a API já que possui um maior conhecimento sobre os exatos requisitos de sincronização do que a implementação da API. O desempenho também é um aspecto importante neste contexto. Algumas vezes, uma função *single-threaded* pode ser melhor em termos de desempenho pelo fato de não haver um *overhead* de sincronização enquanto, em outras ocasiões, uma função *thread-safe* pode ter um melhor desempenho por permitir acesso concorrente.

Os componentes lógicos são, em princípio, *single-threaded* por permitir uma implementação mais simples e pelo fato de, em geral, não ser necessário tornar todo o componente *thread-safe*. A idéia é implementar mecanismos *thread-safe* somente onde é necessário.

Os métodos Get são *thread-safe* por padrão, mesmo quando a interface da qual estes métodos fazem parte não são *thread-safe*. Isto devido à facilidade de implementar esta regra.

Os métodos de notificação não devem ser bloqueantes e devem ser serializados para, evitar que a aplicação seja desnecessariamente complexa, como será visto mais adiante.

Notificações de Eventos da UHAPI

A UHAPI oferece interfaces de controle que permitem controlar a funcionalidade provida pela plataforma. Além de controlar a funcionalidade, as aplicações geralmente necessitam executar ações em reação a eventos que ocorrem na plataforma. Exemplos de eventos são mudanças de estado, detecção de mudanças de

sinal, complemento de ações assíncronas, etc. Existem duas diferentes maneiras de tratamento de eventos suportadas por uma API (ISO/IEC, 2006):

- *Pooling*: a aplicação utiliza funções oferecidas pela API para inspecionar por mudanças de *status* e, assim, detectar a ocorrência de eventos e executar ações;
- Notificação: a plataforma tem a responsabilidade de detectar e notificar a ocorrência de um determinado evento. Ao receber a notificação, o cliente pode tomar a ação imediatamente. Como a plataforma gera o evento, geralmente esta é a solução mais eficiente em termos de desempenho.

A UHAPI suporta o tratamento de eventos por notificações. Nesta abordagem, existe uma correspondência de um-para-um entre funções em uma interface de notificação e tipos de eventos sendo notificados através da interface de notificação.

Por razões de eficiência, não é desejável que as aplicações recebam notificações de todos os eventos. As aplicações necessitam de alguma maneira para indicar que desejam receber notificações de apenas alguns eventos. Este mecanismo é geralmente conhecido por registro de eventos. Registrando um evento, uma aplicação indica que deseja receber notificações de um determinado evento e, tirar o evento do registro indica que a aplicação não deseja mais receber notificações de um evento.

Estas notificações são implementadas como funções de *callback*, sendo que a implementação da API deve garantir que para cada registro de evento, todas as notificações sejam serializadas, ou seja, a próxima função de notificação não deve ser chamada enquanto a função de notificação anterior tenha retornado. Esta regra tem como objetivo evitar que a implementação de uma aplicação tenha todas as notificações de interface *thread-safe*, o que pode ser desnecessariamente complexo.

Tratamento de Erros na UHAPI

A UHAPI oferece os seguintes mecanismos para informar a aplicação da ocorrência de um erro (ISO/IEC, 2006):

- **Código de retorno de uma função** – toda função definida na UHAPI possui um tipo de retorno. Toda vez que um erro é detectado durante a execução de

uma função, o erro é reportado através de um código de retorno, que deve ser especificado como parte do comportamento de uma função;

- **Função de notificação de erros de um componente lógico** – alguns erros de componentes lógicos não são diretamente relacionados a chamada de uma função, entretanto possuem um significado no domínio de um componente lógico. Estes erros estão geralmente relacionados a qualidade de um fluxo de áudio ou vídeo. Exemplos de tais erros são: erro de paridade ou perda de sincronismo. Estes erros não deveriam ocorrer a princípio, mas o cliente do componente lógico é capaz de tratar o erro, isto é, tentar recuperar ou ir para um estado definido. Para informar o cliente destes erros, o componente lógico define uma notificação para o qual pode-se registrar;
- **Handler para tratamento de erros não fatais da plataforma** – a plataforma como um todo oferece um *handler* para erros não fatais. Estes erros são não-fatais no sentido de que o sistema pode continuar depois da ocorrência. A razão para notificar estes eventos é permitir que seja gerado um log destes eventos para testes e serviços;
- **Handler para tratamento de erros fatais da plataforma** – a plataforma como um todo oferece um *handler* para erros fatais. Todos os erros fatais devem ser notificados através deste *handler*. Depois de um erro fatal, o sistema só voltará a um estado definido novamente através de ser reiniciado. Estes erros devem ser tratados pelo cliente do *handler*, que determinará como o programa será terminado.

3.2.3 Escalabilidade da UHAPI

A UHAPI possui mecanismos para permitir a adição de novos componentes lógicos. Assim, um fabricante pode definir o seu próprio componente (VUGTS, 2005) como mostra a Figura 22.

Figura 22 – Componente Lógico definido por um fabricante. Modificado a partir de (VUGTS, 2005).

Também possui mecanismos para a adição de novas interfaces a componentes lógicos já definidos pela UHAPI, assim como ilustrado na Figura 23 (VUGTS, 2005).

Figura 23 – Componente Lógico definido pela UHAPI com a adição de uma interface definida por um fabricante. Modificado a partir de (VUGTS, 2005).

A adição de interfaces adicionais não altera o comportamento das interfaces existentes.

3.2.4 Usabilidade da UHAPI

Sendo a UHAPI baseada no COM, um modelo de componentes largamente utilizado pelos desenvolvedores, a tarefa de desenvolver um aplicativo pode ser facilitada. Este fato pode ajudar os desenvolvedores na compreensão de suas APIs e, assim, acelerar o tempo de desenvolvimento.

3.3 Avaliação do MPEG *Multimedia Middleware*

3.3.1 Funcionalidades cobertas do MPEG *Multimedia Middleware*

Conforme descrito anteriormente, o MPEG *Multimedia Middleware* (M3W) também possui APIs de funcionalidade multimídia. Estas APIs são as próprias APIs da UHAPI, já que a UHAPI Forum também participa da elaboração deste padrão.

Além das funções multimídia, existem APIs que são destinadas para tratar requisitos não-funcionais, tais como funções que permitem o acesso a propriedades como segurança, disponibilidade e gerenciamento do sistema. Estas APIs para requisitos não-funcionais utilizam os mesmos conceitos definidos para as APIs de funcionalidade multimídia, ou seja, funcionam como uma extensão da UHAPI como ilustra a Figura 24.

Figura 24 – Recomendações de APIs do MPEG *Multimedia Middleware*

Esta preocupação com requisitos não-funcionais é um dos diferenciais do M3W. Os sistemas de *software* são caracterizados tanto pela sua funcionalidade, ou seja, como o sistema deve-se comportar em termos de serviços ou tarefas que o sistema desempenha, como pela não-funcionalidade ou qualidade, isto é, como o sistema se

comporta em relação a alguns atributos observáveis como desempenho, reusabilidade e disponibilidade. Estes últimos requisitos impõem restrições no sistema sendo desenvolvido (XU, L.; ZIV, H.; RICHARDSON, D., 2005). Ambos os aspectos são relevantes ao desenvolvimento de *software*, mas, tradicionalmente, tem se trabalhado muito mais no primeiro aspecto do que no segundo. (FRANCH, X; BOTELLA, P, 1998).

É assumido que outras funcionalidades do *middleware* serão necessárias mas serão definidas fora do MPEG *Multimedia Middleware* (M3W). Uma lista não exaustiva pode ser encontrada abaixo (GELISSEN, 2004):

- APIs de ambiente de execução;
- APIs de composição;
- APIs de interface de usuário;
- APIs de comunicação e protocolos (por exemplo: SMS (*Short Message Service*), MMS (*Multimedia Messaging Service*), protocolos de fluxo).

3.3.2 Design de projeto do MPEG *Multimedia Middleware*

Em relação ao *design* de projeto do M3W, deve-se considerar o fato de que o M3W adotou o próprio padrão UHAPI para funcionalidades multimídia e estendeu os conceitos da UHAPI para as propriedades não-funcionais.

Desta forma, as mesmas análises feitas para a UHAPI também valem para o M3W.

3.3.3 Escalabilidade do MPEG *Multimedia Middleware*

Pelo motivo descrito no item anterior, as mesmas análises realizadas para a UHAPI também são válidas para o M3W.

3.3.4 Usabilidade do MPEG *Multimedia Middleware*

Também quanto à usabilidade, devido ao mesmo motivo relatado em relação ao *design* de projeto e à escalabilidade, as mesmas considerações sobre a UHAPI também são válidas para o M3W.

3.4 Avaliação da API OpenMAX do Grupo Khronos

Nos subtópicos a seguir, será apresentada a avaliação da API OpenMAX do Grupo Khronos, dando um enfoque maior para a API OpenMAX AL, uma vez que esta API se encontra no mesmo nível da IOS, da UHAPI e do M3W.

3.4.1 Funcionalidades cobertas do OpenMAX AL

O OpenMAX AL foi projetado tendo em mente os desenvolvedores de aplicações multimídia. Oferece suporte para um conjunto de funções para áudio, vídeo e imagens para o desenvolvimento de uma grande variedade de aplicações. Inclui suporte para:

- Reprodução de fluxos multimídia: inclui a reprodução de áudio PCM, áudio codificado, vídeo codificado e imagens bem como a extração de metadados embutidos no conteúdo da mídia. A reprodução de vídeo refere-se ao suporte para reprodução sincronizada para áudio e vídeo. A reprodução de imagens refere-se à decodificação e exibição de imagens de dados comprimidos;
- Gravação de mídia: inclui o suporte a gravação de áudio e vídeo, e captura de imagens. A gravação de vídeo refere-se ao suporte para a gravação sincronizada de áudio e vídeo. A captura de imagens refere-se a funcionalidade de câmera;
- Efeitos e controles: Para áudio, inclui o suporte para controles genéricos tais como volume e balanço, e efeitos de reprodução de música como o equalizador. Para imagem e vídeo, inclui o suporte para controlar o ajuste do brilho, contraste e gama.

A definição de perfis do OpenMAX AL vem de encontro ao fato do mesmo poder ser usado para uma grande variedade de dispositivos atendendo a diferentes segmentos de mercado. Nem todas as implementações do OpenMAX AL irão necessitar de todos este grande conjunto de funções disponíveis. Por exemplo, para um dispositivo que consiga apenas reproduzir mídias, haveria pouca utilidade implementar todas as funções de gravação de mídia que também são partes da API.

O perfil pode ser entendido como um subconjunto definido de funcionalidades do mesmo tipo funcional, que é coletivamente requerido em qualquer implementação que possua este perfil.

O OpenMAX AL define dois perfis (KHROS, 2007b):

- Reprodutor de mídia: neste perfil, é encapsulada a funcionalidade de reprodução de mídia incluindo a habilidade de renderizar áudio, vídeo e imagens em um ou mais formatos. Este perfil é adequado para dispositivos que possuem apenas a capacidade de reproduzir, sem suporte para captura ou gravação de mídia;
- Reprodutor de mídia/Gravador: Este perfil encapsula a funcionalidade de reprodução de mídia e gravação incluindo a capacidade de capturar bem como renderizar áudio, vídeo e imagens em um ou mais formatos.

3.4.2 Design de projeto do OpenMAX

Conforme dito anteriormente, a especificação da API OpenMAX é formada por três sub-camadas: o OpenMAX AL, o OpenMAX IL e o OpenMAX DL. A Figura 25 mostra o relacionamento entre as APIs OpenMAX AL, OpenMAX IL e OpenMAX DL.

O OpenMAX AL representa o nível mais alto da família de APIs do OpenMAX. Como tal, serve as necessidades multimídia dos desenvolvedores de aplicação.

O grupo Khronos também fornece uma API para integradores de sistema, denotada como OpenMAX IL. Esta API define um arcabouço de integração para os elementos internos de uma arquitetura multimídia que abstrai os codecs, manipulações de arquivo, transformações e componentes periféricos instalados no sistema. Também fornece meios para estes componentes interoperarem uns com os outros – mesmo sendo alimentados de múltiplas fontes. Pode ser utilizado pelos componentes que formam um arcabouço multimídia, como por exemplo o GStreamer.

