PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Suzana Mesquita de Borba Maranhão

Ajuste Elástico em Tempo de Exibição para Fluxos de Áudio Comprimido

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA

Programa de Pós-Graduação em Informática

Rio de Janeiro
Abril de 2006

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Suzana Mesquita de Borba Maranhão

Ajuste Elástico em Tempo de Exibição para Fluxos de Áudio Comprimido

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Luiz Fernando Gomes Soares

Rio de Janeiro, abril de 2006.

Suzana Mesquita de Borba Maranhão

Ajuste Elástico em Tempo de Exibição para Fluxos de Áudio Comprimido

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Luiz Fernando Gomes Soares
Orientador
Departamento de Informática - PUC-Rio

Geber Lisboa Ramalho
Centro de Informática - UFPE

Sérgio Colcher
Departamento de Informática - PUC-Rio

Rogério Ferreira Rodrigues
Departamento de Informática - PUC-Rio

José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico - PUC-Rio

Rio de Janeiro, 03 de abril de 2006.

Suzana Mesquita de Borba Maranhão

Graduou-se em Ciência da Computação pela UFPE (Universidade Federal de Pernambuco) em 2003. Foi bolsista da CAPES em áreas de Probabilidade e Engenharia de Software durante a graduação.

Ficha Catalográfica

Maranhão, Suzana Mesquita de Borba

Ajuste Elástico em Tempo de Exibição para Fluxos de Áudio Comprimido / Suzana Mesquita de Borba Maranhão; orientador: Luiz Fernando Gomes Soares – Rio de Janeiro: PUC, Departamento de Informática, 2006.

136 f. ; 29,7 cm

Incluí referências bibliográficas.

1. Informática – Teses. 2. Ajuste elástico. 3. Áudio Comprimido (MP3, MP2, AAC, AC-3). 4. MPEG-2 Sistemas. 5. Sincronização Intermídia. 6. Apresentação Hipermídia. 7. Formatador Hipermídia. 8. Formatador Hyperprop. I. Soares, Luiz Fernando Gomes. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Este trabalho é dedicado

Aos meus amados pais, que mesmo longe, sempre estiveram por perto. Grandes educadores, eles me ensinaram muito mais do que pode ser encontrado em livros ou expresso em palavras.

A Marcio, essa pessoa tão especial por quem me apaixonei no decorrer deste mestrado. Um poço de loucuras, carinhos e histórias para contar, esse mineirinho me faz muito feliz.

Agradecimentos

Agradeço a Luiz Fernando, exemplo de orientador, cientista e ser humano. Pessoa que trabalha de modo brilhante, contagiante e, ao mesmo tempo, com extrema simplicidade. Sua personalidade forte e transparente é melhor descrita nas palavras de Chang Yü, extraídas do livro A Arte da Guerra: “Bons generais são ao mesmo tempo amados e temidos”.

Em especial agradeço também ao professor Geber Ramalho do Centro de Informática da UFPE. Geber é admirado há muitos anos não só por mim, mas por todos que lotam sua sala de aula para qualquer disciplina que ele se disponha a ser professor. Foi Geber que me orientou para fazer o mestrado na PUC e agora me deu a honra de participar da banca de avaliação do meu trabalho. Também a Sérgio Colcher, o conhecido por tudo saber, que também se dispôs a contribuir na avaliação deste trabalho.

Como não poderia esquecer, agradeço à professora Marcília Campos, também da UFPE, por ter me introduzido na vida acadêmica e ter sempre feito tudo que foi possível, e às vezes o impossível, para incentivar essa minha vocação.

Agradeço também às pessoas do TeleMídia, que estão sempre alegres, solícitas e prontas para o próximo desafio. A energia dessas pessoas torna o laboratório um lugar realmente especial de se trabalhar. Em especial, obrigada a Rogério, que foi meu guia em vários pontos do trabalho, fundamental em vários momentos. Também não posso deixar de ressaltar os professores e funcionários da PUC-Rio, que fazem da instituição um centro de excelência de ensino no Brasil e no mundo.

Por fim, agradeço ainda ao CNPq, a PUC-Rio e ao Laboratório TeleMídia pelo apoio, inclusive financeiro, indispensável à realização deste trabalho.

Resumo

Maranhão, Suzana Mesquita de Borba. **Ajuste Elástico em Tempo de Exibição para Fluxos de Áudio Comprimido.** Rio de Janeiro, 2006. 136p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Ajuste elástico é a técnica de modificar a duração de exibição de um objeto de mídia. Este trabalho propõe algoritmos de ajuste elástico que funcionam em tempo de exibição para fluxos de áudio comprimidos, com aplicabilidade principal em cenários onde não existe interferência do usuário como, por exemplo, a manutenção da consistência temporal de um documento hipermídia. Os algoritmos de ajuste de áudio são propostos, desenvolvidos e integrados a algoritmos de ajuste de vídeo. Além disso, um algoritmo de resincronização é proposto, e implementado, para preservar o sincronismo intermídia durante a realização de ajuste em fluxos de sistemas MPEG-2. A união dos algoritmos implementados com o desenvolvimento da integração a exibidores de conteúdo deu origem à ferramenta de ajuste, capaz de ser facilmente utilizada por aplicações que precisam solicitar ajuste elástico em fluxos de mídia comprimidos. Como exemplo, a ferramenta de ajuste foi integrada a ferramentas de exibição de um formatador hipermídia. Os algoritmos propostos podem ajustar a duração de áudio em até 10% mantendo a qualidade da percepção do áudio dentro de limites aceitáveis, conforme demonstrado através de medidas de qualidade que também são apresentadas neste trabalho.

Palavras-chave

Ajuste elástico; Áudio Comprimido (MP3, MP2, AAC, AC-3); MPEG-2 Sistemas; Sincronização Intermídia; Apresentação Hipermídia; Formatador Hipermídia; Formatador Hyperprop.

Abstract

Maranhão, Suzana Mesquita de Borba. **On-the-fly Timescale for Compressed Audio Streams.** Rio de Janeiro, 2006. 136p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Timescale is a technique used to modify media-object playing duration. This work proposes timescale algorithms, to be performed on-the-fly, for compressed audio streams. The focus is situations where there is no user interference as, for example, the temporal consistency maintenance of a hypermedia document. The algorithms are proposed, developed and integrated to video timescale algorithms. Furthermore, a resynchronization algorithm is also proposed, and implemented, in order to maintain inter-media synchronization in MPEG-2 system streams. The timescale tool is composed by the aforementioned algorithms and their integration to content rendering tools. The proposed solution can be easily used in applications that need timescale compressed media streams. As an example, the adjustment tool was integrated to a hypermedia formatter. The proposed algorithms can adjust audio durations up to 10%, maintaining the audio perceptual quality within acceptable limits, as observed through quality measurements also described in this work.

Keywords

Timescale; Compressed Audio (MP3, MP2, AAC, AC-3); MPEG-2 Systems; Inter-media Synchronization; Hypermedia Presentation; Hypermedia Formatter; HyperProp Formatter.

Índice

1 Introdução 16
1.1. Motivação 16
1.2. Objetivos 19
1.3. Organização do restante do documento 19

2 Trabalhos Relacionados 21
2.1. Categorização de algoritmos de ajuste para áudio 21
2.2. Ferramentas que realizam ajuste elástico 25
2.3. Comparação dos trabalhos relacionados com o proposto 32

3 Ajuste Elástico de Áudio no Fluxo Comprimido 35
3.1. Análise Resumida do Ajuste Elástico Proposto: Requisitos do Algoritmo 35
3.2. Modelagem do Fluxo de Áudio Com Compressão 37
3.3. Mecanismo de Ajuste Elástico de Áudio com Compressão 39
3.4. Ajuste Elástico em Fluxos de Áudio Suportados 45
3.5. Ajuste Elástico em Fluxos de Sistemas 54

4 Ferramenta de Ajuste Elástico 66
4.1. APIs de ajuste elástico 67
4.2. Visão Geral da Implementação da Ferramenta de Ajuste Elástico 70
4.3. Implementação do Algoritmo de Ajuste Elástico de Áudio 75
4.4. Implementação do Algoritmo de Elástico em Fluxos de Sistemas 81
4.5. Mudanças no Algoritmo de Ajuste Elástico de Vídeo para Integração 88

5 Integração da Ferramenta de Ajuste com Exibidores de Conteúdo 90
5.1. Mecanismo de Comunicação utilizado com Ajuste Elástico em Tempo de Compilação 91

5.2. Mecanismo de Comunicação utilizado com Ajuste Elástico em Tempo de Execução 92
5.3. Considerações Adicionais 94

6 Utilização da Ferramenta de Ajuste em Sistemas Hipermídia 95
6.1. Necessidades de Adaptabilidade de Sistemas Hipermídia 95
6.2. Utilização da Ferramenta de Ajuste em Sistema Hipermídia 98
6.3. Integração da Ferramenta de Ajuste com o Sistema HyperProp 100

7 Resultados Obtidos 104
7.1. Medidas de Qualidade Subjetiva 105
7.2. Medidas de Qualidade Objetivas 110
7.3. Análise da Implementação 113

8 Considerações Finais 117

9 Referências 120

10 Apêndice A: Exemplos de Código para Solicitar Ajuste Elástico 126

11 Apêndice B: Aplicação de envio e recepção de fluxo de mídia 129

12 Apêndice C: Exemplo da Integração com Formatador HyperProp 130

13 Apêndice D: Aplicação de Teste da Ferramenta de Ajuste 135

Lista de figuras

Figura 1 - Mudança da taxa de exibição de um sinal. 22
Figura 2 - Exemplo de cortes de quadros de um sinal. 22
Figura 3 - Algoritmo *Ajuste Regular*. 23
Figura 4 - Ajuste elástico com reprodução dos quadros intercalados. 24
Figura 5 - Ajuste elástico de quadros do sinal no domínio da freqüência. 25
Figura 6 - Interface de aplicação de ajuste elástico em arquivos de áudio no *Sound Forge*. 26
Figura 7 - Interface de aplicação de ajuste elástico em arquivos de áudio no *Window Media Player 10*. 27
Figura 8 - Interface do *software Amazing Slow Downer*. 28
Figura 9 - Interface do *Enounce* 2xAV. 28
Figura 10 - Interface de controle do *585 Time Scaling Processor*. 29
Figura 11 - Estrutura de funcionamento do *FastMPEG*. 30
Figura 12 - Decodificador MPEG-4 - fonte: (Wolters & Kjörling, 2003). 31
Figura 13 - Realização do ajuste em fluxo comprimido, utilizando decodificação prévia e nova codificação. 35
Figura 14 - Realização do ajuste em fluxo comprimido, utilizando decodificação prévia. 36
Figura 15 - Realização do ajuste diretamente no fluxo comprimido. 36
Figura 16 - Representação dos quadros que compõem um fluxo de áudio com unidade lógica de dados igual ao campo de dados. 38
Figura 17 - Representação dos quadros que compõem um fluxo de áudio. 38
Figura 18 - Representação dos quadros que compõem um fluxo de áudio com bits de PAD no quadro 1 e 3. 39
Figura 19 - Representação dos quadros no primeiro passo para remover o quadro 3. 40
Figura 20 - Representação dos quadros após remoção do quadro 3. 40

Figura 21 - Representação dos quadros no primeiro passo para duplicar o quadro 2. 41
Figura 22 - Representação dos quadros após duplicação do quadro 2. 41
Figura 23 - Representação dos quadros após transferência do PAD para quadro 4. 42
Figura 24 – Reposicionamento de âncoras em um fluxo de mídia devido ao ajuste. 44
Figura 25 - Cálculo do tamanho de um quadro MP2 ou MP3. 46
Figura 26 - Cálculo do tamanho de um quadro MP1. 46
Figura 27 - Codificador e decodificador MPEG-1 e MPEG-2 áudio MC - fonte: (Noll, 1999). 49
Figura 28 - Tipos de codificação para diferentes taxas de bits em MPEG-4 natural - fonte: (ISO, 2001b). 52
Figura 29 - Quadro do formato AC-3. 53
Figura 30 - Fluxo de sistemas com fluxos elementares de áudio e vídeo uniformemente distribuídos. 55
Figura 31 - Fluxo de sistemas com fluxos elementares de áudio e vídeo com distribuição não-uniforme. 55
Figura 32 – Conjunto de algoritmos para realizar ajuste elástico em fluxos de sistemas. 56
Figura 33 - Transformação de um fluxo de sistemas em um fluxo elementar para aplicar algoritmo de ajuste elástico. 57
Figura 34 - Representação de divisão dos fluxos elementares em trechos para recálculo de fator de ajuste durante aplicação de ajuste elástico. 59
Figura 35 - Cálculo do fator de ajuste a ser aplicado no trecho i+1 dos fluxos elementares componentes da mídia original. 59
Figura 36 - Estrutura de fluxo MPEG-2 - fonte: (Cavendish, 2005). 61
Figura 37 - Representação de parte de um fluxo de sistemas, com PACKs, PACKETs e bits de fluxos elementares. 62
Figura 38 - Cálculo do SCR em fluxos MPEG-2. 63
Figura 39 - Cálculo do SCR base em fluxos MPEG-2. 63
Figura 40 - Cálculo do SCR extension em fluxos MPEG-2. 63
Figura 41 - Intervalo de variação do *system_clock_frequency*. 63

Figura 42 - Cálculo do PTS em fluxos MPEG-2. 64
Figura 43 - Cálculo do DTS em fluxos MPEG-2. 64
Figura 44 - Contexto de utilização da ferramenta de ajuste elástico. 66
Figura 45 - Implementação das APIs de ajuste elástico. 69
Figura 46 - Detalhes da ferramenta de ajuste elástico. 71
Figura 47 - Realização das APIs de ajuste elástico pela ferramenta de ajuste. 72
Figura 48 - Classes do subsistema de entrada. 73
Figura 49 - Classes do subsistema de entrada. 74
Figura 50 - Visão geral da implementação do algoritmo de ajuste de áudio. 75
Figura 51 - Principais classes do formato genérico de áudio. 76
Figura 52 - Principais classes do formato MP3. 77
Figura 53 - Controladores do ajuste elástico de áudio. 77
Figura 54 - Montagem (incluindo leitura), processamento e desmontagem (incluindo escrita) de dados de áudio. 78
Figura 55 - *Assemblers* e *disassembler* de objetos de formatos de áudio. 80
Figura 56 - Principais classes do subsistema de processamento de áudio. 80
Figura 57 - Visão geral da implementação do algoritmo de ajuste em fluxos de sistemas. 81
Figura 58 - Algoritmo de ajuste de dados audiovisuais. 82
Figura 59 - Tempo de vida das *threads* responsáveis pelo ajuste elástico. 82
Figura 60 - Classes do formato de MPEG-2 sistemas. 84
Figura 61 - Classes dos subsistemas de montagem e desmontagem. 84
Figura 62 - Classes dos subsistemas de multiplexação e demultiplexação. 85
Figura 63 - Controladores do algoritmo de ajuste em fluxos de sistemas. 86
Figura 64 - Classes para realizar ajuste em fluxo elementar de áudio. 87
Figura 65 - Classe para adaptar serviços oferecidos pelo subsistema de ajuste de vídeo à ferramenta de ajuste elástico. 89
Figura 66 - Diagrama das classes dos exibidores de conteúdo. 91

Figura 67 - Diagrama das classes da integração via programação com JMF. 92
Figura 68 - Integração de um formatador hipermídia com uma ferramenta de exibição. 97
Figura 69 - Integração de uma ferramenta de exibição com ajuste elástico com um formatador hipermídia. 99
Figura 70 - Escala de votação utilizada para avaliar cada áudio no teste subjetivo. 106
Figura 71 - MOS obtido para cada tipo de áudio com fator de ajuste 0.90 e 0.95 com coeficiente de confiança 95%. 108
Figura 72 - MOS obtido para cada tipo de áudio com fator de ajuste 1.05 e 1.10 com coeficiente de confiança 95%. 108
Figura 73 - Cálculo da duração de um quadro do fluxo elementar. 110
Figura 74 - Cálculo do tempo total para ajustar fluxo de sistemas. 114
Figura 75 - Exibição de uma mídia com aplicação de ajuste elástico recebidas em tempo de exibição. 129
Figura 76 - Interface gráfica da aplicação de teste. 135

Lista de tabelas

Tabela 1 - Comparação entre camadas do MPEG-1 - fonte: (Noll, 2000; Soares, 2005). 45
Tabela 2 - Duração para exibir as amostras associadas a um quadro (em milisegundos). 47
Tabela 3 - Descrição dos métodos da API para realizar ajuste elástico em tempo de compilação. 67
Tabela 4 - Descrição dos métodos da API para realizar ajuste elástico em tempo de execução. 68
Tabela 5 - Descrição dos métodos de configuração da ferramenta de ajuste elástico. 70
Tabela 6 - Mapeamento dos métodos da API FF com os da API de ajuste elástico em tempo de compilação. 100
Tabela 7 - Mapeamento dos métodos da API FF com os da API de ajuste elástico em tempo de execução. 102
Tabela 8 - Tipos de áudio utilizados nos teste de qualidade. 105
Tabela 9 - Categorias de tipos de áudio recomendadas para testes subjetivos, definidas pela ITU-T - fonte:(ITU, 1998). 105
Tabela 10 - Interpretação de cada nível de qualidade. 106
Tabela 11 - MOS obtido para cada tipo de áudio e fator de ajuste. 107
Tabela 12 - Precisão do processamento de arquivos MP3 (em milisegundos). 112
Tabela 13 - Medida de não-linearidade do processamento de arquivos MP3. 112
Tabela 14 - Efeitos do aumento do retardo em conversa telefônica - fonte: (Soares, 2005). 116
Tabela 15 - Trechos de código para solicitar a geração e exibição de um arquivo ajustado. 127
Tabela 16 - Trechos de código para solicitar a exibição de um fluxo sendo ajustado. 128

Tabela 17 - Hiperdocumento escrito em NCL contendo especificação que precisa da aplicação de ajuste elástico para correta exibição. 134

1
Introdução

A modificação da duração de exibição de um objeto de mídia é realizada por uma técnica conhecida como ajuste elástico. A alteração pode ser quantificada através de um fator de ajuste, expresso por um número real maior que zero. Se um objeto de mídia *M*, cuja exibição dura *d* unidades de tempo, for exibido com fator de ajuste *f*, sua exibição ajustada deverá ter a duração de *d*f* unidades de tempo. Quando *f=1* tem-se a exibição normal da mídia, sem alteração de duração. Uma alteração para acelerar a exibição é expressa por *f<1* e para retardar é expressa por *f>1*.

A técnica de ajuste elástico pode ser aplicada a uma mídia estática (como texto e imagem) ou contínua. Considerando que o ajuste da duração das mídias estáticas é trivial, este trabalho irá focar na descrição de algoritmos para mídias contínuas, mais especificamente o áudio. O termo ajuste elástico é conhecido em Inglês como *timescale*, *elastic time computation*, *time stretching/sketching* e *time compression/expansion*.

As subseções a seguir ilustram a motivação para realização deste trabalho, os objetivos a serem atingidos e a estrutura do restante do documento.

1.1.
Motivação

Muitas aplicações são beneficiadas pelo uso de ajuste elástico (Amir et al., 2000; Arons, 1992; Lee et al., 2004; Omoigui et al., 1999). Estudantes de uma nova linguagem podem desejar escutar gravações com voz mais lenta para melhor compreender a pronúncia dos vocábulos. Músicos também podem desejar escutar uma música mais lentamente para entender melhor uma seqüência rítmica ou melódica. Aplicações de síntese de voz a partir de texto irrestrito podem utilizar ajuste elástico para modificar a duração de um segmento de voz sintetizado de acordo com a prosódia inerente ao contexto. A aplicabilidade principal do algoritmo proposto neste trabalho, entretanto, ocorre em cenários que não existe

interferência do usuário, pois o fator de ajuste é dinamicamente calculado por aplicações clientes.

Como primeiro exemplo, pode-se citar a manutenção da consistência temporal de um documento hipermídia, isto é, a garantia que todos os objetos de mídia serão exibidos no tempo obedecendo às restrições impostas às suas sincronizações relativas, especificadas pelo autor do documento. Por exemplo, em documentos hipermídia, a exibição de um fluxo contendo vídeo e áudio em ambientes distribuídos pode ser prejudicada por atrasos na rede durante a transmissão. Se a transmissão do fluxo de vídeo sofrer retardo maior do que o fluxo de áudio, pode-se modificar a taxa de reprodução de uma das mídias para que a sincronização com os movimentos labiais (*lip-sync*) não seja comprometida. Nesse caso, o ajuste precisa ser aplicado o mais próximo possível do exibidor de conteúdo (*player*) para compensar a variação estatística do retardo da transmissão na rede.

Em outros exemplos, o ajuste deve ser aplicado o mais próximo possível do transmissor. Por exemplo, ajuste elástico pode ser utilizado para permitir que emissoras de rádio e TV acelerem ou retardem a sua programação. Supondo que um determinado comercial precisa ser exibido em um horário fixo, pode ser preciso aplicar ajuste elástico em programas que antecedem a exibição do comercial a fim de compensar possíveis atrasos da programação. Outro exemplo de aplicação ocorre durante a transmissão de servidores de vídeo sob demanda (*Vídeo-On-Demand* ou VOD) para múltiplos usuários. Existem várias técnicas de otimizar a alocação de fluxos de transmissão para usuários. Uma possibilidade é transmitir a informação audiovisual com diferentes velocidades até que os fluxos estejam transmitindo o mesmo trecho de informação em um dado instante. Nesse momento, diferentes fluxos podem ser unificados numa mesma transmissão *multicast*. É importante que a variação de velocidade seja pequena, tipicamente 5% no máximo, para não degradar a qualidade da informação recebida (Soares, 2005; Golubchik et al., 1995). Para implementar essa idéia, pode-se armazenar várias cópias do mesmo conteúdo audiovisual com diferentes velocidade de exibição (Lin et al., 2001) ou aplicar ajuste elástico nos conteúdos durante a transmissão.

Como as aplicações hipermídia envolvem diferentes tipos de mídia, que em geral estão comprimidas, é importante que o algoritmo de ajuste suporte formatos

audiovisuais comprimidos. Além do mais, tais aplicações precisam realizar o ajuste elástico em tempo de exibição e com possibilidade de variação do fator, uma vez que o sincronismo deve ser monitorado e mantido durante toda a apresentação de um documento hipermídia. Aplicar ajuste elástico em tempo de exibição significa processar o fluxo de mídia à medida que ele é recebido, gerando um novo fluxo de dados, com um retardo imperceptível para o usuário. A variação do fator aplicado em tempo de exibição permite modificar a velocidade de exibição das mídias contínuas de acordo com a necessidade corrente da aplicação. A fidelidade da mídia resultante da técnica de ajuste deve ser alta e o processamento deve ser realizado da forma o mais linear possível, de tal modo que a variação de velocidade na exibição da mídia resultante seja mínima.

É também interessante que a aplicação de ajuste seja capaz de monitorar o novo valor de um instante específico da mídia original durante a realização do ajuste. Por exemplo, considerando que documentos hipermídia utilizam âncoras[1] para indicar pontos de sincronismo, é importante conhecer quais são os novos valores dessas âncoras após o processamento.

Uma vez que a realização de ajuste não implica necessariamente que a mídia ajustada deva ser imediatamente exibida (podendo ainda sofrer outras transformações ou mesmo ser enviada pela rede) e, principalmente, porque o mecanismo de ajuste não deve ser dependente de qual exibidor será utilizado, é também importante que a aplicação de ajuste elástico seja independente da decodificação que ocorre no exibidor do conteúdo do fluxo de mídia.

Por fim, é importante que a aplicação cliente possa facilmente solicitar o ajuste elástico em qualquer local da transmissão, ou seja, no transmissor, receptor ou em um nó intermediário, e que seja possível manipular não somente arquivos locais, como também remotos, e até mesmo fluxos de mídia de dados “ao vivo” (dados de mídia que não estão armazenadas em arquivos, mas transmitidos como um fluxo de bits na rede).

[1] Uma âncora identifica um subconjunto de unidades de informação de um objeto de mídia. No contexto deste trabalho, uma âncora marca um instante de tempo em um fluxo audiovisual.

1.2. Objetivos

O objetivo principal deste trabalho é propor um algoritmo de ajuste que atue em áudio comprimido e que atenda aos requisitos das aplicações descritas na seção anterior.