Figura 25 – Relacionamento entre OpenMAX AL, OpenMAX IL e OpenMAX DL. Modificado a partir de (KHRONOS, 2007c).

Os grupos de trabalho para cada API do OpenMAX têm feito uma cooperação para fazer com que o OpenMAX AL seja moldado para uma implementação baseada no OpenMAX IL. Assim o OpenMAX IL pode definir o conjunto de componentes de baixo nível para satisfazer a funcionalidade de alto nível dos casos de uso do OpenMAX AL. Portanto, um implementador do OpenMAX AL pode construir um objeto de mídia como uma cadeia de componentes do OpenMAX IL.

O OpenMAX AL não obriga que uma solução sua seja baseada na OpenMAX IL, nem obriga qualquer implementação em particular. Entretanto, o relacionamento entre APIs pode permitir um rico e eficiente ecossistema de *softwares* multimídia se utilizados de forma integrada (KHRONOS, 2007b).

A implementação dos componentes OpenMAX IL, por sua vez, podem ser moldados com base nas primitivas de processamento multimídia oferecidas pela API OpenMAX DL. Da mesma forma que o OpenMAX AL, o OpenMAX IL não obriga que sua solução seja baseada no OpenMAX DL ou qualquer outra solução particular. Mas como as camadas do OpenMAX foram projetadas para funcionar de forma integrada, a implementação pode ser mais eficiente se utilizados de forma conjunta.

A abordagem adotada pela API OpenMAX AL

A API OpenMAX AL adota uma abordagem orientada a objetos usando a linguagem de programação C. A API inclui dois conceitos fundamentais sobre as quais estão todas as outras construções: objetos e interfaces.

Um objeto é uma abstração de um conjunto de recursos, especificado para um conjunto de tarefas bem definidas, e um estado destes recursos. Cada objeto possui um determinado tipo na sua criação. O tipo de objeto determina o conjunto de tarefas que o objeto pode executar. Pode ser considerado semelhante a uma classe em C++.

Uma interface é uma abstração de um conjunto de funções relacionadas que um certo objeto oferece. Assim, uma interface inclui um conjunto de métodos, que são funções da interface. Cada interface também possui um tipo que determina o exato conjunto de métodos da interface. Pode-se definir a interface como uma combinação do seu tipo e do objeto ao qual está relacionado.

Um ID de interface identifica um tipo de interface. Este identificador é usado dentro do código-fonte para referenciar o tipo de interface.

Objetos e interfaces são conceitos fortemente relacionados – um objeto expõe uma ou mais interfaces, sendo que um objeto pode conter no máximo uma interface de cada tipo. Uma determinada instância de interface é exposta por exatamente um objeto. A aplicação controla o estado de objeto e executa operações com o mesmo exclusivamente através das interfaces expostas.

O tempo de vida de um objeto é o tempo entre a sua criação e a sua destruição. A aplicação explicitamente executa tanto a criação como a sua destruição (KHROS, 2007b).

Um objeto mantém uma máquina de estados com os seguintes estados, como mostra a Figura 26:

- **Estado Não-realizado** (estado inicial): um objeto está criado mas não tem seus recursos alocados. Ainda não é utilizável; os métodos de suas interfaces não podem ser chamados;
- **Estado Realizado**: os recursos dos objetos estão alocados e o objeto é utilizável;
- **Estado Suspenso** (estado opcional): o objeto possui menos recursos do que o necessário para ser utilizável mas mantém o estado que estava no momento da suspensão. Este estado é opcional em caso de perda de recursos, a implementação tem tanto a opção de colocar um objeto tanto no estado suspenso como no estado **Não-realizado**.

Quando uma aplicação destrói um objeto, este último faz uma transição implícita para o estado **Não-realizado**, liberando os recursos disponibilizando-os para outros objetos.

Figura 26 – Diagrama de Estados de um Objeto para o OpenMAX AL. Modificado a partir de (KHRONOS, 2007b).

Representação de um objeto da API OpenMAX AL no código

O OpenMAX AL referencia um objeto através da interface XAObjectItf.

A API representa interfaces como estruturas em C, onde todos os campos são ponteiros para métodos, representando os métodos. Estas estruturas de interfaces sempre são armazenadas e passadas como ponteiro para ponteiro para estrutura e nunca por valor (este nível de indireção permite implementações de API mais eficientes).

Cada um dos métodos da interface possui o primeiro argumento, chamado de self, cujo tipo é o tipo da interface. Desta forma, ao chamar o método da interface, o chamador deve passar o ponteiro para a interface neste argumento.

O trecho de código da Figura 27 ilustra uma interface:

```
struct XASomeInterfaceItf_;
typedef const struct XASomeInterfaceItf_ * const * XASomeInterfaceItf;

struct XASomeInterfaceItf_ {
    XAresult (*Method1) (
        XASomeInterfaceItf self,
        XAuint32 value
    );
    XAresult (*Method2) (
        XASomeInterfaceItf self,
        XAuint32 * pValue
    );
};
```

Figura 27 – Ilustração de uma interface do OpenMAX AL (KHRONOS, 2007b)

A interface é chamada de XASomeInterfaceItf e possui dois métodos: Method1() e Method2(). E serão utilizadas como na Figura 28:

```
XAuint32 i;
XAresult res;

XASomeInterfaceItf someItf;

// ... obtain this interface somehow ...
res = (*someItf)->Method1(someItf, 42); CheckErr(res);
res = (*someItf)->Method2(someItf, &i); CheckErr(res);
```

Figura 28 – Utilização de métodos pelo OpenMAX AL

A Interface XAObjectItf

O XAObjectItf é um tipo de interface fundamental para a arquitetura. Todo tipo de objeto na API OpenMAX AL expõe esta interface. É um ponto de entrada para um objeto já que cada método que cria um novo objeto retorna o XAObjectItf desse objeto. Usando esta interface, uma aplicação pode executar todas as operações básicas deste objeto e pode acessar outras interfaces expostas pelo objeto.

A aplicação destrói o objeto chamando o método `Destroy()` da interface `XAObjectItf`.

A aplicação obtém outras interfaces do objeto através do método `GetInterface()` de `XAObjectItf` e recupera uma interface através de seu ID que o identifica unicamente. Um objeto não pode conter mais do que uma interface de um certo tipo.

A aplicação controla o estado deste objeto chamando os métodos `Realize()` e `Resume()` da interface `XAObjectItf` (KHRONOS, 2007b).

O Objeto Engine e a interface XAEngineItf

Outras entidades fundamentais nesta arquitetura são o objeto **Engine** e a interface **XAEngineItf**. Estas entidades atuam como pontos de entradas da API. A aplicação começa uma sessão OpenMAX AL criando um objeto **Engine**. Ou seja, uma aplicação pode adquirir todos os objetos a partir do objeto **Engine** que encapsula uma sessão OpenMAX AL e serve como um guarda-chuva para todos os outros objetos (KHRONOS, 2007b). A Figura 29 ilustra estes conceitos.

Figura 29 – Aplicação OpenMAX AL, o objeto Engine e abstração de objetos. Modificado a partir de (KHRONOS, 2007b).

O princípio dentre objetos é o **objeto de mídia**. Cada objeto de mídia representa tanto uma tarefa de reprodução ou de gravação, executado por um reprodutor de mídia ou um gravador de mídia respectivamente, que capturam dados de uma determinada origem e os enviam para um determinado destino.

Em alguns casos, estas origens e estes destinos são os próprios objetos. Por exemplo, um dispositivo de câmera pode atuar como uma fonte ou uma saída de áudio. Em outros casos, uma origem ou um destino são um único lugar, tal como um *buffer* de memória contendo um som ou um arquivo para o qual se pode gravar dados.

Todos os objetos expõem interfaces que servem como controles relevantes para sua operação. Interfaces constituem estruturas de métodos agrupados por afinidade funcional. Assim, um reprodutor expõe interfaces para reprodução, controle de taxa, e extração de metadados.

Uma aplicação constrói um caso de uso instanciando o conjunto de objetos requisitado e, então, criando as associações corretas entre eles.

A aplicação controla um caso de uso pela recuperação de interfaces dos objetos que o implementam e chamando os métodos destas interfaces. Dado o exemplo da reprodução acima, a aplicação pode recuperar uma interface de reprodução do reprodutor e então controlar a operação do mesmo usando métodos da interface de reprodução para mudar o estado entre parado, pausado e reproduzindo (KHRONOS, 2007b).

Após a criação do objeto Engine, a aplicação pode obter a interface XAEngineItf deste objeto. Esta interface contém métodos para a criação de todos os tipos de objetos na API.

Processo para criar um objeto (KHRONOS, 2007b):

- Criar e realizar (colocar no estado Realizado) um objeto *Engine*;
- Obter uma interface XAEngineItf deste objeto;
- Chamar o método de criação de objeto apropriado desta interface;
- Se a chamada for feita com sucesso, o método irá retornar uma interface XAObjectItf do novo objeto;
- Depois de trabalhar com o objeto, chamar o método Destroy() de XAObjectItf para liberar o objeto e seus recursos.

Relacionamento entre Objetos e Interfaces

O tipo de objeto determina os tipos de interfaces que podem ser expostas. Cada definição de tipo de objeto inclui uma lista detalhada de interfaces.

O conjunto de tipos de interfaces permitidas para um determinado tipo de objeto pode variar de acordo com o perfil.

O conjunto de interfaces expostas por um objeto é determinado por três fatores:

- O tipo do objeto;
- As interfaces requisitadas pela aplicação durante a criação do objeto;
- As interfaces adicionadas e removidas pela aplicação durante o tempo de vida do objeto.

O tipo de objeto determina o conjunto de interfaces que estão sempre expostas, independentemente da aplicação requisitá-las ou não. Estas interfaces são chamadas de interfaces implícitas e todos os tipos de objetos possuem o seu próprio conjunto de interfaces implícitas. A interface `XAObjectItf` é fundamentalmente requisitada em todos os objetos, assim é designada como implícita em todos os tipos de objetos, isto é, a aplicação nunca necessitará explicitar a requisição para expor um objeto.

Todo tipo de objeto também define um conjunto de interfaces que estão disponíveis, mas não são expostas por um objeto a menos que uma aplicação faça uma requisição explícita durante a criação do objeto. Estas interfaces requisitadas explicitamente são chamadas de interfaces explícitas.

Finalmente, todo tipo de objeto também define um conjunto de interfaces que podem ser adicionadas ou removidas pela aplicação durante o tempo de vida do objeto. Este tipo de interface é chamada de interface dinâmica. Todas as interfaces dinâmicas são gerenciadas por uma interface dedicada, chamada de **`XADynamicInterfaceManagementItf`**.

O conjunto de interfaces explícitas e dinâmicas para um determinado tipo de objeto pode variar entre implementações. Entretanto, para um determinado perfil, cada tipo de objeto tem um conjunto de interfaces explícitas obrigatórias e um conjunto de interfaces dinâmicas obrigatórias que devem ser suportadas por todas as implementações. Interfaces que representam um consumo de recursos significativos

se realizados na criação de objeto mas nunca usados são os primeiros candidatos para interfaces dinâmicas. Tornando-as dinâmicas, o desenvolvedor da aplicação pode usá-la somente quando necessário, resultando na otimização de recursos (KHROS, 2007b).

Alocação de recursos

A quantidade exata de recursos disponíveis numa aplicação OpenMAX AL pode variar entre diferentes implementações e de acordo com o tempo de vida de um objeto *Engine*. Como resultado, uma aplicação que utiliza a API OpenMAX AL deve sempre estar preparada para tratamento em caso de falhas na criação de objetos ou adição de interfaces dinâmicas. Os recursos de um objeto podem ser retirados por uma outra entidade, tais como um outro objeto ou o sistema, sem aviso prévio (KHROS, 2007b).

Modelo de *Threads* do OpenMAX

A API do OpenMAX AL é geralmente síncrona. Isto significa que um método irá retornar somente após sua operação estar completa, e quaisquer mudanças de estado causadas pela chamada do método serão imediatamente refletidas nas chamadas subsequentes.

Entretanto, em alguns casos específicos, uma operação síncrona não é desejável, devido a operações que podem tomar um longo tempo. Em tais casos, o fim de uma operação será sinalizado por uma notificação. Quaisquer mudanças de estado causadas pela chamada da operação estão indefinidas entre o momento da chamada e o momento da notificação.

A API OpenMAX AL pode operar em um desses dois modos, que determinam o comportamento da API com relação a reentrância:

- Modo *Thread-safe*: a aplicação pode chamar as funções da API de vários contextos concorrentemente. A API inteira será *thread-safe* – ou seja, qualquer combinação das funções da API podem ser invocadas concorrentemente (incluindo a invocação do mesmo método mais que uma vez concorrentemente) por múltiplas *threads* da aplicação, e se comportarão conforme o especificado;

- Modo não *Thread-safe*: a aplicação deve cuidar da sincronização e assegurar que num determinado instante, no máximo um método da API está sendo chamado. A API inteira não será *thread-safe*, ou seja, a aplicação necessita se certificar que no máximo uma das funções da API está sendo executada num determinado instante, ou um comportamento inesperado pode acontecer.

O modo de operação é determinado da criação do objeto *Engine*, e não pode ser alterado durante o tempo de vida deste objeto. Uma implementação deve suportar pelo menos um destes modos e deve documentar quais destes modos são suportados (KHRONOS, 2007b).

Notificações de Eventos do OpenMAX

Em muitos casos, a aplicação precisa ser notificada quando algum evento ocorre, como por exemplo, quando a reprodução de um arquivo é encerrada, ou quando um método assíncrono foi completado. Estas notificações são implementadas como funções de *callback* – a aplicação registra um método cuja assinatura é especificada pela API, e este método será chamado pela implementação OpenMAX AL toda vez que um certo evento ocorre.

Funções de *callback* são registradas por interface e por tipo de evento. Desta forma, o registro de um *callback* para um evento num determinado objeto não irá causar o disparo deste *callback* se o mesmo evento ocorrer em um objeto diferente, ou se um diferente evento ocorre em um diferente objeto.

Num determinado instante de tempo, o máximo de uma função de *callback* pode ser registrada por interface, por tipo de evento. Ao registrar um novo *callback* na mesma interface, usando o mesmo método de registro, o *callback* antigo não estará mais registrado. Similarmente, registrar o valor nulo (*NULL*) é uma forma de tirar do registro um *callback* existente sem registrar um novo *callback* (KHRONOS, 2007b).

Tratamento de erros do OpenMAX

Quase todos os métodos de API indicam sucesso ou falha por um código de resultado (exceto para os métodos que não têm a permissão de falhar sob nenhuma circunstância). A documentação dos métodos da API descreve os códigos de resultados válidos para um determinado método e uma implementação deve retornar

um destes códigos de resultados. Para métodos síncronos, o código de resultado é o valor de retorno do método. Para funções assíncronas, o código de resultado está contido nos dados da notificação enviada após o término da operação.

Cada método da API tem um conjunto de pré-condições associadas, consistindo de:

- Faixa de valores para parâmetros;
- Estado da API no qual o método deve ser chamado;
- Contexto do qual o método pode ser chamado.

As pré-condições são claramente documentadas para cada método. Quando uma aplicação viola quaisquer das pré-condições, a chamada do método irá falhar, e o código de resultado do método irá indicar a violação. A API irá permanecer estável e seu estado não será afetado pela chamada (KHRONOS, 2007b).

3.4.3 Escalabilidade do OpenMAX AL

É possível fazer a extensão da API introduzindo novos tipos de interfaces e expor estas interfaces tanto em tipos de objetos já existentes como em tipos de objetos estendidos.

Uma API estendida também pode expor interfaces padrão em tipos de objetos padrão ou estendidos que normalmente não requerem a exposição destas interfaces.

As interfaces estendidas serão definidas de maneira similar às interfaces padrão. Os tipos de interfaces estendidas irão possuir IDs únicos, gerados por aquele que fornecer a extensão.

A API estendida não altera interfaces padrão ou aplica diferentes semânticas em interfaces padrão, mesmo que a sintaxe seja preservada. Uma exceção a esta regra é a extensão para a faixa de valores válidos para parâmetros de função.

É altamente recomendável que todas as vezes em que a assinatura de uma interface mudar, um novo ID de interface seja gerado, e a interface modificada seja considerada uma nova. Isto protege as aplicações já escritas para trabalhar com as interfaces originais (KHRONOS, 2007b).

Uma API estendida pode também introduzir novos tipos de objetos a aqueles especificados na API padrão. Os objetos estendidos podem expor tanto os tipos de

interfaces padrão como os tipos de interfaces estendidas. Caso exponha as interfaces padrão, devem se comportar conforme o especificado. A API estendida pode oferecer tipos de interfaces estendidas com diferentes semânticas (KHRONOS, 2007b).

Também é possível estender a API através da extensão da faixa de valores válidos de parâmetros para um método padrão que tenha uma faixa obrigatória definida pela especificação. A semântica para os valores da faixa estendida é deixada para a especificação da API estendida. Entretanto, para faixas com os valores obrigatórios, a API deve se comportar exatamente de acordo com a especificação (KHRONOS, 2007b).

3.4.4 Usabilidade do OpenMAX AL

Como se pode perceber, o modelo de componentes oferecido pelo OpenMAX AL é semelhante ao da UHAPI (que é um subconjunto do COM), no sentido em que ambos utilizam de maneira muito similar os conceitos de interfaces, objetos e casos de uso. Além disso, a interface XAObjectItf definida pela especificação do OpenMAX AL possui semelhança com a interface IUnknown do COM, sendo uma espécie de interface fundamental.

Uma das principais diferenças do OpenMAX AL em relação a UHAPI é que não existe o conceito da entidade Gerenciador de Conexões no modelo do OpenMAX AL, até porque a quantidade de casos de uso disponíveis ainda é pequena.

O que pode facilitar os desenvolvedores de *software* que utilizam o OpenMAX AL é a sua documentação, que oferece vários exemplos de como utilizar esta API. Uma das desvantagens do OpenMAX AL é que a especificação do mesmo ainda não foi finalizada e, portanto, ainda não existe uma grande comunidade de desenvolvedores trabalhando para aperfeiçoar esta API.

Quanto ao conjunto de funções oferecidas pela API para aplicações voltadas para terminais de acesso, também existe a desvantagem de haver poucos casos de uso previstos e poucos objetos existentes para desenvolver aplicações típicas de TV Digital quando comparados a quantidade de componentes lógicos especificados pela UHAPI.

3.5 Avaliação do arcabouço GStreamer

3.5.1 Funcionalidades cobertas pelo arcabouço GStreamer

O GStreamer é o um arcabouço multimídia, sendo uma tecnologia projetada para fazer o processamento de fluxos de áudio e vídeo. Não é restrita somente para áudio e vídeo, podendo processar qualquer tipo de fluxo de dados (TAYMANS, 2007).

3.5.2 *Design* de projeto do arcabouço GStreamer

O *design* de projeto é baseado na formação de uma *pipeline* a partir de *plugins* de processamento, que formam componentes de *software* interconectados. A *pipeline* é projetada de forma que o *overhead* na passagem de dados entre os *plugins* seja mínimo. Este arcabouço é fundamente projetado sobre três conceitos básicos: os elementos, as arcas (ou *pipelines*) e os pinos.

Um elemento é o bloco de construção mais básico para a *pipeline* do GStreamer. Todos os diferentes componentes são derivados de um elemento. Existem basicamente três tipos de elementos: os elementos de origem, os filtros e os elementos de entrada que serão detalhados a seguir.

Os elementos de origem geram dados para uso por uma *pipeline*, por exemplo ler do disco ou da placa de som. A Figura 30 mostra a ilustração de um elemento de origem. Um pino de saída está desenhado ao lado direito do elemento (RUBY-GNOME2, 2007).

Figura 30 – Ilustração de um elemento de origem. Modificado a partir de (RUBY-GNOME2, 2007).

Elementos de origem não aceitam dados, apenas geram dados. Isso pode ser observado na Figura 30, já que o elemento possui apenas um pino de saída. Um pino de saída pode apenas gerar dados.

Elementos do tipo Filtro têm pinos de entrada e saída. Eles operam nos dados que recebem nos seus pinos de entrada e produzem dados nos seus pinos de saída. Decodificadores de MPEG e filtros de volume são exemplos de elementos dessa categoria.

Os Elementos não possuem restrições quanto ao número de pinos que eles podem ter. Por exemplo, um misturador de vídeo pode ter dois pinos de entrada (as imagens dos dois fluxos de vídeo diferentes) e um pino de saída (RUBY-GNOME2, 2007).

A Figura 31 mostra a visualização de um filtro. Esse elemento tem um pino de entrada e um pino de saída. Pinos de entrada são desenhados a esquerda do elemento (RUBY-GNOME2, 2007).

Figura 31 – Ilustração de um filtro. Modificado a partir de (RUBY-GNOME2, 2007).

A Figura 32 mostra a visualização de um filtro com mais de um pino de saída. Um exemplo de um filtro assim é um demultiplexador. Esse elemento irá interpretar os dados de entrada e extrair os dados de vídeo e áudio. A maioria desses filtros envia um sinal dinamicamente quando um novo pino é criado, assim o programador de aplicação pode ligar um elemento arbitrário ao recém-criado pino (RUBY-GNOME2, 2007).

Figura 32 – Ilustração de um filtro com mais de um pino de saída. Modificado a partir de (RUBY-GNOME2, 2007).

Já os elementos de entrada são pontos terminais em uma *pipeline*. Estes elementos aceitam dados, mas não fazem o processamento dos mesmos para produzir uma saída com os dados transformados. Escrita de disco, som tocado por placas de som, e saídas de vídeo seriam todos implementados por elementos de entrada (RUBY-GNOME2, 2007). A Figura 33 mostra a ilustração de um elemento de entrada.

Figura 33 – Ilustração de um elemento de entrada. Modificado a partir de (RUBY-GNOME2, 2007).

Uma **arca** é um repositório para uma coleção de elementos. A *pipeline* é um subtipo especial de arca que permite a execução de todos os seus elementos filhos. Como as arcas são subclasses de elementos, pode-se controlar uma arca como se fosse um elemento, abstraindo a complexidade de uma aplicação. Pode-se, por exemplo, alterar o estado de todos os elementos da arca mudando o estado da própria arca.

A *pipeline* é uma arca de alto-nível. Alterando o estado para Pausado ou para o estado Reproduzindo, o fluxo de dados será iniciado e processamento de mídia irá acontecer. Uma vez iniciado, as *pipelines* irão executar em uma *thread* separada até que ela seja interrompida ou o fluxo de dados seja encerrado.

Os **pinos** são usados para fazer estabelecer ligações e o fluxo de dados entre os elementos do GStreamer. Um pino pode ser visto como uma porta ou um elemento pelo qual os dados podem fluir de ou para outros elementos. Os pinos possuem capacidades específicas de tratamento de dados: um pino pode restringir o tipo de dado que pode fluir pelo mesmo. As conexões são somente estabelecidas entre dois pinos quando os tipos de dados permitidos são compatíveis entre si (TAYMANS, 2007).

Modelo de *Threads* do GStreamer

O GStreamer é inerentemente projetado para suportar múltiplas *threads* em execução, sendo totalmente *thread-safe*.

Cada fluxo de dados acontece em uma *thread* separada, e o estado de cada elemento é alterado de forma assíncrona (TAYMANS, 2007).

Notificações de Eventos do GStreamer

O GStreamer possui um sistema que é encarregado de encaminhar notificações de mensagens das *threads* de uma *pipeline* para o contexto da *thread* de uma aplicação.

Este sistema periodicamente verifica se existem novas mensagens, e uma função de *callback* é chamada quando alguma mensagem estiver disponível. O GStreamer possui algumas mensagens pré-definidas que estão sendo monitoradas, mas mensagens adicionais podem ser definidas (TAYMANS, 2007).