O algoritmo de ajuste será então integrado a um algoritmo de fluxos de vídeos através de desenvolvimento de um terceiro algoritmo para fluxos de sistemas[2].

O algoritmo de ajuste de áudio irá manipular fluxos de mídia cujo formato de dados é divido em quadros, que são unidades independentes (ou quase independentes) de decodificação. Como exemplo, o algoritmo será instanciado para os formatos MP2 e MP3, especificados pelos padrões MPEG-1 (ISO, 1993), MPEG-2 BC (*Backward Compatible*) (ISO, 1998), AC-3 (ATSC, 1995) e MPEG-2/4 AAC (ISO, 1997; ISO, 2001b).

O algoritmo de vídeo a ser utilizado foi desenvolvido por Cavendish (Cavendish, 2005) e contempla o formato MPEG-2 vídeo (ISO, 2000b). Finalmente, o mecanismo de ajuste de fluxos de sistemas será desenvolvido para o MPEG-2 sistemas (ISO, 2000a).

Como último objetivo deste trabalho, a ferramenta de ajuste deverá ser integrada a um formatador hipermídia, o HyperProp. O formatador HyperProp é um controlador de apresentações hipermídia que incorpora mecanismos de otimização para ajuste elástico das apresentações, a fim de manter o sincronismo temporal dos documentos exibidos (Rodrigues, 2003).

1.3. Organização do restante do documento

O restante deste documento está estruturado como se segue. O Capítulo 2 classifica os algoritmos de ajuste elástico de áudio existentes na literatura, apresenta alguns trabalhos relacionados e os compara com o trabalho proposto.

O Capítulo 3 descreve o mecanismo de ajuste elástico de áudio desenvolvido, assim como sua particularização para cada formato suportado e sua execução como fluxo elementar de um fluxo de sistemas. A ferramenta de ajuste é

[2] Um fluxo de sistema contém dados de áudio e vídeo multiplexados.

descrita no Capítulo 4. Esse capítulo apresenta como ferramentas clientes podem solicitar serviços de ajuste elástico através de APIs de ajuste e detalhes da implementação da ferramenta, o que inclui a implementação do mecanismo apresentado no Capítulo 3 e a integração do ajuste de áudio com o vídeo. O Capítulo 5 apresenta o mecanismo de integração desenvolvido para possibilitar a apresentação da mídia ajustada em exibidores de conteúdo.

O Capítulo 6 apresenta a integração da ferramenta de ajuste em sistemas hipermídia, em especial com o formatador Hyperprop. O Capítulo 7 apresenta medidas de qualidade, subjetivas e objetivas, extraídas do algoritmo de ajuste de áudio proposto por este trabalho. As análises subjetivas foram realizadas com base em um experimento também descrito nesse capítulo. Além disso, esse capítulo também realiza uma análise da implementação da ferramenta de ajuste descrita no Capítulo 4.

O Capítulo 8 apresenta conclusões e trabalhos futuros e o Capítulo 9 lista referências utilizadas no decorrer deste trabalho.

2
Trabalhos Relacionados

Este capítulo faz uma categorização dos algoritmos de ajuste elástico para áudio comprimido existentes na literatura, lista exemplos de ferramentas de ajuste de áudio relevantes e realiza uma comparação entre elas e o algoritmo de ajuste de áudio proposto.

Trabalhos relacionados de algoritmos e ferramentas de ajuste em fluxos audiovisuais e somente visuais são descritos por Cavendish (Cavendish, 2005).

2.1.
Categorização de algoritmos de ajuste para áudio

Alguns trabalhos propõem uma categorização dos algoritmos de ajuste elástico, como em (Arons, 1992; Lee et al., 2004; Bernsee, 2003). As subseções a seguir descrevem classes de algoritmos de ajuste, de acordo com os trabalhos de Arons e Lee (Arons, 1992; Lee et al., 2004), em ordem crescente de qualidade e custo computacional.

2.1.1.
Reprodução rápida/lenta

Os algoritmos dessa categoria modificam a taxa de exibição das amostras da mídia, o que acarreta uma modificação do tempo necessário para a apresentação da mídia. No entanto, tais algoritmos possuem o efeito indesejável de alterar as freqüências componentes do sinal. A referência (Lemay, 1998) apresenta um *applet* que efetua uma série de cálculos relacionados à aplicação dessa categoria de ajuste elástico e à alteração das freqüências componentes do sinal.

A Figura 1 ilustra a modificação do sinal representado no domínio do tempo quando seu tempo de exibição foi duplicado. A metade de cima da figura ilustra o áudio original e a metade inferior mostra o áudio processado. Nesse caso, o áudio perdeu altas freqüências (Nyquist) e se tornou mais grave.

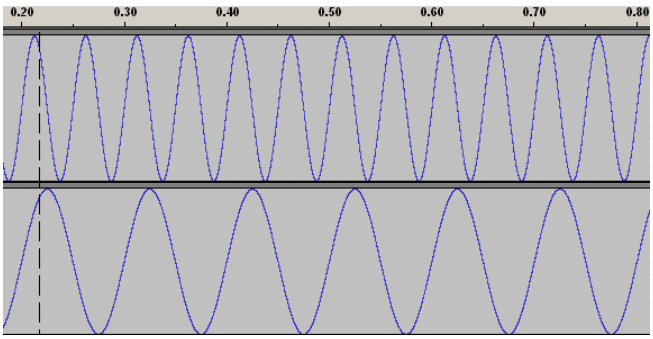


Figura 1 - Mudança da taxa de exibição de um sinal.

Embora essa categoria possa ser considerada um caso particular da categoria no domínio do tempo, descrita a seguir, esse trabalho optou por descrevê-la separadamente devido aos efeitos resultantes típicos nos áudios processados por algoritmos dessa categoria, que é análogo ao áudio resultante da alteração de velocidade de reprodução em fitas e LPs. O áudio resultante é modificado, porém inteligível.

2.1.2. No domínio do tempo

Essa classe de algoritmos processa o áudio no domínio do tempo. O áudio é divido em pequenos quadros e o ajuste é realizado manipulando-os. A idéia de dividir o áudio em pequenos pedaços no domínio do tempo para efetuar algum processamento é bem antiga, datando de 1946 (Gabor, 1946).

È importante ressaltar que as modificações realizadas no domínio do tempo influem em outros domínios do sinal, como freqüência (e vice-versa). A Figura 2 ilustra um exemplo de aplicação de algoritmos no domínio do tempo. Na figura, pequenos quadros do sinal foram removidos, acelerando a reprodução sem perder altas freqüências, mas degradando a qualidade do sinal.

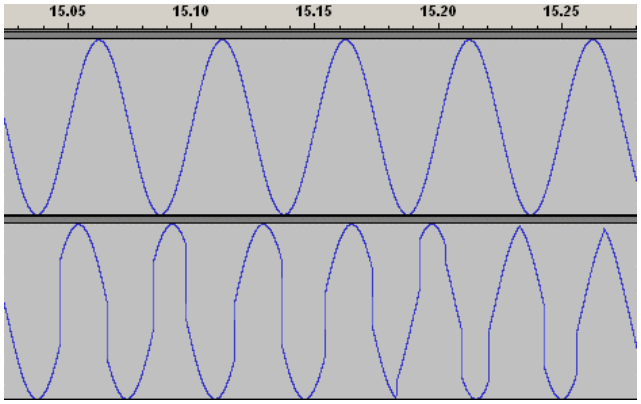


Figura 2 - Exemplo de cortes de quadros de um sinal.

Esses algoritmos possuem baixo custo de processamento. No entanto, a qualidade é limitada a uma pequena variação da taxa. Tipicamente, esses algoritmos atingem boa qualidade com fator de ajuste limitado ao intervalo *0.8<f<1.2* (Lee et al., 2004).

O *Ajuste Regular* é um exemplo de algoritmo dessa classe que descarta ou duplica quadros de áudio a intervalos regulares sem considerar o conteúdo do sinal. Na Figura 3 cada número representa um segmento de áudio. O áudio original possui *6* quadros. É possível reduzir a duração do áudio à metade descartando um segmento sim e outro não, indicado na figura com *3* quadros. Para duplicar o tempo do áudio, é necessário replicar todos os quadros, ilustrado na figura com *12* quadros.

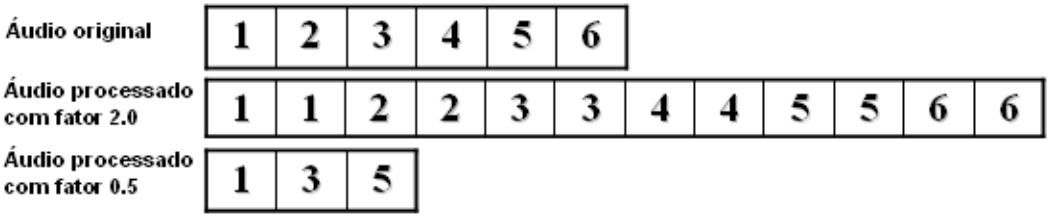


Figura 3 - Algoritmo *Ajuste Regular*.

Quando aplicado à voz humana, o tamanho de tempo de um segmento de áudio deve ser maior do que o tempo necessário para mudar freqüências do sinal de voz (por exemplo, maior do que *15ms*) e menor do que o tempo de pronunciar um fonema de modo a minimizar a degradação da qualidade do áudio (Portnoff, 1981).

Outros algoritmos dessa classe tentam melhorar o desempenho do *Ajuste Regular*. Aron (Arons, 1994) sugere reproduzir os quadros descartados de um canal em outro canal deslocado no tempo. Desse modo, só ocorrem perdas do sinal original quando se utiliza um fator menor do que *0.5*. Na Figura 4 cada número representa um segmento de áudio. O áudio original possui *9* quadros. É possível reduzir a duração do áudio descartando um segmento sim e outro não. Os quadros não descartados são reproduzidos no canal esquerdo e os demais no canal direito. Esse algoritmo aumenta inteligibilidade e compreensão do ouvinte depois de rápida sensação de confusão.

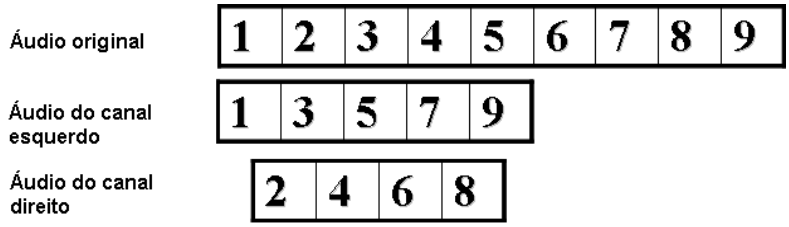


Figura 4 - Ajuste elástico com reprodução dos quadros intercalados.

Outros algoritmos sugerem que o segmento a ser descartado ou duplicado deve ser escolhido de acordo com uma análise (simples) das características do sinal. O algoritmo é aplicável a áudios bem-comportados, como a voz humana. Alguns estudos sugerem que é possível retirar até *50%* do silêncio entre palavras e sentenças sem prejuízos à compreensão (Soares, 2005). No entanto, a proporção de remoção do silêncio da voz é assunto de bastante discussão na literatura (Arons, 1992).

Por fim, alguns algoritmos sugerem que os quadros não sejam simplesmente descartados (ou duplicados), mas sim interpolados. Esse é caso dos conhecidos algoritmos como OLA (*Overlap Add Method*) e SOLA (*Synchronized Overlap Add Method*).

2.1.3. No domínio da freqüência

Os algoritmos dessa classe realizam o ajuste manipulando as freqüências componentes do sinal de áudio. O exemplo mais representativo é o algoritmo *Phase Vocoder (*Hammer, 2001).

A idéia desse algoritmo é similar ao *Ajuste Regular*, no entanto a alteração da duração do sinal é realizada no domínio da freqüência. Seu objetivo é alterar o número de ciclos de freqüências componentes de um sinal, sem mudar quais são as freqüências. A idéia para realizar o ajuste elástico é dividir o sinal original em quadros e realizar o ajuste elástico alterando o tamanho desses quadros. A Figura 5 ilustra um exemplo de ampliação em dois quadros da duração do sinal utilizando o ajuste.

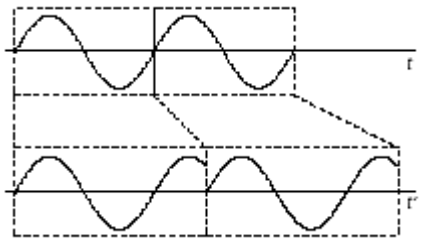


Figura 5 - Ajuste elástico de quadros do sinal no domínio da freqüência.

O problema da alteração descrita é que as extremidades dos quadros alterados não estão alinhadas em relação à fase do sinal, o que introduz altas freqüências. Além disso, quando o sinal é composto por mais de uma freqüência, o algoritmo trata cada freqüência independentemente, o que causa reverberação no sinal processado. Existem alguns trabalhos que aprimoram esse algoritmo (Laroche & Dolson, 1999; Hammer, 2001).

2.1.4. Baseados em análise detalhada

São algoritmos que realizam ajuste elástico após uma análise detalhada do sinal. Tais algoritmos, em geral proprietários, geram áudio de alta qualidade e exigem grande custo computacional, não sendo aplicáveis no tempo de exibição.

O *MPEX* (Prosoniq Mpex, 2004) é um algoritmo que utiliza rede neural treinada para simular algumas propriedades da percepção humana e obter um áudio processado com excelente qualidade. A grande vantagem desse algoritmo é que ele não é baseado estritamente em rígidos modelos matemáticos.

2.2. Ferramentas que realizam ajuste elástico

A maioria das ferramentas comerciais possui algoritmos da categoria do domínio do tempo e da freqüência, uma vez que a reprodução rápida/lenta modifica bastante o áudio original e o custo computacional dos algoritmos baseados em análise detalhada é muito alto. As subseções a seguir detalham os trabalhos relacionados mais representativos para o contexto deste trabalho. Pelo que foi observado, todas as ferramentas citadas realizam o ajuste de modo o mais linear possível e obviamente tentam preservar a fidelidade ao máximo.

2.2.1.
Sound Forge

O *Sound Forge* (Sony, 2005) é um *software* para gravação e edição profissional de áudio. A ferramenta oferece suporte a vários formatos de áudio, no entanto, sempre realiza um pré-processamento para abrir arquivos comprimidos.

A funcionalidade de ajuste elástico é acessada através da janela apresentada na Figura 6. Nela, o usuário seleciona o modo e o fator a ser aplicado. A operação de ajuste elástico pode ser experimentada em tempo de exibição, provavelmente utilizando o arquivo pré-processado. No entanto, somente quando o usuário fechar essa janela (clicando em OK) é que o *software* gera o fluxo de dados comprimido, resultante da operação, e essa ação não é realizada em tempo de exibição.

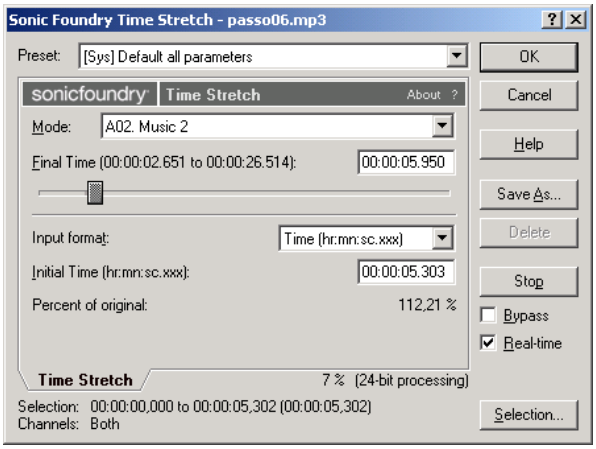


Figura 6 - Interface de aplicação de ajuste elástico em arquivos de áudio no *Sound Forge*.

Existem *19* modos diferentes de aplicar ajuste elástico para o usuário escolher em função de suas necessidades. Alguns modos são adequados para preservar a qualidade da voz, ou do som de algum instrumento (como a bateria), enquanto outros modos tentam preservar a qualidade do áudio como um todo. O fator de ajuste pode variar entre *0.5* e *5*.

2.2.2.
Windows Media Player 10

O *Windows Media Player 10* (Microsoft, 2005) é um *software* que permite exibir conteúdo audiovisual dos formatos Windows Media Áudio (WMA), MP3 e ASF. A ferramenta permite selecionar um fator de ajuste a ser utilizado para a reprodução do fluxo de mídia, como ilustrado na Figura 7, e modificá-lo em

tempo de exibição. Embora o fator possa variar entre *0.0625* e *16*, o limite recomendado pela especificação do programa é entre *0.5* e *2.0* para manter alta qualidade.

Ainda que a especificação do programa não indique qual o algoritmo de ajuste elástico utilizado, supomos que essa ferramenta processa o áudio depois da decodificação por dois motivos. O primeiro é que o *Windows Media Player* precisa descomprimir o áudio para exibi-lo e nesse cenário é mais vantajoso aplicar o algoritmo de ajuste no áudio sem compressão (já que é menos custoso) e o segundo motivo é que não é possível salvar o áudio processado em formato comprimido.

Utilizando o *Microsoft Windows Media Player 10 Software Development Kit (SDK)* é possível interagir programaticamente com *Window Media Player*. Dentre as possibilidades, pode-se monitorar marcações de tempo e exibir não somente arquivos, mas também fluxos audiovisuais sendo processados em tempo de exibição.

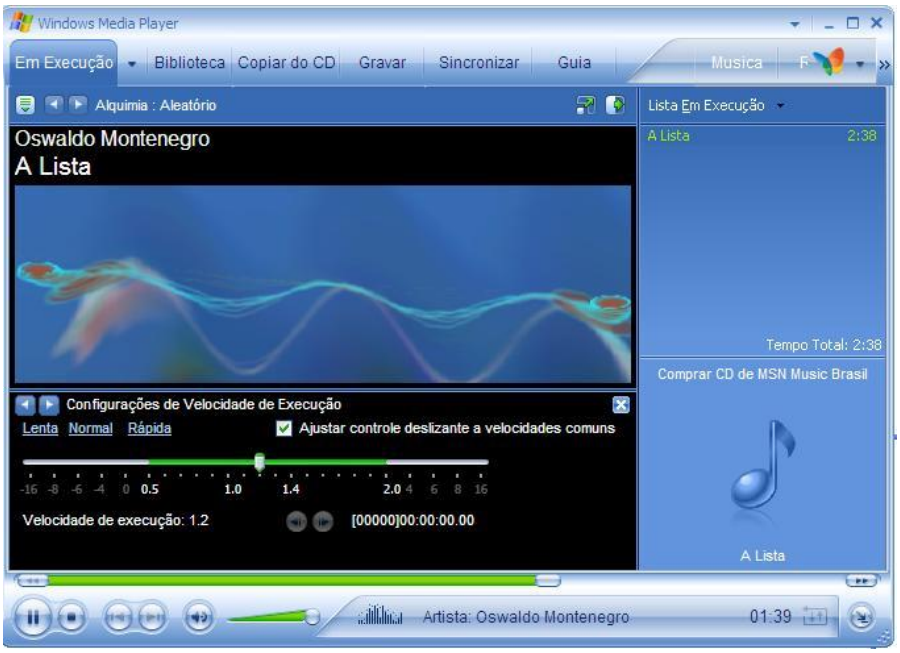


Figura 7 - Interface de aplicação de ajuste elástico em arquivos de áudio no *Window Media Player 10*.

2.2.3. Amazing Slow Downer

O *Amazing Slow Downer* (Roni Music, 2005) é um *software* que permite exibir áudio nos formatos MP3, Wave e Windows Media Áudio (WMA) com uso

de ajuste elástico em tempo de exibição para fatores variando entre *0.5* e *4*, sendo possível alterá-lo durante a apresentação. A Figura 8 mostra a tela principal da ferramenta.

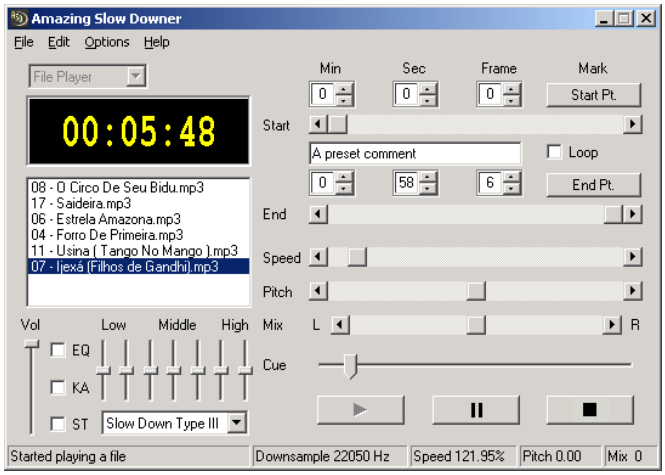


Figura 8 - Interface do *software Amazing Slow Downer*.

A ferramenta disponibiliza três algoritmos diferentes para efetuar ajuste elástico, com crescente uso de recursos computacionais e níveis de qualidade. Pelos mesmos motivos expostos para o *Windows Media Player*, supomos que essa ferramenta aplica o ajuste no áudio sem compressão. Nessa ferramenta é possível salvar os arquivos que foram ajustados, mas somente no formato Wave.

2.2.4. Enounce 2xAV

Enounce 2xAV (Enounce, 2003) é um *plug-in* que adiciona uma barra de controle de taxa de exibição aos programas *RealPlayer*, *RealOne Player* e *Windows Media Player*. Utilizando o *plug-in*, é possível aplicar ajuste elástico em tempo de exibição a fluxos de vídeo e áudio comprimidos com fator *f* dentro dos limites $0.3 \leq f \leq 2.5$ (como ilustra a Figura 9), sendo possível variar o valor do fator aplicado também durante a exibição. Mais uma vez, supomos que essa ferramenta aplica o ajuste no áudio sem compressão pelos motivos já expostos para o *Windows Media Player*.

Figura 9 - Interface do *Enounce* 2xAV.

2.2.5.
585 Time Scaling Processor

O *hardware 585 Time Scaling Processor* (Dolby, 2005b) é um exibidor de mídia que permite realizar ajuste elástico em áudio sem compressão com até oito canais com fatores de ajuste variável entre *0.85* e *1.15*.

Os algoritmos de ajuste elástico utilizados são proprietário da empresa *Dolby* (Dolby, 2005a). Se o requisito de tempo de exibição não for necessário, o ajuste elástico gera áudios com excelente qualidade (distorção máxima de *0.01%* quando o áudio de entrada possui freqüências de até *1kHz* e de *0.02%* quando possui freqüências de *20Hz* até *20kHz*). Este algoritmo é ideal para pequenos programas (cerca de *23* minutos para áudio com um canal e de apenas *3* minutos para áudio com oito canais), já que o *hardware* pode armazená-lo para posterior exibição. No entanto, se for necessário aplicar o ajuste elástico em tempo de exibição, o processador aplicará um algoritmo da categoria 2.1.1, ocasionando mudança proporcional nas freqüências do sinal. A Figura 10 ilustra a interface do processador.

Figura 10 - Interface de controle do *585 Time Scaling Processor*.

2.2.6.
DIRAC

O *DIRAC* (Bernsee, 2005) é uma biblioteca que permite realizar ajuste elástico em áudio não comprimido em tempo de exibição com possibilidade de mudança do fator, que pode variar entre *0.5* e *2.0*. A biblioteca é escrita em C/C++ e é distribuída em três versões, sendo uma delas gratuita.

O *DIRAC* manipula apenas sinais sem compressão, sendo assim, se for aplicar ajuste a sinais comprimidos, é necessário primeiro decodificá-lo e, se necessário, depois codificá-lo novamente.

2.2.7. FastMPEG

O *FastMPEG* (Covell et al., 2001) propõe três algoritmos de ajuste elástico para áudio MP2, um deles da categoria reprodução rápida/lenta e dois do domínio do tempo. Todos os algoritmos são aplicados após uma decodificação parcial do fluxo de dados, seguidos, então, por uma codificação parcial. Segundo os autores, os algoritmos funcionam em tempo de exibição, mesmo com o custo do pré- e pós-processamento. O fator de ajuste é variável entre o intervalo de *2/3* a *2.0*.

Existem quatro passos no funcionamento do *FastMPEG*, ilustrados na Figura 11. No primeiro passo, o fluxo MPEG é analisado, os fatores de escala são removidos e os *30* MDSSs[3] são separados. No segundo, algum dos três algoritmos de ajuste elástico é aplicado em paralelo em todos os *30* MDSSs. O modelo de mascaramento psicoacústico é inferido e modificado no terceiro passo. Por último, o fator de escala é novamente aplicado, os novos sinais são quantizados com o novo modelo de mascaramento psicoacústico e os bits são organizados para compor um novo fluxo MPEG.