Tratamento de Erros do GStreamer

As notificações de erros são mensagens pré-definidas sendo monitoradas pelo sistema que monitora as notificações do GStreamer. Desta forma, quando um erro acontece, uma função de *callback* é chamada e o erro deve ser reparado para que a *pipeline* do GStreamer retome a reprodução. Todas estas mensagens contêm informações sobre o tipo e a descrição do erro (TAYMANS, 2007).

3.5.3 Escalabilidade do arcabouço GStreamer

O arcabouço do GStreamer oferece uma API para que novos *plugins* sejam criados. Além disso, todos os *plugins* podem ser estendidos e atualizados de forma independente (TAYMANS, 2007).

Além disso, o GStreamer possui o mecanismo de *autoplugging*, que é o processo pelo qual uma aplicação pode detectar automaticamente o tipo de mídia do fluxo sendo processado e, assim, gerar a *pipeline* mais adequada com base em todos os *plugins* disponíveis no sistema. Devido a natureza dinâmica deste conceito, o GStreamer pode ser automaticamente estendido para suportar novos tipos de mídia sem maiores adaptações na aplicação desenvolvida (TAYMANS, 2007).

3.5.4 Usabilidade do arcabouço GStreamer

Utiliza alguns mecanismos complexos, o que torna a tarefa de criar novas aplicações nem sempre fácil. A flexibilidade e a modularidade oferecida pelo GStreamer é obtida à custa de uma maior complexidade.

O programador utiliza funções de baixo nível para fazer a configuração de uma *pipeline* no GStreamer, o que aumenta o tempo de aprendizado e o tempo de desenvolvimento de uma aplicação.

Além disso, o GStreamer possui vários conceitos avançados como os seus mecanismos internos que incluem o processo *autoplugging*, sincronização, maneiras de interação entre a *pipeline* e a aplicação, parâmetros dinâmicos e manipulação de dados de uma *pipeline*. Estes conceitos forçam o programador a ter um profundo conhecimento sobre este arcabouço de forma a aproveitar a flexibilidade oferecida pelo GStreamer.

Diferentemente da UHAPI, o GStreamer não inclui em seu manual nenhum mecanismo para gerenciar uma grande diversidade de diferentes pipelines, ou seja, um Gerenciador de Conexões.

E, por fim, o GStreamer oferece documentações para desenvolver novas aplicações e novos *plugins*, com vários exemplos de códigos-fonte de como criá-los, o que pode ajudar os desenvolvedores.

3.6 Pontos fortes e Pontos fracos das abordagens em componentes de *software*

Nesta seção será feita uma análise de quais dos trabalhos referidos anteriormente utilizam abordagens de soluções em componentes de *software*, bem como os seus pontos fortes e pontos fracos em relação às soluções que não os utilizam.

A IOS é composta por APIs abertas como o DirectFB para a IOS-GRA e o FusionSound para a IOS-SND, além de APIs que foram desenvolvidas pelo consórcio de universidades brasileiras como a IOS-QOS (qualidade de serviço) e a IOS-IOD (interação com o usuário).

Em princípio, o conjunto de interfaces escolhido pela IOS não é baseado numa abordagem em componentes de *software*, diferentemente de outras interfaces no nível da IOS baseados em componentes de *software* como a UHAPI, o MPEG *Multimedia Middleware* e o OpenMAX do Grupo Khronos.

Assim, esta seção irá avaliar se a abordagem em componentes de *software* é adequada e pode facilitar o desenvolvimento de aplicações e de *software* intermediário.

As principais vantagens de adoção de uma abordagem baseada em componentes são:

- As aplicações num terminal de acesso realizam tarefas comuns em relação aos fluxos multimídia de áudio e vídeo. Tipicamente, a trajetória de dados é direta: os pacotes de dados são recebidos num fluxo contínuo e passam por uma série de transformações distintas (como por exemplo: deciptação, decodificação, renderização de imagens, etc) até o resultado dos dados ser exibido, armazenado ou transmitido. Essas transformações podem ser organizadas para implementar uma variedade de aplicações específicas que são definidas pela sequência na qual essas transformações são executadas. Desta forma, o uso de componentes para cada uma dessas transformações evita que o programador tenha que lher dar manualmente com passos complexos típicos destas aplicações multimídia;

- Os componentes podem formar a unidade base para a manutenção e o atualização de *software*. Desta forma, não haveria mais a necessidade de atualizar sistemas inteiros; ao invés disso, componentes podem ser substituídos ou adicionados conforme a necessidade (D'SOUZA; WILLIS, 1999).
- A divisão em componentes permite o desenvolvimento paralelo. Identificar blocos de granularidade média e focar na especificação prévia de interfaces facilita o desenvolvimento de partes em paralelo (D'SOUZA; WILLIS, 1999).
- O *design* centrado em interfaces pode resultar em arquiteturas escaláveis e extensíveis. Com cada componente possuindo múltiplas interfaces, reduz-se a dependência de um componente possuir funcionalidades irrelevantes de um outro componente que se conecta a ele. Além disso, pode-se introduzir novos componentes e adicionar interfaces relevantes a componentes existentes. Dessa forma, pode-se conseguir sistemas escaláveis de um modo mais fácil (D'SOUZA; WILLIS, 1999).
- Permite a alavancagem de padrões. Já que a tecnologia de componentes implica num conjunto base de padrões para serviços de infra-estrutura, uma vasto conjunto de aplicações pode alavancar estes padrões e poupar um esforço considerável como resultado (D'SOUZA; WILLIS, 1999).

Recentes avanços no desempenho de microprocessadores têm permitido a migração de processamento de mídia contínua do *hardware* dedicado, tais como a descompressão e o processamento digital de sinal, para o *software*. A capacidade de extensão e de configuração de bibliotecas de *software* permite que aplicações multimídia acessem uma gama maior de objetos multimídia, armazenada em uma variedade de formatos comprimidos, e empreguem um conjunto de ferramentas extensível para processamento desses objetos. Além disso, bibliotecas de *software* configuráveis permitem que aplicações tirem vantagem dos novos padrões de compressão de áudio e vídeo assim que emergirem, ao invés de ficarem obsoletos. Apesar dessas vantagens, existem dois problemas fundamentais que tem limitado o

sucesso de bibliotecas de *software* para processamento digital de áudio e vídeo (POSNAK, E. J.; LAVENDER, R.G, 1997):

- Dificuldade de desenvolver componentes de *software*: a tarefa de desenvolver componentes de *software* que decodificam e processam o áudio e o vídeo digital permanece demorada e custosa, devido a falta de capacidade de reutilizar o código. Implementações de componentes de processamento de mídia obtém alto desempenho pela alta adequação do código de processamento de mídia com o código específico do ambiente. Entretanto, esta falta de modularidade torna difícil a reutilização de código e geralmente resulta em uma forma de reuso através da replicação de código, ou seja, através do processo de copiar e colar. A insuficiência de modularidade e o ocultamento de informações requerem que cada novo componente seja projetado do início, e força que o desenvolvedor do componente entenda, em detalhes, a implementação do código reusável bem como a representação digital de suporte dos objetos de áudio e vídeo;
- Variação de recursos durante a exibição. Enquanto a descompressão por *hardware* fornece um conjunto de recursos de processamento dedicado e fixo, um decodificador em *software* é executado numa variedade de diferentes processadores, com velocidades diferindo por uma ordem de magnitude. Além disso, como os recursos de processamento são normalmente compartilhados por várias aplicações, a disponibilidade destes recursos para o decodificador de *software* irá variar com as mudanças no sistema. Tais variações na disponibilidade de recursos de processamento podem resultar em sobressaltos indesejáveis e/ou distorções no sinal de saída, degradando de forma significativa a qualidade da exibição.

3.7 Conclusões

Este capítulo apresentou uma análise com as tecnologias IOS, UHAPI, M3W, OpenMAX AL e GStreamer baseada nos aspectos de funcionalidades cobertas, *design* de projeto, escalabilidade e usabilidade. A Tabela 3 resume os principais pontos levantados neste capítulo.

Tabela 3 – Principais pontos das tecnologias IOS, UHAPI, M3W, OpenMAX e GStreamer

	Funcionalidades cobertas	Design de projeto	Escalabilidade	Usabilidade
IOS	Controle de Energia, Áudio, Gráficos, TV, Segurança, Qualidade de Serviço, Comunicação em rede, Interação com o usuário e <i>hardware</i> básico de computação	Não utiliza abordagem em componentes de <i>software</i> . Cada área funcional possui a sua própria filosofia.	Pode-se adicionar ou remover áreas-funcionais.	Forma um conjunto de interfaces heterogêneo.
UHAPI	Funcionalidades multimídia	Baseado numa abordagem em componentes de <i>software</i> . Seu modelo de componentes é um subconjunto do COM.	Possui mecanismos para criar novos componentes ou novas interfaces a componentes já existentes.	Sendo um subconjunto do COM, um modelo largamente utilizado, o tempo de aprendizado tende a ser menor.
M3W	Funcionalidades multimídia e requisitos não funcionais.	Idem UHAPI	Idem UHAPI	Idem UHAPI.
OpenMAX AL	Funcionalidades multimídia	Baseado numa abordagem em componentes de <i>software</i> .	Possui mecanismos para criar novos componentes ou modificar componentes existentes.	O seu modelo de componentes possui semelhança com o da UHAPI. Também possui boa documentação.
GStreamer	Processamento de fluxos multimídia	Baseado numa abordagem em componentes	Sua API permite a criação de novos componentes.	Sua API inclui mecanismos complexos, o que aumenta o tempo de aprendizagem

Em termos de funcionalidade cobertas, a UHAPI, o OpenMAX e o GStreamer cobrem funcionalidades multimídia, ao passo que a IOS e o M3W são conjuntos mais amplos. O M3W possui funções para acessar propriedades não-funcionais, o que não é coberto pela IOS, mas, em compensação, a IOS possui funções para controle de energia, interação com o usuário e comunicação em rede que não são cobertos pelo M3W.

Dos trabalhos mencionados, o único que não possui um *design* de projeto baseado em componentes é a IOS. A UHAPI, o M3W e o OpenMAX AL especificam APIs padronizadas fundamentadas nos conceitos de objetos e interfaces. A desvantagem destas abordagens baseadas em componentes é que a navegação através de interfaces pode ser uma fonte de erros em tempo de execução caso algum objeto requisitado não seja suportado. Por outro lado, estes três trabalhos possuem um modelo de *threads*, um tratamento de erros e um mecanismo de notificações de eventos padronizado que é válido para todas as suas funcionalidades cobertas. Aqui vale destacar que a UHAPI, assim como o M3W, possuem um modelo de *threads*, mecanismos de tratamentos de erros e de notificações de eventos mais detalhado em relação ao OpenMAX AL, permitindo mais recursos ao programador.

Em relação à escalabilidade, as abordagens baseadas em componentes possuem a vantagem de possuir mecanismos para adicionar componentes ou interfaces, o que não é possível na IOS.

Já em relação à usabilidade, também as abordagens baseadas em componentes possuem uma vantagem em relação a IOS, pois as suas APIs são homogêneas, ou seja, possuem a mesma filosofia de uso para todas as funcionalidades cobertas. Na IOS, cada área funcional possui as suas próprias regras e convenções, obrigando o programador a aprender a lidar cada área funcional de maneira diferenciada.

O único arcabouço multimídia dos trabalhos destacados é o GStreamer, definindo uma infra-estrutura para implementação de componentes. O arcabouço DirectShow não foi avaliado pois não sendo multiplataforma, não pode ser utilizado num sistema operacional Linux e, assim, não pode ser incorporado numa arquitetura de *software* que esteja em conformidade ao padrão SBTVD.

CAPÍTULO 4

PROPOSTA DA ARQUITETURA ARQUIMEDIA

Com base nas análises realizadas no capítulo anterior, apresenta-se, neste capítulo, a proposta da Arquitetura Arquimedia e, em seguida, alguns ensaios parciais para validá-la frente as alternativas existentes de arquiteturas de *software* para terminais de acesso à TV Digital no Brasil.

4.1 Apresentação da proposta da Arquitetura de *Software* Arquimedia

Tendo em vista as análises realizadas no Capítulo 3 e as avaliações dos itens 4.1 e 4.2, propõe-se uma arquitetura de *software* como ilustrado pela Figura 34.