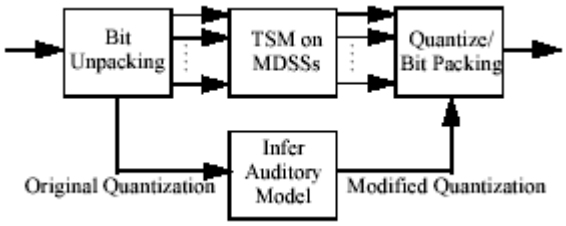


Figura 11 - Estrutura de funcionamento do *FastMPEG*.

A qualidade dos áudios modificados pelo *FastMPEG* é comprometida principalmente pelo algoritmo de ajuste, por pré-supostos da codificação que não são mais verdadeiros quando o sinal é modificado e pelo modelo de mascaramento psicoacústico recuperado. Não existe uma ferramenta disponível com a implementação dos algoritmos, mas alguns resultados de mídias ajustadas utilizando tais algoritmos são apresentados por Covell (Covell et al., 2001).

[3] O codificador MPEG aplica uma ou mais transformadas nas amostras, dividindo-as em sub-bandas. Uma sub-banda contém um conjunto de amostras correspondente a uma faixa do espectro de freqüências audível. Cada conjunto é conhecido como MDSS (*maximally decimated subband streams*).

2.2.8.
PICOLA do MPEG-4

O padrão de áudio do MPEG-4 (ISO, 2001b) codifica sinais de voz humana e áudio multicanal com alta qualidade e também oferece suporte a áudios naturais e sintéticos. Por tratar sinais tão distintos, o MPEG-4 áudio possui diferentes mecanismos de codificação de acordo com as características do áudio e da taxa de bits a ser atingida.

Um decodificador MPEG-4 de áudio, como o software de referência MPEG (Moving Picture Experts Group, 2001), deve saber manipular todos os diferentes formatos de áudio e poder aplicar ferramentas de efeito ou mixagem no áudio decodificado antes de o enviar para exibição. A Figura 12 ilustra esse tipo de decodificador.

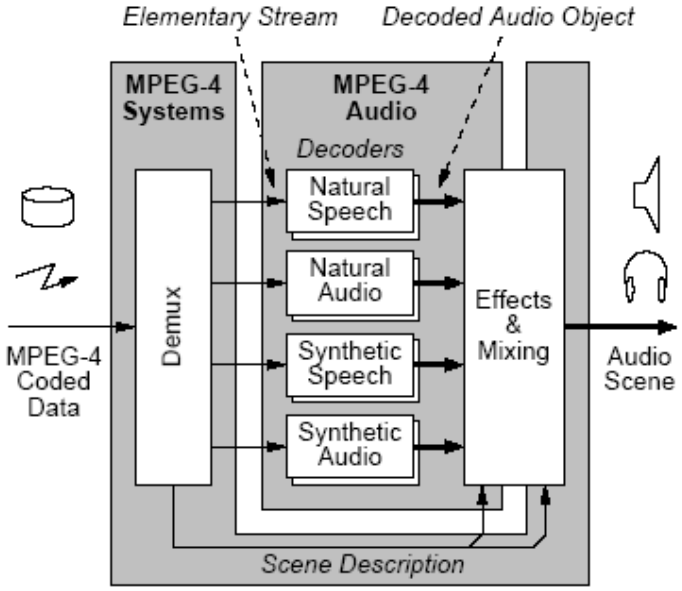


Figura 12 - Decodificador MPEG-4 - fonte: (Wolters & Kjörling, 2003).

Um exemplo de ferramenta de efeito é a chamada *PICOLA* (*Pointer Interval Controlled OverLap Add*), que permite realizar ajuste elástico utilizando algoritmos do domínio do tempo em sinais de áudio MPEG-4 monofônicos com taxa de amostragem de *8kHz* ou *16kHz*. O ajuste pode ser realizado em tempo de exibição com fator variável entre *0.5* e *2.0*.

2.2.9. Padrão sines + transients + noise

Levine (Levine & Smith, 1998) propõe um novo padrão de áudio comprimido que facilita a realização de processamento no domínio comprimido. O padrão proposto é capaz de produzir áudio com taxas altas de compressão (*16-48kbps*) com qualidade similar a do MPEG-2 AAC, quando se compara utilizando uma taxa de *32kbit/canal* em ambos os padrões.

O padrão especifica que o áudio deve ser dividido em três sinais independentes: *sine*, *transient* e *noise*. O sinal *transient* contém os sons de ataques do áudio original. O sinal *sines* é uma soma de freqüências da região de *0* a *5kHz*. O sinal *noise* modela altas freqüências que não estão no *transient*. Essa separação permite que cada uma das três partes possa ser manipulada de modo diferenciado e eficientemente codificada.

É possível aplicar ajuste elástico de boa qualidade no fluxo modificando apenas os sinais *sine* e *noise*. O sinal *transient* é apenas deslocado para acompanhar a mudança dos demais sinais, mantendo seu envelope temporal. Desse modo é possível realizar ajuste elástico conservando os ataques do áudio original. Numa música, por exemplo, é possível realizar ajuste elástico em instrumentos harmônicos e vozes e ainda manter os ataques de instrumentos de percussão. Alguns exemplos de áudios processados nesse formato são apresentados por Levine (Levine, 1998).

2.3. Comparação dos trabalhos relacionados com o proposto

A comparação dos trabalhos citados com o proposto por este trabalho deve ser realizada considerando os requisitos das aplicações descritas na Seção 1.1: tipo de sinal processado, tempo de processamento, possibilidade de variação do fator em tempo de exibição, amplitude do fator de ajuste, fidelidade, linearidade, independência do exibidor de conteúdo, suporte a dados “ao vivo” e a monitoramento de âncoras.

A primeira análise comparativa diz respeito ao tipo de sinal a ser processado em tempo de exibição. Os trabalhos *Windows Media Player 10*, *Amazing Slow Downer*, *Enounce 2xAV*, *FastMPEG* e *PICOLA* podem manipular sinais de áudio comprimidos em tempo de exibição. O *DIRAC* e *585 Time Scaling Processor* não

oferecem essa funcionalidade e o *Sound Forge* precisa primeiramente pré-processar o áudio para descomprimi-lo (ver Subseção 2.2.1). Vale ainda ressaltar que o *PICOLA* do MPEG-4 só oferece suporte a áudios monofônicos com determinadas taxas de amostragem.

Todos os algoritmos apresentados permitem modificar o fator de ajuste em tempo de exibição (exceto o *Sound Forge*), possuem uma amplitude do fator de escala maior ou igual a *10%*, tentam preservar a fidelidade ao máximo e realizam o ajuste de modo o mais linear possível, assim como o algoritmo proposto.

Em relação à independência do exibidor de conteúdo, uma análise mais detalhada é necessária. Embora não seja claro como os algoritmos de *Sound Forge*, *Windows Media Player*, *Amazing Slow Downer* e *Enounce 2xAV* funcionam, supomos que essas ferramentas processam o áudio depois da decodificação. Sendo assim, o algoritmo de ajuste dessas ferramentas é dependente do exibidor de conteúdo (que, muitas vezes, é a própria ferramenta). A especificação do *PICOLA* do MPEG-4 define que novos decodificadores devem prover ajuste elástico e funciona para esses decodificadores. O *DIRAC* e *585 Time Scaling Processor* não manipulam áudio comprimido, por isso, fica fácil para tais ferramentas serem independentes do exibidor de conteúdo. O *FastMPEG* atua diretamente no domínio comprimido, gerando um novo fluxo de áudio comprimido. No entanto, o algoritmo que o *FastMPEG*[4] utiliza é fortemente dependente do modo como arquivos MPEG BC são codificados, sendo bastante complicado generalizar um algoritmo desse tipo para outros formatos de áudio.

Em síntese, todos os algoritmos de ajuste elástico citados não manipulam o fluxo de áudio comprimido diretamente, adotando a opção de decodificar (ainda que parcialmente), processar e, se necessário, codificar novamente o fluxo de áudio. Essas soluções são custosas e dependentes da decodificação (e muitas vezes do exibidor de conteúdo). O mecanismo de ajuste proposto por este trabalho opera diretamente no domínio comprimido, sendo independente do processo de decodificação.

Vale ainda destacar que embora seja interessante definir um novo padrão de áudio, como o mencionado na Subseção 2.2.9, que facilite a realização de

[4] A referência do *FastMPEG* é um artigo de resumo, mas nenhuma aplicação executável parece estar disponível na Internet.

transformações como o ajuste elástico, é difícil acreditar que este formato conquistará o mercado de áudio rapidamente, principalmente sem o apoio de um órgão conhecido de padronização e grandes empresas. Sendo assim, este trabalho opta por manipular formatos de áudio padronizados e maciçamente utilizados.

Por fim, é interessante comparar as características da ferramenta de ajuste proposta com as demais. Apenas o *Windows Media Player* (via SDK), o *DIRAC*, o *FastMPEG* e o algoritmo proposto foram desenvolvidos com objetivo de facilitar integração via programação com outras aplicações. Duas características de uma ferramenta de ajuste merecem destaque: suporte a dados “ao vivo” e monitoramento de marcações de tempo. O *Windows Media Player* (via SDK) e o trabalho proposto oferecem suporte a essas características. O DIRAC, como mencionado, não funciona para dados comprimidos e a especificação do *FastMPEG* não deixa claro se a ferramenta possui tais características.

Em resumo, percebe-se que nenhum dos trabalhos relacionados encontrados atende satisfatoriamente a todos os requisitos propostos. Em todo material pesquisado na literatura nenhuma ferramenta foi encontrada com propósito e solução similares aos apresentados por este trabalho.

3
Ajuste Elástico de Áudio no Fluxo Comprimido

Este capítulo descreve primeiramente uma análise resumida do algoritmo proposto e a seguir a modelagem do fluxo de áudio comprimido adotada, o mecanismo genérico de ajuste de áudio proposto e a instanciação do mecanismo para alguns formatos de áudio. Por fim, o capítulo descreve como o algoritmo de ajuste pode ser aplicado em um fluxo de sistemas.

3.1.
Análise Resumida do Ajuste Elástico Proposto: Requisitos do Algoritmo

O problema de realizar ajuste elástico em uma mídia com compressão pode ser resolvido pelo menos de três maneiras diferentes, como já discutido quando da apresentação das soluções existentes. A primeira opção é realizar um pré- e pós-processamento de modo a somente aplicar o algoritmo de ajuste na mídia sem compressão. Nesse cenário, é preciso primeiro decodificar o fluxo, realizar o ajuste e, em seguida, codificá-lo novamente, como ilustrado na Figura 13.

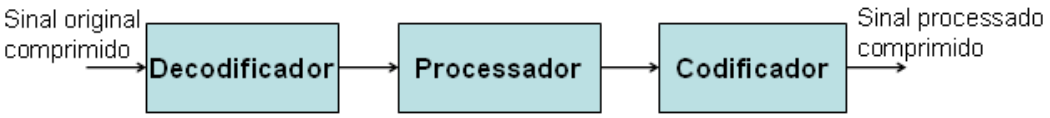


Figura 13 - Realização do ajuste em fluxo comprimido, utilizando decodificação prévia e nova codificação.

A vantagem dessa opção é que existem vários bons algoritmos de ajuste elástico para mídias sem compressão propostos na literatura (Lee et al., 2004). No entanto, essa estratégia é bastante custosa em termos de processamento, dificultando seu uso em tempo de exibição. Além disso, existe perda de qualidade inerente a uma nova compressão.

Outra possível solução é realizar o ajuste depois da descompressão (e antes da exibição), como ilustra a Figura 14. É fácil perceber que esse cenário é útil quando a recodificação não é mais necessária. A vantagem dessa opção é que o

algoritmo de ajuste ainda irá manipular um fluxo sem compressão. Por outro lado, essa opção precisa ser realizada de modo dependente do decodificador.

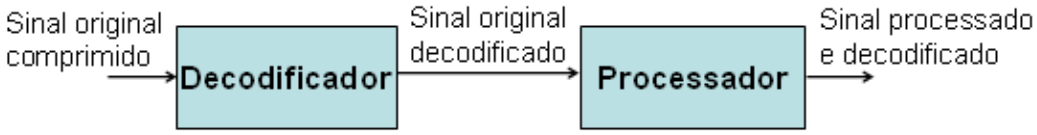


Figura 14 - Realização do ajuste em fluxo comprimido, utilizando decodificação prévia.

A última abordagem é realizar o ajuste antes de enviar o fluxo de dados para a descompressão, diretamente no fluxo de dados comprimido, como ilustra a Figura 15. A vantagem dessa opção é ter um ajuste computacionalmente mais barato e totalmente independente do decodificador. No entanto, o ajuste elástico deve ser realizado considerando as regras de formatação dos dados especificadas pelo padrão de compressão.

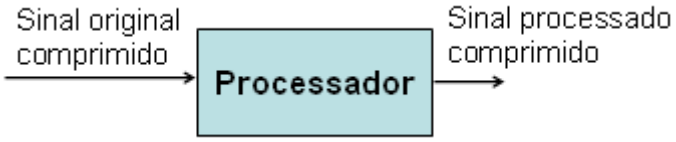


Figura 15 - Realização do ajuste diretamente no fluxo comprimido.

Considerando que o ajuste precisa ser realizado em tempo de exibição e independente do exibidor de conteúdo, a última abordagem foi adotada. Por isso, no decorrer deste trabalho será dito que o algoritmo de ajuste é aplicado diretamente no domínio comprimido.

O mecanismo de ajuste deste trabalho utiliza um fator de ajuste que pode ser modificado em tempo de exibição. Para preservar alta fidelidade em relação ao conteúdo original e atingir um processamento linear aceitável, o fator de ajuste deve variar em até *10%*, ou seja, entre o intervalo de *0.90* a *1.10*. É esperado que esse intervalo já seja suficiente para a manutenção do sincronismo entre mídias nas plataformas de exibição de documentos hipermídia atuais. Além disso, para dados audiovisuais, o vídeo correlacionado e multiplexado normalmente não tolera variação maiores que *5%*, sem perda de qualidade (Golubchik et al., 1995). Para comprovar que os requisitos de fidelidade e linearidade são atendidos, medidas subjetivas e objetivas de qualidade serão extraídas de mídias processadas (ver Seção 7.1).

Para atingir o requisito de independência do exibidor de conteúdo, o algoritmo de ajuste elástico deve preservar as mesmas características do fluxo

original, tanto em relação ao formato do fluxo como às características de codificação. Este trabalho considera importante, além da independência do exibidor, ser também independente do processo de decodificação, já que desse modo o mesmo algoritmo pode ser facilmente aplicado a vários formatos de mídia.

Como último requisito, o algoritmo deve ser desenvolvido como uma ferramenta de fácil acesso às aplicações clientes, chamada ferramenta de ajuste (ver Capítulo 4). Os clientes podem solicitar a realização do ajuste no local adequado, que pode ser no transmissor, no receptor ou em algum nó intermediário. A ferramenta de ajuste deve ser capaz de manipular arquivos locais, arquivos remotos e fluxos de mídia de dados "ao vivo", além de permitir monitorar marcações de tempo.

3.2. Modelagem do Fluxo de Áudio Com Compressão

Grande parte dos formatos atuais de áudio comprimido de alta qualidade define que um fluxo de mídia é constituído por um conjunto de quadros. O modelo adotado por este trabalho procura generalizar os formatos hoje existentes. Nele, cada quadro é univocamente identificado e possui um cabeçalho e um campo de dados. Além disso, cada quadro tem associado um conjunto de amostras referentes a um curto intervalo de tempo. O conjunto de amostras associado a um quadro, quando comprimidas, formam uma *unidade lógica de dados*.

A Figura 16 ilustra um exemplo de parte de fluxo de áudio com quadros divididos por grandes traços verticais. Os bits referentes ao cabeçalho são verticalmente listrados e se localizam sempre no início do quadro. Na figura, cada quadro transporta seu cabeçalho e sua unidade lógica de dados, e o campo de dados dos quadros sempre coincide com sua unidade lógica de dados. Entretanto, de modo mais geral, esses campos não são necessariamente iguais. O tamanho de uma unidade lógica de dados é diferente para cada quadro. Os dados de uma única unidade lógica associada a um quadro podem estar contidos no campo de dados do próprio quadro, ou também em campos de dados de quadros imediatamente anteriores. O conjunto de bits que armazenam amostras de quadros posteriores é denominado *reservatório de bits*, como ilustrado na Figura 17.

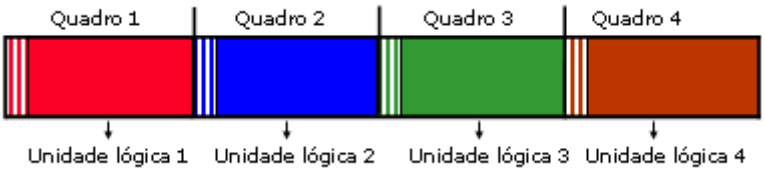


Figura 16 - Representação dos quadros que compõem um fluxo de áudio com unidade lógica de dados igual ao campo de dados.

Na Figura 17, quadros, como na figura anterior, são também divididos por grandes traços verticais e os bits referentes ao cabeçalho são verticalmente listrados e se localizam sempre no início do quadro. A unidade lógica do quadro *1*, entretanto, está ocupando parte do seu campo de dados e parte do campo de dados do quadro anterior. Os bits restantes no campo de dados do quadro *1* são suficientes para armazenar toda unidade lógica do quadro *2* e parte da unidade lógica do quadro *3*. O quadro *2* armazena somente a unidade lógica do quadro *3*. O quadro *3* armazena parte da sua unidade lógica e o início da unidade lógica do quadro *4*. Os bits iniciais e finais da figura não pertencem a nenhuma unidade lógica de dados e são representados apenas para indicar que o fluxo de mídia é maior do que os quatro quadros.

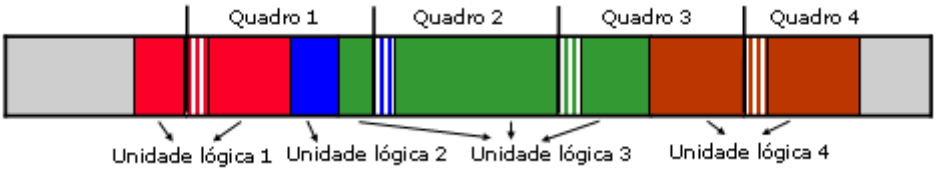


Figura 17 - Representação dos quadros que compõem um fluxo de áudio.

O modelo de fluxo de áudio comprimido ainda permite que no final de cada unidade lógica de dados existam dados de enchimento (PAD), ou seja, dados que não são provenientes da codificação das amostras daquele quadro. Esses dados podem representar algum metadado padronizado ou não. O tamanho do PAD não é limitado, mas os bits da unidade lógica do quadro nunca podem ocupar campo de dados de quadros posteriores. Por outro lado, podem existir bits de PAD em quadros anteriores, apenas se a unidade lógica de dados associada a esse quadro terminar antes mesmo do campo físico do quadro começar. A Figura 18 ilustra o mesmo fluxo de dados da Figura 17, mas apresenta que no final da unidade lógica de dados dos quadros *1* e *3* existem bits de PAD. Os bits de PAD estão ilustrados com a cor branca. Além dos dados de enchimento dentro das unidades lógicas de

dados, também podem existir bits antes de iniciar o primeiro quadro e depois de terminar o último quadro do fluxo de áudio.

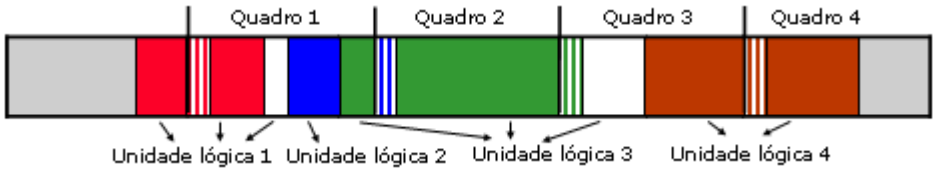


Figura 18 - Representação dos quadros que compõem um fluxo de áudio com bits de PAD no quadro 1 e 3.

O cabeçalho de um quadro sempre inicia com uma mesma seqüência de bits, chamada de *palavra de sincronismo*, para ajudar a manter o sincronismo. O primeiro quadro do fluxo inicia quando a primeira palavra de sincronismo for encontrada alinhada em bytes (ou seja, o primeiro bit da palavra de sincronismo deve ocorrer no primeiro bit de um byte do fluxo).

Um quadro sempre dispõe de informações suficientes para localizar sua unidade lógica de dados. Além disso, o tamanho do reservatório de bits não pode crescer indefinidamente, existindo um número máximo de bits que um quadro pode utilizar de quadros anteriores. Esse tamanho máximo varia entre diferentes formatos de áudio e depende do modo de localização da unidade lógica de dados do quadro e de restrições de codificação ou decodificação do formato.

Por fim, é válido destacar que a única base temporal do fluxo de mídia é a unidade lógica de dados. Como mencionado, cada quadro codifica um conjunto de amostras que representa um curto intervalo de tempo do sinal original. Tipicamente, esse intervalo é da ordem de poucas dezenas de milisegundos. Os bits de uma unidade lógica não necessariamente representam ordenadamente as amostras do sinal original no tempo, uma vez que as amostras originais são submetidas a várias transformações durante o processo de codificação.

3.3. Mecanismo de Ajuste Elástico de Áudio com Compressão

Considerando os requisitos de realizar o ajuste em tempo de exibição, alta fidelidade, pequena amplitude de variação do fator de escala e os formatos de áudio comprimidos estudados, a classe de algoritmos que funciona no domínio do tempo foi escolhida. Dentre os algoritmos dessa categoria, apenas o *Ajuste Regular* (ver Subseção 2.1.2) não realiza algum processamento envolvendo as

amostras do sinal. Esse foi o algoritmo escolhido como base para ser aplicado diretamente no formato de áudio comprimido porque os algoritmos mais complexos aumentariam muito o custo computacional e não atenderiam os requisitos de eficiência. Além disso, uma vez que esse trabalho foca em ajustes de pequena faixa, poucos defeitos são introduzidos pelo fato do algoritmo de ajuste ser simples e a qualidade do áudio resultante é aceitável, como apresentado nos testes de qualidade na Seção 7.1.

O algoritmo de ajuste é aplicado percorrendo o fluxo de áudio quadro a quadro e selecionando quadros a intervalos aproximadamente regulares para efetuar processamento. Processar quadros significa removê-los ou duplicá-los, caso se deseje, respectivamente, acelerar ou retardar a exibição de uma mídia. Considerando a modelagem do fluxo de áudio anteriormente descrita, a unidade de ajuste adotada no algoritmo de *Ajuste Regular* deve ser a unidade lógica de dados de um quadro.

Remover um quadro significa retirá-lo do fluxo de áudio, juntamente com sua unidade lógica de dados. No entanto, os bits do campo de dados do quadro removido que armazenam unidades lógicas de quadros posteriores devem permanecer no fluxo de dados, se espalhando pelos quadros anteriores não removidos. A Figura 19 ilustra o mesmo fluxo da Figura 17 após a simples remoção do quadro *3*. Nesse estágio, parte da unidade lógica do quadro *4* foi perdida. Para que isso não ocorra, os bits que estavam no campo de dados do quadro *3* que armazenavam amostras do quadro *4* precisam ser copiados para o quadro *2*, resultando no que está ilustrado na Figura 20. Como apresentado nas figuras, após a realização de operações de processamento de quadros, bits adicionais de PAD podem ter de ser inseridos no fluxo de dados.

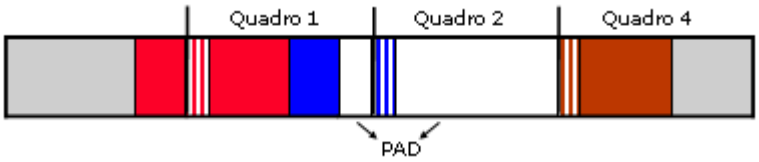


Figura 19 - Representação dos quadros no primeiro passo para remover o quadro 3.