Figura 34 – Arquitetura de *software* proposta

A arquitetura de *software* proposta inclui a utilização do conjunto de APIs definidas pelo MPEG *Multimedia Middleware* (M3W) para acesso a funcionalidades multimídia de áudio, vídeo e gráficos, além de requisitos não funcionais como

segurança, disponibilidade e gerenciamento do sistema. Além disso, propõe-se a utilização da IOS-IOD como interface padronizada para funções de acesso ao *hardware* de interação com o usuário, como o controle remoto e o painel frontal do terminal de acesso.

Como arcabouço multimídia para implementar este conjunto de APIs, propõe-se a utilização do GStreamer. Vale ressaltar que o GStreamer é um arcabouço voltado para processamento de fluxos como são os fluxos multimídia. Portanto, a idéia é que o GStreamer seja usado para a implementação das APIs de funcionalidade multimídia.

A abordagem de componentes de *software* do M3W pode facilitar a criação de aplicações multimídia. Em relação aos outros esforços analisados, o M3W possui as seguintes vantagens:

- O M3W é um esforço internacional de padronização de APIs para *middleware* multimídia de uma organização como o MPEG;
- Diferentemente da IOS, que é um conjunto de interfaces de *software* heterogêneo, o M3W forma uma família de APIs mais homogênea uma vez que a utilização de cada um de seus componentes obedece às regras e convenções de seu modelo de componentes. Na IOS, cada área funcional possui as suas próprias regras e filosofia o que dificulta os desenvolvedores de aplicações;
- Possui APIs extras para tolerância a falhas, gerenciamento de recursos e gerenciamento do sistema, que são um conjunto de funções não cobertas pela IOS e pelo OpenMAX AL. Também cobre funções relacionadas a segurança, sendo uma área-funcional comum a IOS;
- Possui funções específicas para acessar recursos que são típicos e necessários em um sistema de televisão digital como sintonizador, demultiplexador e decodificador, o que facilita o desenvolvimento de aplicações em terminais de acesso. Além disso, sua API é muito mais ampla e cobre um número maior de funcionalidades em relação à API do OpenMAX AL, permitindo que um número muito maior de casos de uso seja implementado;

- Como as APIs multimídia do M3W são baseadas em um subconjunto do COM, que é um padrão largamente utilizado pelos desenvolvedores, existe uma facilidade maior para a compreensão e o aprendizado desta API;
- O M3W possui mecanismos de tratamento de erros, notificações e um modelo de *threads* que tenta balancear problemas de sincronismo e desempenho. Estes mecanismos são comuns para todo o conjunto de APIs definido pelo M3W. Na IOS, cada área-funcional possui os seus próprios mecanismos, sendo que para algumas delas estes mecanismos estão ausentes;
- Possui mecanismos para permitir uma arquitetura escalável, uma vez que componentes totalmente novos podem ser adicionados ou outros possam ser criados a partir de componentes já existentes.

Em relação a utilização da IOS-IOD como APIs para interação com o usuário, isto se deve ao fato do MPEG *Multimedia Middleware* (M3W) reconhecer que é necessário a existência de uma interface padronizada para interação com usuário, mas que a mesma será definida fora desta norma. Dentre as tecnologias apresentadas nesta dissertação, apenas a IOS define uma API para interface com o usuário. A IOS-IOD, que foi desenvolvida pelo consórcio de universidades que participou do projeto do SBTVD.

Já a utilização do GStreamer como arcabouço multimídia está associada ao desafio de oferecer uma infra-estrutura comum para implementação dos componentes de software que possibilite um desempenho elevado para o processamento das aplicações multimídia.

4.2 Ensaio do GStreamer como arcabouço multimídia

Nesta seção, faz-se um ensaio do GStreamer como arcabouço de processamento de mídia. Os principais objetivos nesta etapa serão:

- Avaliar o *overhead* de processamento e de memória que pode ser provocado pelo GStreamer. Muitos sistemas embarcados podem se beneficiar da reutilização de componentes entre produtos e modelos. Por outro lado, também existem barreiras impostas pelo desenvolvimento baseado em

componentes que são particulares aos sistemas de tempo real embarcados. Um dos exemplos mais óbvios é que existe um risco de que os modelos de componentes e tecnologias possam introduzir *overheads* inaceitáveis (LÜDERS, 2003);

- Avaliar o GStreamer como API baseada numa abordagem baseada em componentes de *software*. A criação do GStreamer foi motivada, entre outros aspectos, para permitir que uma grande variedade de aplicações seja desenvolvida através de seu arcabouço baseado em *plugins*. Sendo assim, esta seção tem por objetivo avaliar se esta API oferecida, baseada em *plugins*, pode ser utilizada como uma camada no nível da IOS, do M3W e do OpenMAX AL, dispensando a utilização destas tecnologias.

Uma das principais funções de um terminal de acesso é a decodificação dos fluxos de multimídia e a exibição dos mesmos na tela, ou seja, funções típicas de um terminal de acesso não interativo.

Assim, para avaliar o possível *overhead* introduzido pelo GStreamer na utilização de recursos de processamento e de memória, foram realizados testes utilizando dois programas:

- O programa Ffplay, que é um reprodutor multimídia do projeto Ffmpeg. O projeto Ffmpeg é uma coleção de bibliotecas de *software* capazes de gravar, converter e manipular vários formatos de áudio e vídeo, sendo utilizado como base para a maioria dos programas de *softwares* multimídia livre. Assim, mais de cinquenta outros projetos, muitos deles voltados para sistemas embarcados, utilizam o Ffmpeg (FFMPEG, 2008). O Ffplay não utiliza o arcabouço GStreamer;
- Uma aplicação desenvolvida sobre o arcabouço GStreamer cujo código-fonte pode ser encontrado no **Apêndice A**. Esta aplicação foi desenvolvida especialmente para este teste também com a finalidade de avaliar a API oferecida pelo GStreamer.

Ambos os programas realizam as mesmas funções de decodificar e exibir fluxos de vídeo, utilizando as mesmas bibliotecas de *software* para fazer a decodificação

(bibliotecas do Ffmpeg) e a mesma saída de vídeo (X11). Espera-se que a aplicação GStreamer possa utilizar uma maior quantidade de processamento e de memória. Isto devido aos recursos necessários para que o arcabouço GStreamer ofereça toda a infra-estrutura de suporte aos componentes.

O ambiente de teste utilizado constituiu-se de um PC Pentium IV de 2GHz, com 512MB de Memória RAM e com Sistema Operacional Linux

Para alimentar os programas em questão, foram utilizados fluxos de vídeo armazenados em arquivos em disco, que estão disponíveis em (ELECARD, 2007). O critério para a escolha dos mesmos foi utilizar arquivos de diferentes resoluções (720x576 e 368x208) e codificações (MPEG-2 e H.264/AVC), totalizando ao todo dez arquivos diferentes. Os resultados são mostrados na Tabela 4 e na Tabela 5.

Tabela 4 – Porcentagem (%) de CPU utilizada na decodificação e exibição de arquivos com fluxos de vídeo utilizando o Ffplay e a aplicação GStreamer

Nome do arquivo	Codificação	Resolução	%CPU utilizada	
			Ffplay	Aplicação GStreamer
Amelia_PSP	H.264/AVC	368x208	5,0	4,0
Baroness_best_so_far_PSP	H.264/AVC	368x208	10,7	11,0
Basketball_720p	H.264/AVC	720x576	55,7	58,3
Belukha_PSP	H.264/AVC	368x208	8,7	9,0
KUK_PSP	H.264/AVC	368x208	9,7	9,6
Tank_BMP2	MPEG-2	720x576	21,3	18,0
Tank_T80UK	MPEG-2	720x576	21,3	20,0
Tank_T90	MPEG-2	720x576	21,9	19,6
Tomsk_SD	H.264/AVC	720x576	44,3	44,9
Tomsk_PSP	H.264/AVC	368x208	6,3	6,3
% média de CPU utilizada →			20,49	20,07

Os experimentos mostraram que o consumo de processamento demandado por ambos os programas foi muito semelhante. A aplicação GStreamer consumiu, em média, 0,02% a menos no processamento de um arquivo. Sendo o Ffplay um dos reprodutores de mídia mais utilizados por softwares livre e de código-aberto multimídia, pode-se dizer que em, termos de processamento, o *overhead* introduzido pelo arcabouço GStreamer não foi significativo.

Tabela 5 – Porcentagem (%) de memória utilizada na decodificação e exibição de arquivos com fluxos de áudio e vídeo utilizando o ffplay e a aplicação GStreamer

Nome do arquivo	Codificação	Resolução	% memória utilizada	
			Ffplay	Aplicação GStreamer
Amelia_PSP	H.264/AVC	368x208	1,2	1,5
Baroness_best_so_far_PSP	H.264/AVC	368x208	1,2	1,5
Basketball_720p	H.264/AVC	720x576	2,6	3,0
Belukha_PSP	H.264/AVC	368x208	1,2	1,5
KUK_PSP	H.264/AVC	368x208	1,2	1,6
Tank_BMP2	MPEG-2	720x576	1,5	1,9
Tank_T80UK	MPEG-2	720x576	1,5	1,9
Tank_T90	MPEG-2	720x576	1,5	1,9
Tomsk_SD	H.264/AVC	720x576	2,5	3,0
Tomsk_PSP	H.264/AVC	368x208	1,2	1,9
% média de memória utilizada →			1,56	1,97

Em relação à quantidade de memória consumida, constatou-se que a aplicação GStreamer utilizou mais memória de forma sistemática, sendo que, em média, foi cerca de 21% maior por arquivo em relação ao Ffplay . Este acréscimo no consumo de memória, apesar de ser significativo, não contribuiu para que a aplicação compromettesse o sistema como um todo, uma vez que a porcentagem de memória utilizada ainda é baixa (cerca de 3% para o pior caso). Além disso, sendo o Ffmpeg

um dos projetos mais ativos e utilizados para *softwares* multimídia livre, inclusive em sistema embarcados, é de se esperar que o código-fonte do Ffplay esteja escrito de forma que a aplicação utilize os recursos de memória de forma muito eficiente.

Assim, quanto ao objetivo de avaliar o GStreamer quanto ao seu *overhead* de processamento e memória, o GStreamer se mostrou um arcabouço capaz de atender aos requisitos de desempenho requisitado pelas aplicações.

Já em relação a API do GStreamer, que é baseada numa abordagem de componentes de *software*, pode ser observado pelo **Apêndice A** que a tarefa de configurar uma pipeline é realizada através da utilização de código de baixo nível.

O **Apêndice A** mostra o código da aplicação construída sobre o arcabouço GStreamer utilizado neste teste. Primeiramente, a aplicação inicializa o GStreamer através da chamada à função `gst_init()`. A seguir, diferentes elementos são criados usando a função `gst_element_factory_make()`. Neste caso, foram necessários criar o elemento **filesrc** para ler arquivos do disco, o elemento **decodebin** para detectar o tipo de mídia contido no fluxo, o elemento **ffmpegcolorspace** para decodificar o fluxo de vídeo armazenado no arquivo e o elemento **xvimagesink** para exibir o conteúdo processado na saída de vídeo X11. Então, todos os elementos são adicionados a um elemento que representa uma *pipeline*.

Em relação ao nível de abstração, a API do GStreamer pode ser considerada de baixo nível. Além dos recursos de programação utilizados no código-fonte do **Apêndice A**, o GStreamer possui mecanismos internos como a manipulação de *buffers* de dados e sincronização de dispositivos de saída (TAYMANS, 2007). Assim, a sua API possui uma maior complexidade possibilitando também uma maior flexibilidade.

No entanto, para o contexto de aplicações em TV Digital, o GStreamer não oferece, em sua API, interfaces padronizadas para funções específicas de áudio, vídeo e processamento gráfico. Existem alguns *plugins* que possuem algumas funções padronizadas, mas a maior parte das funções necessárias para um sistema de TV analógico/digital não é coberta. Sendo assim, GStreamer pode ser uma boa opção para ser utilizado para implementar as APIs especificadas por padrões abertos como o M3W e o OpenMAX, que possuem uma gama maior de funções próprias para aplicações de TV.

4.3 Ensaio da utilização de APIs multimídia baseadas em componentes

Nesta seção, será feito um ensaio dos aspectos envolvidos na utilização de APIs baseadas em abordagens em componentes de *software* como a UHAPI e o OpenMAX AL em comparação com APIs que não são baseadas nesta abordagem como a IOS.

Para tanto, serão mostrados alguns trechos de código de duas aplicações:

- Aplicação que utiliza a UHAPI. Os trechos de código desta aplicação foram retirados do projeto UHAPI4Linux (UHAPI4Linux, 2007), um projeto de código-aberto que implementa a UHAPI para computadores pessoais (PCs);
- Aplicação que utiliza a IOS. Os trechos de código desta aplicação foram desenvolvidas pelo autor deste trabalho em conjunto com a equipe de *software* do Núcleo de Engenharia de Mídias do Laboratório de Sistemas Integráveis.

4.3.1 Aplicação construída sobre a UHAPI

O primeiro passo para uma aplicação que utiliza a UHAPI é criar o componente Gerenciador de Conexões, usando a função `uhComCreateInstance()`, como mostra a Figura 35.

```
//declaração de variáveis
struct Instance {
    ...
    uhIConnMgrSelUseCase *connMgr;
    ...
};

static struct Instance inst;

//Criação do Gerenciador de conexões
result = uhComCreateInstance(UHCLSID_uhConnMgr,
                             UHIID_uhConnMgr,
                             reinterpret_cast<void**>(&inst.connMgr));
```

Figura 35 – Trecho de código em que é criado o Gerenciador de Conexões

A função `uhComCreateInstance()` necessita de três parâmetros:

- UHCLSID_uhConnMgr – ID que especifica uma implementação particular de um componente;

- UHIID_uhConnMgr – ID de 128-bits, que identifica de forma única a interface de um componente;
- &inst.connMgr – ponteiro de interface para o Gerenciador de Conexões.

Após criar o Gerenciador de Conexões, o próximo passo é selecionar o caso de uso adequado para a aplicação através do método `SelectUseCase()`, como mostra a Figura 36:

```
uhIConnMgrSelUseCase *connMgr = inst->connMgr;
result = connMgr->SelectUseCase(new_usecase);
```

Figura 36 – Trecho de código exemplificando a seleção de um caso de uso

Ao selecionar um caso de uso, os componentes necessários são criados e configurados, as conexões entre os mesmos são estabelecidas, assim como quaisquer *buffers* são alocados. Isto é feito pelo Gerenciador de Conexões e é transparente para o programador da UHAPI. Assim, os fabricantes que disponibilizam uma plataforma em conformidade com o padrão UHAPI devem oferecer uma documentação com os componentes lógicos e casos de uso suportados pela plataforma.

4.3.2 Aplicação construída sobre a IOS

A aplicação construída sobre algumas interfaces de *software* que compõem a IOS consiste em ler fluxos de áudio e vídeo armazenados em um arquivo em disco, demultiplexá-los, decodificá-los e exibi-los em um *display*.

Para o desenvolvimento desta aplicação, foram utilizadas as seguintes bibliotecas:

- Libavcodec: biblioteca de código-aberto que implementa uma grande variedade de codecs de áudio e vídeo. Escrita em C, a biblioteca faz parte do projeto Ffmpeg. A gama de codecs aliada ao fato do desenvolvimento ser bastante ativo são dois pontos fundamentais para a escolha desta biblioteca;
- DirectFB: biblioteca gráfica que é uma abstração do dispositivo de Frame Buffer do Linux capaz de prover aceleração do hardware gráfico. A IOS-GRA é, em grande parte, composta por funções da API desta biblioteca;

- FusionSound: biblioteca para renderização de áudio com capacidade de suportar mixagem de som, ajuste de frequência, dentre outras vantagens. A IOS-SND é composta por várias das suas funções.

Foram encontradas várias dificuldades no desenvolvimento desta aplicação dentre as quais podemos destacar a falta de uma documentação detalhada, funcionalidades não documentadas e até mesmo documentações não compatíveis com a sintaxe das funções oferecidas pelas bibliotecas citadas. Isso exigiu vários experimentos e testes com cada uma das bibliotecas para descobrir como utilizá-las, aumentando o tempo necessário para o desenvolvimento. Além disso, para se obter o melhor aproveitamento destas bibliotecas, foi constatado que é necessário possuir um alto grau de conhecimento por parte do desenvolvedor.

A seguir, apresentaremos alguns comentários dos trechos de código mais importantes desta aplicação de forma a ilustrar as dificuldades relatadas anteriormente.

Primeiramente, a biblioteca libavcodec foi inicializada através da função `av_register_all()`, registrando todos os formatos de arquivo e codecs disponíveis na biblioteca, de modo que o formato e o codec correspondentes ao arquivo a ser manipulado serão usados automaticamente, conforme a Figura 37.

```
// Registra todos os formatos e codecs
av_register_all();
```

Figura 37 – Trecho de código em que inicializa-se a libavcodec

O segundo passo é abrir o arquivo de áudio/vídeo através da função `av_open_input_file()`, como mostra a Figura 38. Os últimos três parâmetros especificam respectivamente o formato do arquivo, o tamanho do arquivo e os parâmetros do formato.

```
static struct video d_video;
static struct audio d_audio;

av_open_input_file(&d_video.videoFormatCtx, argv[2], NULL, 0,
NULL); // Abre arquivo de vídeo

av_open_input_file(&d_audio.audioFormatCtx, argv[2], NULL, 0,
NULL); // Abre arquivo de áudio
```

Figura 38 – Trecho de código para a abertura do arquivo com os fluxos de áudio e vídeo

O terceiro passo é recuperar informações dos fluxos armazenados no arquivo, conforme ilustrado pela Figura 39.

```
if(av_find_stream_info(d_video.videoFormatCtx)<0)
    return -1; // Não conseguiu encontrar informações do fluxo

if(av_find_stream_info(d_audio.audioFormatCtx)<0)
    return -1; // Não conseguiu encontrar informações do fluxo
```

Figura 39 – Trecho de código para recuperar informações dos fluxos audiovisuais

O quarto passo é obter um ponteiro para o contexto dos codecs para os fluxos de áudio e vídeo, como mostra a Figura 40.

```
d_video.videoCodecCtx=
    d_video.videoFormatCtx->streams[d_video.videoStream]->codec;
d_audio.audioCodecCtx=
    d_audio.audioFormatCtx->streams[d_audio.audioStream]->codec;
```

Figura 40 – Trecho de código para obter um ponteiro para o contexto dos codecs

O quinto passo é encontrar o codec necessário para decodificar os fluxos de áudio e vídeo e abri-los, conforme descrito pela Figura 41.

```
d_video.videoCodec=
    avcodec_find_decoder(d_video.videoCodecCtx->codec_id);

if(d_video.videoCodec==NULL)
    return -1; // Codec não encontrado

d_audio.audioCodec=
    avcodec_find_decoder(d_audio.audioCodecCtx->codec_id);

if(d_audio.audioCodec==NULL)
    return -1; // Codec não encontrado

// Abre codecs
if(avcodec_open(d_video.videoCodecCtx, d_video.videoCodec)<0)
    return -1; // Não foi possível abrir o codec de vídeo

if(avcodec_open(d_audio.audioCodecCtx, d_audio.audioCodec)<0)
    return -1; // Não foi possível abrir o codec de vídeo
```

Figura 41 – Trecho de código para encontrar o codec para decodificação

O sexto passo é inicializar as bibliotecas DirectFB e FusionSound através das funções CreateVideo() e CreateAudio(), conforme a Figura 42.

```

        if (CreateVideo (&v_out, d_video.videoCodecCtx->width,
d_video.videoCodecCtx->height, DSPF_RGB32) != 0)
            printf("Erro em CreateVideo!\n");

        if(atoi(argv[1])==2)
            v_out.sync = 1;

        if(CreateAudio (&a_out))
            printf("Erro em CreateAudio!\n");

        if(ConfigureAudio(&a_out, 9000, 47500, 2))
            printf("Erro em ConfigureAudio!\n");

```

Figura 42 – Trecho de código para inicializar as bibliotecas FusionSound e DirectFB

Em seguida, são disparadas as *threads* de decodificação e renderização de áudio e vídeo, como mostra a Figura 43.

```

pthread_create(&decode_video, &attr, decode_thread_video,
&d_video );
pthread_create(&display_video, NULL, display_thread_video,
&d_video );
pthread_create(&decode_audio, &attr, decode_thread_audio,
&d_audio );
pthread_create(&display_audio, NULL, display_thread_audio,
&d_audio );

```

Figura 43 – Trecho de código para o disparo das *threads* de renderização e decodificação

E, finalmente, os elementos inicializados anteriormente são liberados, conforme ilustrado pela Figura 44.

```

if(atoi(argv[1])==1 || atoi(argv[1])==2){
    // Libera recursos do DirectFB
    DestroyVideo(&v_out);
    DestroyAudio(&a_out);
}

av_free(d_video.videoFrame); // Libera frame de vídeo
av_free(d_audio.audioFrame); // Libera frame de áudio

// Fecha os codecs
avcodec_close(d_video.videoCodecCtx);
avcodec_close(d_audio.audioCodecCtx);

// Fecha arquivos de áudio/vídeo
av_close_input_file(d_video.videoFormatCtx);
av_close_input_file(d_audio.audioFormatCtx);

```

Figura 44 – Trecho de código para liberar os elementos inicializados

Na *thread* de renderização de vídeo, existe uma chamada para a função `DisplayVideo()`, que utiliza a API do DirectFB para apresentar o fluxo de vídeo decodificado na saída de vídeo. Para isso, utilizou-se um recurso do DirectFB que permite a criação de uma superfície a partir de dados pré-alocados. Isto possibilita que uma superfície criada a partir de um decodificador possa ser passada diretamente através de um controlador gráfico, que, por sua vez, exibirá o conteúdo dessa superfície na saída de vídeo, evitando uma cópia de memória custosa em processamento. O trecho de código ilustra este recurso, como mostra a Figura 45.

```
int DisplayVideo(struct vout *v_out, AVFrame *pFrame){
    DFBRectangle rect;
    DFBSurfaceDescription surf_desc;
    int ret;

    surf_desc.flags = DSDESC_WIDTH | DSDESC_HEIGHT |
DSDESC_PIXELFORMAT | DSDESC_PREALLOCATED;
    surf_desc.width = v_out->screen_width;
    surf_desc.height = v_out->screen_height;
    surf_desc.pixelformat = DSPF_I420;
    surf_desc.preallocated[0].data = pFrame->framebuffer;
    surf_desc.preallocated[0].pitch = pFrame->linesize[0];
    ret = v_out->dfb->CreateSurface(v_out->dfb, &surf_desc,
&v_out->frame);
    if (ret != DFB_OK) {
        DirectFBErrorFatal("Erro ao criar superficie",ret);
        return 1;
    }

    rect.x = 0;
    rect.y = 0;
    rect.w = v_out->screen_width;
    rect.h = v_out->screen_height;

    if(v_out->frame == NULL){
        printf("frame NULL\n");
    }

    if(v_out->primary == NULL){
        printf("primary NULL\n");
    }

    if((v_out->primary)->Blit(v_out->primary, v_out->frame, NULL,
0, 0) != DFB_OK){
        printf("Erro em Blit\n");
        return 1;
    }

    if((v_out->primary)->Flip(v_out->primary, NULL, DSFLIP_ONSYNC)
!= DFB_OK){
        printf("Erro em Flip\n");
        return 1;
    }
}
```

```

        v_out->frame->Release(v_out->frame);
        v_out->disp_frames++;

        return 0;
    }