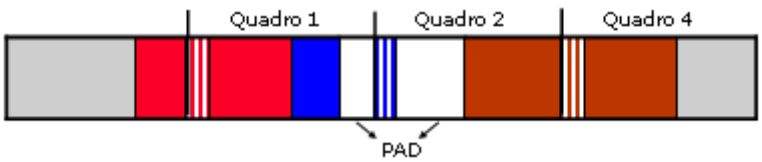


Figura 20 - Representação dos quadros após remoção do quadro 3.

De modo análogo, duplicar um quadro significa criar um novo quadro no fluxo de áudio e, possivelmente, modificar bits de quadros anteriores ao inserido no fluxo, com objetivo de inserir a unidade lógica de dados duplicada do quadro processado. A Figura 21 ilustra o fluxo de áudio após a simples duplicação do quadro *2* do fluxo de dados apresentado na Figura 17. Nesse estágio, os dados da unidade lógica do quadro *2* não foram duplicados. Para que isso ocorra, os bits dos quadros *1* e *2* precisam ser modificados, conforme ilustra a Figura 22.

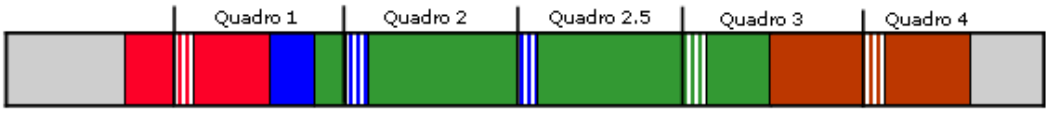


Figura 21 - Representação dos quadros no primeiro passo para duplicar o quadro 2.

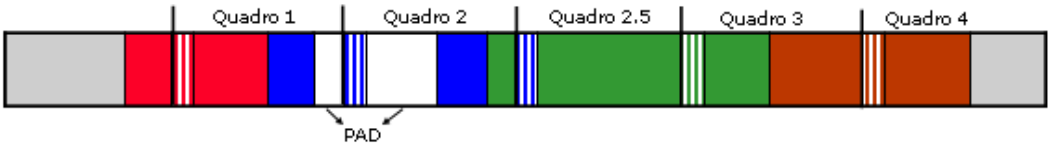


Figura 22 - Representação dos quadros após duplicação do quadro 2.

Nem todos os quadros têm um processamento fácil. No algoritmo proposto, a primeira decisão tomada foi considerar a diferença entre o tamanho do campo de dados e da unidade lógica de dados de um quadro durante o processamento. Assim, *a princípio*, nem todos os quadros podem ser processados (removidos ou duplicados). As regras consideradas foram as seguintes:

1. Somente os quadros que possuem o tamanho da unidade lógica de dados associada maior do que o tamanho do campo de dados podem ser removidos. Essa decisão evita a falta de espaço para dados de unidades lógicas de quadros posteriores.

2. Analogamente, apenas os quadros que possuem o tamanho da unidade lógica de dados associada menor do que o tamanho do campo de dados podem ser duplicados. Essa decisão evita que falte espaço para a unidade de dados lógica duplicada.

Os exemplos anteriores de remoção e duplicação respeitam essas regras. Na Figura 17, a unidade lógica do quadro *3* ocupa todo o campo físico do quadro *2* e parte dos campos físicos dos quadros *1* e *3*. Por isso, a remoção do quadro *3* pode ocorrer sem problemas, segundo a regra *1* acima. Por essa regra, os quadros *1* e *2* não podem ser removidos. De modo análogo, a unidade lógica do quadro *2* ocupa

apenas parte do campo físico do quadro *1*. O quadro *2* pode, então, ser duplicado segundo a regra *2* acima. Segundo a regra *2*, os quadros *3* e *4* não podem ser duplicados.

Sempre que ocorre uma operação de processamento respeitando as regras *1* e *2*, bits de PAD devem ser inseridos no espaço que restou no fluxo de áudio. Os bits de PAD podem ser transferidos entre os quadros do fluxo utilizando o seguinte algoritmo:

- Calcular o número *X* de bits do PAD que podem ser transferidos de um quadro para o próximo considerando que o tamanho máximo que o reservatório de bits pode atingir não pode ser ultrapassado.
- Deslocar em *X* bits a unidade lógica de dados do próximo quadro para ocupar parte do espaço atual do PAD e inserir novos bits de PAD ao final da unidade lógica transferida.

A Figura 23 ilustra o fluxo de áudio da Figura 22 após a transferência do PAD do final da unidade lógica do quadro *1* para o final da unidade lógica do quadro *4* (depois de ter passado pelos quadros *2*, *2.5* e *3*). A presença do PAD no final da unidade lógica de um quadro nunca invalida o fluxo, pois tais bytes são interpretados como uma continuação dos dados auxiliares do quadro.

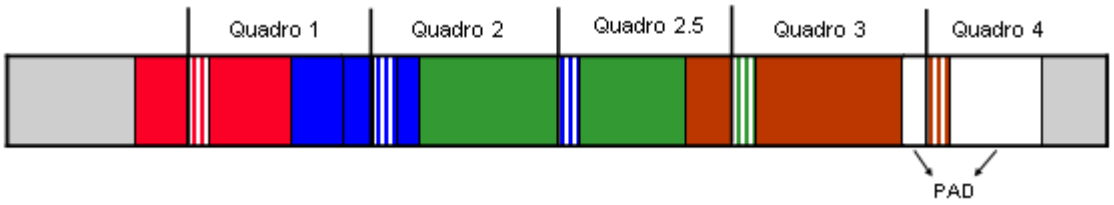


Figura 23 - Representação dos quadros após transferência do PAD para quadro 4.

Desse modo, os bytes que formam o enchimento podem ser utilizados para refinar o algoritmo de ajuste elástico, anteriormente proposto, com as seguintes regras:

3. Quadros que não atendem à regra de remoção definida no item *1* podem ser retirados do fluxo, desde que o tamanho da unidade lógica de dados associada, somada aos enchimentos do quadro, seja maior do que o tamanho do campo de dados.

4. Quadros que não atendem à regra de duplicação definida no item *2* podem ser re-inseridos no fluxo se o tamanho da unidade lógica de dados associada for menor do que o tamanho do campo de dados, somado a enchimentos de quadros anteriores.

Com a utilização das novas regras, mais quadros podem ser processados. Por exemplo, seria possível duplicar o quadro *4* no fluxo de áudio da Figura 23. Para fazer uso de tais regras, um PAD está sendo constantemente transferido entre os quadros do fluxo. O número de bits do PAD aumenta quando um quadro que atende às regras *1* e *2* é processado e diminui quando um quadro que não atende a essas regras, mas satisfaz as regras *3* e *4*, é processado ou quando não é possível transferir o PAD completo para o quadro seguinte por esse ter atingido o tamanho máximo do reservatório de bits.

É válido destacar que o tamanho dos quadros não é necessariamente constante. O algoritmo extrator de quadros é responsável por reconhecer e criar quadros utilizando as seguintes regras:

a) Reconhecer e criar o primeiro quadro analisando os bits iniciais do fluxo byte a byte até encontrar a palavra de sincronismo.

b) Verificar se imediatamente depois do final do quadro reconhecido existe outra palavra de sincronismo.

- Se sim, reconhecer novo quadro e voltar ao passo (b).
- Se não, indicar que fluxo de quadros terminou.

Durante o processamento do áudio, cada quadro é analisado para verificar se deve ou não ser processado. Essa decisão é responsabilidade do algoritmo de seleção de quadros a processar, descrito a seguir:

a) Calcular o fator de ajuste efetivamente aplicado ao fluxo de áudio até o momento considerando o número de quadros já processados dividido pelo número de quadros já analisados.

b) Comparar o resultado do fator de ajuste efetivamente aplicado com o fator originalmente solicitado para decidir se o quadro em análise deve ou não ser processado.

- Se sim, verificar se o quadro pode ser processado considerando os tamanhos do PAD, do campo físico e da unidade lógica do quadro segundo as regras 1, 2, 3 e 4 citadas.

Caso ocorra uma solicitação de mudança no fator de ajuste, o algoritmo de seleção de quadros recalcula o fator de ajuste efetivo do momento da solicitação em diante, desconsiderando os ajustes anteriores.

Uma vez que nem todos os quadros podem ser processados, o conjunto de quadros escolhidos pelo algoritmo de seleção de quadros a processar não é regularmente distribuído. Uma medida de linearidade do algoritmo será discutida na Seção 7.1.

A não-linearidade do algoritmo aliada ao fato da granularidade da unidade de ajuste elástico ser discreta implica que os novos valores das âncoras (lembrando que uma âncora marca um instante de tempo em um fluxo audiovisual) existentes sobre o fluxo de áudio não podem ser simplesmente estimados através de uma relação linear do fator de ajuste aplicado (ver Figura 24). Em outras palavras, considerando que existia uma âncora no fluxo de áudio no tempo a_1 e que foi aplicado um fator de ajuste f, não é válido supor que o novo valor da âncora após o ajuste será $f*a_1$.

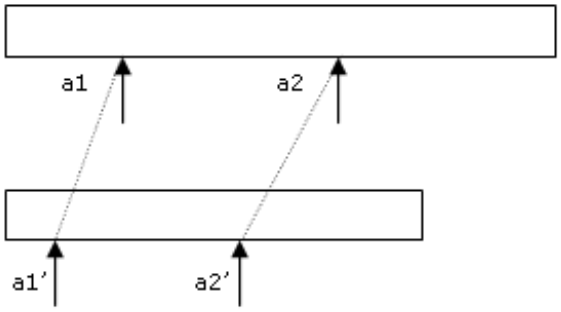


Figura 24 – Reposicionamento de âncoras em um fluxo de mídia devido ao ajuste.

Por essa razão, o mecanismo de ajuste elástico é capaz de indicar quais os novos valores de determinados instantes de tempo do fluxo original da mídia após o processamento. Isso pode ser feito depois da realização do ajuste ou durante sua execução. No primeiro caso, o programa que solicitou previamente a realização do ajuste pode perguntar qual os novos tempos dos instantes desejados. Quando realizado em tempo de execução, a chamada da realização do ajuste deve indicar quais instantes deseja monitorar. Nesse caso, o próprio algoritmo de ajuste

elástico dispara eventos ao encontrar o novo valor de uma âncora desejada. Sendo assim, o programa que solicita o ajuste pode ser avisado em tempo de exibição qual o novo valor de apresentação da âncora.

Por fim, cabe ressaltar que o algoritmo de ajuste proposto preserva bits do fluxo de áudio existentes antes de iniciar o primeiro quadro e depois de terminar o último quadro, bem como dados de enchimento dos quadros presentes no fluxo de áudio original. A preservação desses bits é muito importante uma vez que esses podem representar metadados importantes.

3.4. Ajuste Elástico em Fluxos de Áudio Suportados

O mecanismo de ajuste da seção anterior foi aplicado a alguns formatos de áudio comprimido. As subseções a seguir detalham as particularidades do mecanismo de ajuste proposto para MPEG-1 áudio, MPEG-2 áudio BC, MPEG-2 AAC, MPEG-4 AAC e AC-3.

3.4.1. MPEG-1 áudio

O padrão MPEG-1 áudio (ISO, 1993) define três camadas de codificação de áudio, denominadas MP1, MP2 e MP3. Quanto maior o número da camada, mais complexa é a sua codificação, sua sintaxe dos dados gerados e sua decodificação. Todas as camadas podem codificar áudio com um ou dois canais e taxa de amostragem de *32*, *44.1* ou *48*kHz. A Tabela 1 ilustra uma comparação entre as 3 camadas acima citadas em relação à taxa de bits por canal e taxa de compressão atingida.

Camadas	**Taxa de bits (kbp/s)**	**Taxa de compressão**
MP1	32 a 448	4:1
MP2	32 a 384	6:1 - 8:1
MP3	32 a 320	10:1 - 12:1

Tabela 1 - Comparação entre camadas do MPEG-1 - fonte: (Noll, 2000; Soares, 2005).

O formato de dados de todas as camadas é dividido em quadros. O tamanho de um quadro é determinado a partir de uma fórmula que relaciona a taxa de bits e

a de amostragem[5] que, na maioria dos casos, não se modifica dentro de um mesmo fluxo de áudio. As fórmulas a seguir calculam o tamanho de um quadro em fluxos MP3, MP2 e MP1.

$$Tamanho\_Quadro = 144 * \frac{taxa\_de\_bits}{taxa\_de\_amostragem} + (byte\_complementar)$$

Figura 25 - Cálculo do tamanho de um quadro MP2 ou MP3.

$$Tamanho\_Quadro = 12 * \frac{taxa\_de\_bits}{taxa\_de\_amostragem} + (byte\_complementar)$$

Figura 26 - Cálculo do tamanho de um quadro MP1.