```

Figura 45 – Função usada para renderizar fluxos de vídeo

Já a função `ConfigureAudio()` faz a configuração da saída de áudio como mostra a saída de áudio, conforme mostrado pela Figura 46.

```

int ConfigureAudio(struct aout *a_out, unsigned int buffer_size,
unsigned int sample_rate, unsigned int channels){

    FSSDFStreamDescription      desc;
    int ret;

    /* Preenche descrição do fluxo de áudio(defaults : 44kHz and
16bit). */
    desc.flags      = FSSDF_BUFFERSIZE | FSSDF_CHANNELS |
FSSDF_SAMPLERATE | FSSDF_PREBUFFER;
    desc.buffer_size = buffer_size;
    desc.channels    = channels;
    desc.samplerate  = sample_rate;
    desc.prebuffer   = -1;

    /* Cria o fluxo de áudio e o alimenta. */
    ret = a_out->sound->CreateStream (a_out->sound, &desc, &a_out->stream);
    if (ret) {
        DirectFBEError ("IFusionSound::CreateStream", ret);
        return 1;
    }else{
        printf("CreateStream OK\n");
        a_out->stream->GetDescription(a_out->stream, &desc);
        printf("Descricao do stream criado:\n");
        printf("BufferSize:      %d\n", desc.buffer_size);
        printf("Channels:          %d\n", desc.channels);
        printf("SampleRate:         %d\n", desc.samplerate);
        printf("PreBuffer:          %d\n", desc.prebuffer);
        a_out->channels = desc.channels;
        printf("channels:          %d\n", a_out->channels);
    }

    ret = a_out->stream->GetPlayback(a_out->stream, &a_out->playback);
    if(a_out->playback == NULL){
        printf("fusion sound get playback interface failed\n");
        return 1;
    }else
        printf("Playback OK\n");

    return 0;
}

```

Figura 46 – Função para configurar a saída de áudio

Como pode ser verificado a partir dos trechos de código apresentados anteriormente, para que cada uma das APIs utilizadas nesta aplicação funcione de forma adequada, foi necessário utilizar uma sequência de funções peculiares. Nem sempre a documentação disponibilizada por estas APIs demonstra como utilizá-las, o que torna a tarefa de desenvolver mais difícil, pois obriga o desenvolvedor a fazer vários testes. Na função de renderização de vídeo, o desenvolvedor deve possuir um profundo conhecimento acerca do DirectFB para evitar uma cópia de memória e, assim, tirar vantagem de um recurso disponibilizado por esta API.

4.4 Conclusões

Este capítulo apresentou a proposta de Arquitetura Arquimedia, que inclui a utilização do MPEG *Multimedia Middleware* como API oferecida às camadas superiores (Aplicações e *Middleware*) para as funcionalidades multimídia e requisitos não funcionais, além da utilização da API especificada pela IOS-IOD, que define um conjunto de funções para interação com o usuário. O M3W reconhece que uma interface de interação com o usuário é necessária, mas não será definida por esta norma. Outras áreas funcionais da IOS que não são cobertas pela IOS como a IOS-PWR (Controle de energia) e a IOS-NET (Comunicação em rede) podem, eventualmente, vir a ser novos componentes a serem adicionados pelo M3W.

Além disso, o Arquimedia inclui a utilização do GStreamer como arcabouço multimídia para implementar a família de APIs multimídia formada por componentes definida pelo M3W.

Como pode ser observado, o M3W possui diversas vantagens em relação a IOS e OpenMAX AL, que incluem o fato de ser um esforço internacional de padronização de uma organização como o MPEG, de ser uma API homogênea com funções para acessar requisitos não funcionais, de possuir características do COM diminuindo o tempo de aprendizado e permitir uma arquitetura escalável.

A escolha do arcabouço GStreamer baseado em componentes, por sua vez, está associado do mesmo ser um elemento de infra-estrutura que oferece suporte ao modelo de componentes do M3W. Além disso, os ensaios realizados evidenciam que o GStreamer não dispensa a utilização do M3W e que o seu arcabouço não introduz *overheads* inaceitáveis de processamento e memória.

CAPÍTULO 5

CONSIDERAÇÕES FINAIS

Este capítulo apresenta as principais conclusões deste trabalho, bem como as contribuições obtidas e a indicação de trabalhos futuros para impulsionar o desenvolvimento de outros trabalhos na mesma área.

5.1 Conclusões

Este trabalho apresentou uma proposta de arquitetura de *software* para terminais de acesso e se encontra dentro do âmbito de atividades de pesquisas e desenvolvimento relacionadas com o Sistema Brasileiro de TV Digital.

Essa proposta foi realizada a partir da análise de diversos esforços que estão consolidando técnicas que podem ser aplicadas a sistemas embarcados multimídia. Foram selecionados trabalhos baseados nos principais requisitos da Arquitetura do Terminal de Acesso de Referência tais como utilização de alternativas tecnológicas baseadas em *software* aberto e livre, escalabilidade e requisitos de tempo real. Levando em conta que esforços como o da UHAPI, do M3W e do OpenMAX são recentes, iniciando-se a partir de meados de 2004, além do Gstreamer, este trabalho procurou explorar os principais aspectos de cada um deles. Assim, estas tecnologias foram analisadas em termos de funcionalidades cobertas, *design* de projeto, escalabilidade e usabilidade, e desta forma, foram apresentadas as suas prioridades e as suas especificidades. Foi observado que a adoção de interfaces de *software* padronizadas que utilizam a abordagem de componentes de *software* como o padrão M3W pode facilitar o desenvolvimento de aplicativos e do *Middleware* diminuindo a fragmentação de implementações provocada pela grande diversidade de plataformas e diminuindo o tempo de desenvolvimento.

Entretanto, tais interfaces definem apenas o comportamento de uma determinada função, mas não a forma de implementação. Desta forma, a adoção do GStreamer está relacionada com a definição de uma infra-estrutura comum de componentes para implementar o processamento de fluxos multimídia, sendo uma tecnologia complementar ao M3W.

Verificou-se a viabilidade de utilizar o GStreamer como um arcabouço multimídia e concluiu-se que o mesmo não introduz *overheads* inaceitáveis de processamento e de memória, que são uma das principais barreiras impostas pelo desenvolvimento baseado em componentes em sistemas de tempo real embarcados. Também verificou-se que a API oferecida pelo GStreamer não é adequada o suficiente para dispensar a utilização de uma camada de *software* com um conjunto de APIs padronizadas para acesso aos recursos da plataforma do terminal de acesso como é o caso da IOS, do M3W e do OpenMAX AL.

Assim, chegou-se a proposta da arquitetura Arquimedia que culmina na adoção de interfaces de *software* abertas para o acesso dos recursos da plataforma de um terminal de acesso, no caso do M3W em conjunto com a IOS-IOD, e na adoção do Gstreamer como um arcabouço multimídia que atende aos requisitos de desempenho.

5.2 Contribuições

A principal contribuição deste trabalho é a proposta arquitetura de *software* para terminais de acesso denominada de Arquimedia. Esta arquitetura tenta combinar a adoção de interfaces de *software* abertas que acessam os recursos da plataforma do terminal de acesso e a utilização de um arcabouço multimídia que atenda aos requisitos de desempenho elevado exigidos por serviços de televisão digital.

No Capítulo 2, a contribuição deste trabalho foi um levantamento do histórico e dos trabalhos em sistemas embarcados multimídia com a apresentação dos conceitos e tecnologias relacionadas aos principais requisitos da Arquitetura do Terminal de Acesso de Referência.

Já no Capítulo 3, esta dissertação contribuiu com uma análise de tecnologias selecionadas no Capítulo 2 baseada nos aspectos de funcionalidades cobertas, *design* de projeto, escalabilidade e usabilidade.

A contribuição do Capítulo 4 foi a apresentação da proposta de arquitetura Arquimedia, que foi concebida com base nas pesquisas do Capítulo 2, nas análises do Capítulo 3 e em ensaios realizados no próprio Capítulo 4.

Na literatura a contribuição foi a publicação do artigo (HIRA et al., 2007) sobre o Sistema Operacional do Terminal de Acesso de Referência, o qual apresenta os principais aspectos de projeto e os desafios na concepção do SOTAR e da IOS.

5.3 Trabalhos Futuros

Como continuidade do trabalho, está o desafio de fazer a implementação da Arquitetura Arquimedia.

Por se tratar de uma arquitetura baseada em alternativas tecnológicas de *software* aberto e livre, o Arquimedia pode motivar um grande esforço em conjunto por parte dos desenvolvedores de *software* para terminais de acesso, incentivando a formação de uma grande comunidade de desenvolvedores e criando condições para implementações com uma qualidade cada vez maior. Além disso pode ser uma alternativa muito melhor do ponto de vista econômico pela diminuição de implementações provocada pela grande variedade de plataformas.

Mas para que esta arquitetura de *software* seja interessante para todos os envolvidos, é necessário estimular a criação de um ecossistema de aplicações que utilizem os componentes especificados. A IOS é bom exemplo de que o fato de utilizar um conjunto comum de interfaces de *software* não é uma garantia de facilidade de desenvolvimento, já que, conforme demonstrado no Capítulo 4, a documentação de várias áreas funcionais era escassa, incompleta ou mesmo diferente da sintaxe especificada, o que exigia vários testes por parte do desenvolvedor.

REFERÊNCIAS

ALSA. **Projeto ALSA**. Site oficial do projeto ALSA. Disponível em: <http://www.alsa-project.org>. Acesso em 09 de julho de 2007.

ANDREWS, D. et al. **Impact of Embedded Systems Evolution on RTOS Use and Design**. In: Workshop on Operating System Platforms for Embedded Real-Time Applications, 1., Palma de Mallorca, Espanha, 2005. Anais. Palma de Mallorca: OSPERT'05, 2005. p.13-19.

BACHMANN, F. et al. **Technical Concepts of Component-Based Software Engineering**. Pittsburgh: Software Engineering Institute, Setembro 2002. (Relatório Técnico CMU/SEI-2000-TR-008).

BONDI, A. **Characteristics of Scalability and Their Impact on Performance**. In: Proceedings of the 2nd. Internacional Workshop on Software and Performance, Ottawa, Ontario, Canada, p. 195-203, 2000.

BORE, C; BORE, S. **Profiling software API usability for consumer electronics**. IEEE International Conference on Consumer Electronics, 2005.

BRASIL. **Decreto No 4.901, de 26 de Novembro de 2003**. Institui o projeto do Sistema Brasileiro de Televisão Digital (SBTVD). Diário Oficial da União, Brasília, DF, 26 Nov 2003.

BRASIL. **Decreto No 5.820, de 29 de Junho de 2006**. Dispõe sobre a implantação do SBTVD-T, estabelece diretrizes para a transição do sistema de transmissão analógica para o sistema de transmissão digital do serviço de radiodifusão de sons e imagens e do serviço de retransmissão de televisão, e dá outras providências. Diário Oficial da União, Brasília, DF, 29 Jun 2006.

CARDOSO, N.; ESTEVES, G. **Utilizando componentes em aplicações distribuídas em 3 camadas**. Dissertação - Universidade Federal de Goiás. Goiânia, 2002.

CASTRO, A. **O Tratamento Jurídico do Software no Brasil**. Disponível em: <http://www.cbeji.com.br/br/novidades/artigos/main.asp?id=3905>. Acesso em 20 de Agosto de 2007.

CESAR, P. **A Graphics Software Architecture for High End Interactive TV Terminals**. Dksmdlasm. 2005. Dissertação (Doutorado) - Universidade de Tecnologia de Helsinki. Helsinki, 2005.

Consumer Electronics Linux Forum. **CELF Audio Video Graphics Specification v2.0**. Estados Unidos, 2006.

Consumer Electronics Linux Forum. **CELF Specification V1.0 R2**. Estados Unidos, 2004.

Consumer Electronics Linux Forum. **Informações sobre a entidade e especificações**. Disponível em: <http://www.celinuxforum.org>. Acesso em: 21 de Fevereiro de 2007.

CRNKOVIC, I.; LARSSON, M. **Building reliable component-based software systems**. Norwood, MA: Artech House, 2002.

DIRECTSHOW. **Arcabouço DirectShow**. Página com informações sobre o arcabouço. Disponível em <http://en.wikipedia.org/wiki/DirectShow>. Acesso em 20 de fevereiro de 2008.

DIRKS, B.; VERKUIL, H.; RUBLI, M. **Video for Linux Two API Specification**. Draft 0.21, 2006.

D'SOUZA, D.; WILLIS, A. **Objects, Components and Frameworks with UML – The Catalyst Approach**. Addison-Wesley, 1999.

ELECARD Ltd. **Área de download de clips**. Disponível em: <http://www.elecard.com/download/clips.php>. Acesso em 30 de Novembro de 2007.

FFMPEG. **Projeto Ffmpeg**. Disponível em <http://sourceforge.net/projects/ffmpeg>. Acesso em 08 de Janeiro de 2008

FRANCH, X; BOTELLA, P. **Putting Non-Functional Requirements into Software Architecture**. Dept. Llenguatges i Sistemes Informàtics (LSI), Universitat Politècnica de Catalunya (UPC), Barcelona, 1998.

FUSIONSOUND. **Projeto FusionSound**. Manual de Referência do FusionSound. Disponível em: http://www.directfb.org/docs/FusionSound_Reference/index.html. Acesso em 25 de Maio de 2007

GELISSEN, J. **Middleware for MPEG applications**. In: Euromicro Conference on Real-Time Systems, 16, Catania, Itália, 2004.

GINGA. **Middleware GINGA**. Site oficial do *middleware* Ginga. Disponível em <http://www.ginga.org.br> . Acesso em 09 de Julho de 2007

GLIBC. **GNU C LIBRARY**. Disponível em: <http://www.gnu.org/software/libc/>. Acesso em 10 de Julho de 2007.

GSTREAMER. **Página com informações sobre o arcabouço**. Disponível em <http://gstreamer.freedesktop.org> . Acesso em: 22 de Fevereiro de 2007.

HIRA, C.; COSTA, L; NUNES, R; REAL, L; ZUFFO, M. **Sistema Operacional do Terminal Acesso de Referência**. In: Workshop de Sistemas Operacionais, Rio de Janeiro, RJ, 2007. SBC 2007 WSO Anais. Rio de Janeiro. p.786-795.

HUNDT, A. **DirectFB Overview (v0.2 for DirectFB 0.9.21)**. 2004.

ISO/IEC. **MPEG Multimedia Middleware: Requirements on the MPEG Multimedia Middleware v2.0**, Mallorca, Espanha, Outubro, 2004

ISO/IEC FCD 23004-1. **Information Technology – Multimedia Middleware – Part 1: Architecture**, 2006.

KHRONOS. **Grupo Khronos**. Site oficial do grupo Khronos. Disponível em <http://www.khronos.org> . Acesso em 09 de Julho de 2007a

KHRONOS. **OpenMAX Application Layer Application Programming Interface Specification**. Version 1.0 Provisional RC2 September 2, 2007b

KHRONOS. **OpenMAX Integration Layer Application Programming Interface Specification**. Version 1.1.1 Setembro, 2007c

LATURA, B. **Linux and Embedded OS's**. International Association of Open Systems Professionals, Chicago, 2002

LEITE, L. E. C.; BATISTA C. E. C. F.; FILHO, G. L. de S.; KULESZA R.; ALVES, L. G. P.; BRESSAN, G.; RODRIGUES, R. F.; SOARES, L. F. G.. **FlexTV — Uma Proposta de Arquitetura de Middleware para o Sistema Brasileiro de TV Digital**, Revista de Engenharia de Computação e Sistemas Digitais PCS/EPUSP no 2 pág 29-49, Novembro de 2005.

LEMOS, G.; FERNANDES, J.; SILVEIRA, G.. **Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas**. In: Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação (JAI-SBC2004), 2004.

LINUXTV. **Projeto LinuxTV**. Site oficial do projeto LinuxTV. Disponível em <http://www.linuxtv.org>. Acesso em 22 de Fevereiro de 2008

LSI-TEC. **Implementação de Referência do Terminal de Acesso**. São Paulo, 2005.

LÜDERS, F. **Use of Component-Based Software Architectures In Industrial Control Systems**. 2003. 134p. Tese (Doutorado). Department of Computer Science and Engineering, Mälardalen University. Suécia, 2003.

MICROSOFT. **The Component Object Model**. Disponível em: <http://msdn2.microsoft.com/en-us/library/ms694363.aspx>. Acesso em: 06 de Abril de 2007.

MORENO, M.; SOARES NETO, C.; GOMES A.T.; COLCHER, S; SOARES, L.F. **QoSOS: An adaptable architecture for QoS provisioning in network operating systems**. In: Journal of the Brazilian Telecommunications Society, Special Issue, v.18, n.2, p.118-131, 2003.

MORRIS, S.; SMITH-CHAIGNEAU, A. **Interactive TV Standards**. Estados Unidos: Elsevier, 2005. 585p.

NSS. **Network Securities Services**. Disponível em <http://www.mozilla.org/projects/security/pki/nss/>. Acesso em 25 de Maio de 2007.

PEREIRA, F. **Technologies for Digital Multimedia Communications: An Evolutional Analysis of MPEG Standards**. China Communications Journal, vol.3, n. 5, p.9-19, Outubro, 2006

PLAGEMANN, T. **Middleware + Multimedia = multimedia middleware?**. Multimedia Systems, v.8, n.5, Dezembro 2002.

POSIX. **ISO/IEC 9945-1:2003 Information technology - Portable Operating System Interface (POSIX) - Part 1: Base Definitions**, 2003.

POSNAK, E. J.; LAVENDER, R.G. **An Adaptive Framework for Developing Multimedia Software Components**. Communications of the ACM. Volume 40, n. 10, p.43-47, 1997.

RUBY-GNOME2. **Tutorial Ruby/GStreamer**. Disponível em: http://ruby-gnome2.sourceforge.jp/pt_BR/hiki.cgi?tut-gst-intro. Junho, 2007.

SAMETINGER, J. **Software Engineering with Reusable Components**. New York: Springer, 1997.

SEI. **SEI Views and Beyond Architecture Documentation Template**, 2006

SEI. **Component Object Model (COM), DCOM, and Related Capabilities**. Disponível em: <http://www.sei.cmu.edu/str/descriptions/com.html>. Acesso em: 07 de Abril de 2007

SOUZA, C. T.; OLIVEIRA, C. T.. **Especificação de Canal de Retorno em Aplicações para TV Digital Interativa**. In: Simpósio Brasileiro de Telecomunicações, 22., Campinas, 2005. Campinas: SbrT, 2005. p. 1141-1146.

SOUZA, F. B.; ELIAS, G. **Uma Arquitetura de Middleware para Sistemas de Televisão Interativa**. In: Workshop de Desenvolvimento Baseado em Componentes, 2003.

SPAGNOLI, L.; BECKER, K. **Um estudo sobre o desenvolvimento baseado em componentes**. PUCRS, 2003.

SZYPERSKI, C. **Component Software: beyond object-oriented programming**. Harlow: Addison-Wesley, 1999.

TAYMANS, W. et al. **GStreamer Application Development Manual (0.10.9.1)**. Disponível em <http://www.sti.uniurb.it/acquaviva/didattica/manual.pdf>. Acesso em: 08/12/2007.

UHAPI Forum. **The Universal Home API – Basic concepts and example applications – A WHITE PAPER**. Outubro, 2004.

UHAPI4Linux. **Implementação de código-aberto da UHAPI**. Disponível em: <http://sourceforge.net/projects/uhapi4linux>. Acesso em 30 de Novembro de 2007.

Universal Home API. **Informações sobre a entidade e especificações.** Disponível em: <http://www.uhapi.org>. Acesso em 21 de Fevereiro de 2007.

VUGTS, J. **Overview of UHAPI Architecture and Specifications.** Internationale FunkAusstellung, Berlin, 2005.

WHATIS Computer Dictionary. **Definição de Escalabilidade.** Disponível em: <http://www.whatis.com/scalabil.htm>. Acesso em 30 de Novembro de 2007.

WU, Y; HIRAKAWA, S.; REIMERS, U. H.; WHITAKER, J. **Overview of digital television development worldwide.** Proceedings of the IEEE, Canadá, v.94, n.1, p. 8-21, janeiro 2006.

WVDIAL. **Software WvDial.** Disponível em <http://open.nit.ca/wiki/?WvDial>. Acesso em 10 de Julho de 2007.

XU, L.; ZIV, H.; RICHARDSON, D. **Towards Modeling Non-Functional Requirements in Software Architecture.** In: Proceedings of Early Aspects 2005: Aspect-oriented requirements engineering and architecture design workshop, Chicago, EUA, 2005.

APÊNDICES

APÊNCICE A – Código da aplicação GStreamer

```
#include <gst/gst.h>

GstElement *pipeline, *video;

static gboolean
bus_call (GstBus *bus,
          GstMessage *msg,
          gpointer data)
{
    GMainLoop *loop = (GMainLoop *)data;
    switch (GST_MESSAGE_TYPE(msg)) {
        case GST_MESSAGE_EOS:
            g_print("Fim do arquivo\n");
            g_main_loop_quit(loop);
            break;
        case GST_MESSAGE_ERROR: {
            gchar *debug;
            GError *err;
            gst_message_parse_error(msg, &err, &debug);
            g_free(debug);
            g_print("Erro:%s\n", err->message);
            g_error_free(err);
            g_main_loop_quit(loop);
            break;
        }
        default:
            break;
    }
    return TRUE;
}

static void
cb_newpad (GstElement *decodebin,
          GstPad *pad,
          gboolean last,
          gpointer data)
{
    GstCaps *caps;
    GstStructure *str;
    GstPad *videopad;

    videopad = gst_element_get_pad (video, "sink");
    if (GST_PAD_IS_LINKED (videopad)) {
        g_object_unref (videopad);
        return;
    }

    /* checagem do tipo de mídia */
}
```

```

caps = gst_pad_get_caps (pad);
str = gst_caps_get_structure (caps, 0);
if (!g_strrstr (gst_structure_get_name (str), "video")) {
    gst_caps_unref (caps);
    gst_object_unref (videopad);
    return;
}
gst_caps_unref (caps);

gst_pad_link (pad, videopad);
}

gint
main (gint  argc,
      gchar *argv[])
{
    GMainLoop *loop;
    GstElement *src, *dec, *conv, *sink;
    GstPad *videopad;
    GstBus *bus;

    /* inicializa GStreamer */
    gst_init (&argc, &argv);
    loop = g_main_loop_new (NULL, FALSE);

    /* assegurar que temos um arquivo como entrada */
    if (argc != 2) {
        g_print ("Usage: %s <filename>\n", argv[0]);
        return -1;
    }

    /* configuração */
    pipeline = gst_pipeline_new ("pipeline");

    /* cria elemento filesrc */
    src = gst_element_factory_make ("filesrc", "source");
    g_object_set (G_OBJECT (src), "location", argv[1], NULL);

    /* cria elemento decodebin */
    dec = gst_element_factory_make ("decodebin", "decoder");
    g_signal_connect (dec, "new-decoded-pad", G_CALLBACK
(cb_newpad), NULL);
    gst_bin_add_many (GST_BIN (pipeline), src, dec, NULL);
    gst_element_link (src, dec);

    /* criação da saída de vídeo */
    video = gst_bin_new ("videobin");
    conv = gst_element_factory_make ("ffmpegcolorspace",
"conv");
    videopad = gst_element_get_pad (conv, "sink");
    sink = gst_element_factory_make ("xvimagesink", "sink");
    gst_bin_add_many (GST_BIN (video), conv, sink, NULL);
    gst_element_link (conv, sink);

```

```
gst_element_add_pad (video,  
    gst_ghost_pad_new ("sink", videopad));  
gst_object_unref (videopad);  
gst_bin_add (GST_BIN (pipeline), video);  
  
bus = gst_pipeline_get_bus(GST_PIPELINE(pipeline));  
gst_bus_add_watch(bus, bus_call, loop);  
gst_object_unref(bus);  
  
/* executa a pipeline */  
gst_element_set_state (pipeline, GST_STATE_PLAYING);  
g_main_loop_run (loop);  
  
/* desalocando memória */  
gst_element_set_state (pipeline, GST_STATE_NULL);  
gst_object_unref (GST_OBJECT (pipeline));  
  
return 0;  
}
```