Nas fórmulas acima, as taxas de bits e de amostragem podem assumir os valores já mencionados. A unidade a considerar da taxa de bits é de kbp/s e da taxa de amostragem é kHz. Se a divisão da taxa de bits pela taxa de amostragem não resultar em um número inteiro, o resultado é truncado e, nessas situações, o codificador irá adicionar um byte complementar em alguns quadros. Em todos os demais casos, o valor do byte complementar é sempre zero. Vale mencionar que caso a taxa de bits mude no decorrer do fluxo, indicando a mudança do tamanho dos quadros, diz-se que o fluxo é VBR (*variable bit rate*), mas muitos decodificadores não são capazes de manipular fluxos desse tipo.

O MPEG-1 não padroniza bits que possam existir antes ou depois dos quadros do fluxo, embora seja comum que alguns codificadores armazenem metadados nesses bits. Por exemplo, existe um padrão informal[6] chamado *ID3 tag* (Nilsson, 2003) que se aplica a arquivos que armazenam música em formato MP3. O padrão é bastante utilizado e convenciona utilizar os últimos *128* bytes do arquivo para adicionar informações relacionadas à música como: nome, artista, álbum, ano de lançamento, gênero, comentário e outros dados definidos pelo usuário.

Um quadro MPEG-1 áudio é composto pelos seguintes campos:

---

[5] Existe a possibilidade de um campo do cabeçalho desses formatos indicar que o tamanho do quadro não é determinável *a priori*, mas somente durante a interpretação individual dos campos do quadro. Esse formato é conhecido como livre e não foi tratado por este trabalho.

[6] Padrão informal quer dizer uma especificação elaborada por alguém, mas não regulamentada por nenhum órgão de padronização (como ISO e ITU).

- *Cabeçalho*: inicia com a palavra de sincronismo "*1111 1111 1111*" e contém dados de descrição do quadro, como presença ou não de CRC, taxa de bits e taxa de amostragem.
- *CRC*: campo opcional para detectar erros em partes críticas do quadro.
- *Dados de áudio*: contém todas as informações relativas às amostras de áudio desse quadro.
- *PAD*: campo opcional para bits de enchimento.

Ao empregar o algoritmo da seção anterior, os campos *cabeçalho* e *CRC* foram mapeados para o campo de *cabeçalho* do formato genérico e os campos *dados de áudio* e *PAD* foram mapeados para a *unidade lógica de dados*.

Um quadro MP1 contém informações de *384* amostras de áudio e quadros MP2 e MP3 contêm informações de *1152*. O tempo necessário para exibir as amostras associadas a um quadro de fluxos MPEG-1 é calculado pela divisão do número de amostras no quadro pela taxa de amostragem. A Tabela 2 ilustra esses valores em milisegundos.

Camada	**Taxa de amostragem em kHz**		
	32	**44,01**	**48**
I	12	8,73	8
II	36	26,12	24
III	36	26,12	24

Tabela 2 - Duração para exibir as amostras associadas a um quadro (em milisegundos).

Em fluxos MP1 e MP2 não existe reservatório de bits e, conseqüentemente, o campo físico é igual à unidade lógica de dados. Desse modo, qualquer quadro do fluxo pode ser processado pelo mecanismo de ajuste e nunca são inseridos ou transferidos bytes de PAD. Sendo assim, o mecanismo de ajuste proposto pode ser utilizado supondo que o número de bytes emprestados é sempre zero.

Em fluxos MP3 existe o conceito de reservatório de bits e o mecanismo de ajuste proposto pode ser utilizado em sua plenitude. Cada quadro possui um campo de *9* bits para armazenar o número de bytes emprestados de quadros anteriores (chamado *main_data_begin*). Sendo assim, um quadro MPEG-1 MP3 pode depender de, no máximo, *5* quadros anteriores.

O processamento de quadros MP3 durante a realização do ajuste deve alterar o valor do campo *main_data_begin* para alterar o tamanho do reservatório

de bits dos quadros. Como conseqüência, o campo CRC (caso exista) precisa ser recalculado.

O algoritmo de ajuste poderia aproveitar os bits de preenchimento do final da unidade lógica dos quadros para aumentar o número de bits do PAD e facilitar o processamento de quadros. No entanto, esse trabalho optou por preservar esses bits porque eles podem representar metadados importantes para alguma aplicação, como já mencionado anteriormente.

3.4.2. MPEG-2 áudio BC

O MPEG-2 áudio BC (*Backward Compatible*) (ISO, 1998) também possui três camadas de codificação que geram fluxos MP1, MP2 e MP3. Esse formato define dois novos padrões de áudio: LSF (*Low Sampling Frequencies*) e MC (*Multi Channel*).

O MPEG-2 áudio LSF define um formato de dados bastante semelhante ao MPEG-1 áudio. Algumas diferenças entre esses formatos relevantes para o contexto deste trabalho podem ser enumeradas.

1. As taxas de amostragem suportadas pelo novo padrão são *16*, *22.05* ou *24*kHz.
2. As taxas de bits também estão mais baixas. De 32 a 256kbit/s para fluxos MP1 e de 8 a 160kbit/s para fluxos MP2 e MP3.
3. O número de amostras associadas a um quadro no MP2 é *576*.
4. Alguns campos do formato foram modificados, por exemplo, o *main_data_begin* passou a ter *8* bits. Devido a essas mudanças, um quadro MP3 pode depender de até *10* quadros anteriores considerando o tamanho mínimo de um quadro e o tamanho máximo do reservatório de bits.

O mecanismo de ajuste elástico no MPEG-2 BC LSF é realizado de modo similar ao que foi descrito para o MPEG-1, não trazendo nenhuma informação adicional que mereça destaque.

O MPEG-2 áudio MC define um formato compatível com o MPEG-1 áudio e acrescenta, opcionalmente, mais *4* canais de *surround*, para possibilitar configuração 5.1, e até *7* canais com qualidade de voz. Um decodificador MPEG-

2 áudio MC é capaz de reproduzir áudio codificado como MPEG-1 áudio do mesmo modo que um decodificador MPEG-1 áudio. Porém, um decodificador MPEG-1 áudio reproduz áudio codificado como MPEG-2 áudio MC desconsiderando as extensões do formato de dados acrescentadas pelo novo padrão, ou seja, desconsiderando os novos canais. Isso ocorre porque os dados de tais canais são adicionados no campo de PAD dos quadros. A Figura 27 ilustra um codificador e decodificador MPEG-2 áudio MC, destacando a semelhança com codificadores e decodificadores MPEG-1.

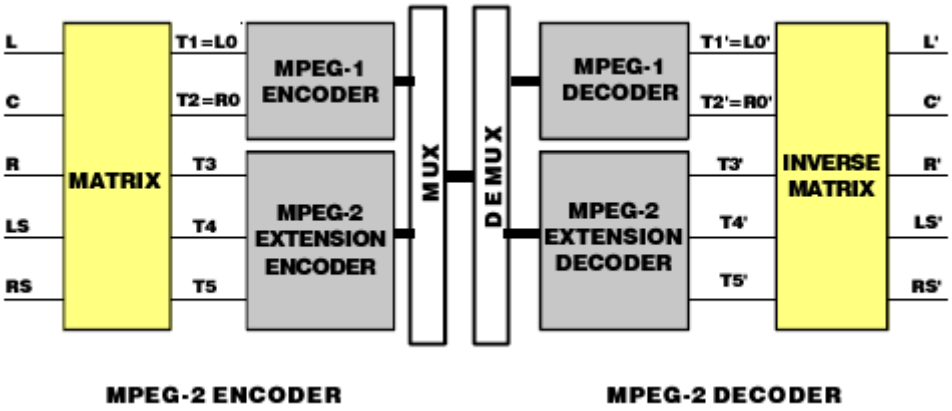


Figura 27 - Codificador e decodificador MPEG-1 e MPEG-2 áudio MC - fonte: (Noll, 1999).

O mecanismo de ajuste elástico no MPEG-2 BC MC também é realizado de modo similar ao que foi descrito para o MPEG-1. Vale comentar apenas que os canais de *surround* estão codificados nos quadros do sinal de áudio da mesma maneira que os outros canais. Portanto, o descarte e duplicação de quadros do algoritmo de ajuste também acontece com os canais de *surround*, sem modificar o algoritmo proposto.

3.4.3. MPEG-2 AAC

O AAC (*Advanced Audio Coding*) (ISO, 1997; ISO, 2004) é o padrão de áudio definido pelo MPEG-2 para aplicações que não precisam manter a compatibilidade com o formato MPEG-1 áudio.

Esse padrão suporta taxa de amostragem de *8* a *96*kHz e taxa de bits varia de *48* até *576*kbits/s por canal. Utilizando o AAC, é possível obter uma taxa de compressão de *16:1* mantendo a qualidade de um áudio de CD. O AAC pode

suportar até *48* canais, sendo, portanto, possível configurar o som em *5.1*, *7.1* ou mesmo *10.2* canais.

Dada a variedade grande de configurações possíveis para número de canais e taxas de amostragem e de bits, o AAC define um conjunto de perfis de configuração padrão. São eles:

- *Low-complexity* (LC): utiliza um conjunto de ferramentas de codificação para possibilitar a codificação e, principalmente, a decodificação com pequeno custo computacional.
- *Main*: inclui o perfil LC e adiciona novas ferramentas de codificação com objetivo de aumentar a taxa de compressão com o custo de aumentar a complexidade computacional.
- *Scalable Sampling Rate* (SSR): codifica fluxo em camadas complementares de modo que decodificadores diferentes possam extrair somente as camadas necessárias. A qualidade do fluxo obtido pode ser menor do que a do perfil *LC*, chegando até a do perfil *Main*.

O padrão AAC define que as amostras codificadas são dividas em blocos, cada qual com informações de *1024* amostras. Entretanto, como não existe um campo que determina o tamanho dos blocos, estes podem ser delimitados somente ao final da leitura e interpretação completa de seus bits.

Para permitir a correta identificação dos blocos, o padrão AAC sugere dois formatos de dados para protocolos de transporte. O formato ADIF (*Audio Data Interchange Format*) contém um cabeçalho inicial seguido por blocos. Desse modo, esse formato é adequado apenas para situações em que o fluxo será lido do inicio ao fim, e onde a possibilidade do decodificador perder o sincronismo é bastante remota, como a leitura de um arquivo armazenado localmente.

O formato ADTS (*Audio Data Transport Stream*) possui sintaxe similar ao MPEG áudio BC, sendo, por isso, conhecido como MP4. Um quadro ADTS é composto pelos seguintes campos:

- *Cabeçalho de tamanho fixo*: inicia com a palavra de sincronismo "*1111 1111 1111*" e contém dados de descrição do quadro, como presença ou não de CRC e taxa de amostragem.
- *Cabeçalho de tamanho variável*: contém informações de *copyright* (codificadas nos campos *copyright_identification_bit* e

copyright_identification_start), o tamanho do quadro (campo *frame_length*), o tamanho do reservatório de bits (campo *adts_buffer_length*) e o número de blocos no quadro (campo *number_of_raw_data_blocks_in_frame*).

- *CRC*: campo opcional para detectar erros em partes críticas do quadro.
- *Blocos de áudio*: contém um conjunto de blocos AAC.
- *PAD*: campo opcional com bits de enchimento.

Nesse formato, cada quadro pode conter informações de até *4* blocos (campo *number_of_raw_data_blocks_in_frame*), o que equivale a dizer que cada quadro possui informações de até *4*1024* amostras. A vantagem do codificador adicionar vários blocos em um quadro é diminuir o *overhead* de codificar os campos de cabeçalho e CRC no início de um quadro. Entretanto, é válido destacar que sempre é possível codificar um fluxo AAC utilizando quadros com um único bloco e que em todas as codificações pesquisadas durante o decorrer deste trabalho, essa foi a opção de compressão utilizada pelos codificadores.

Os quadros ADTS podem conter reservatório de bits cujo tamanho é indicado pelo campo *adts_buffer_fullness*. O tamanho máximo do reservatório de bits depende do número de canais do fluxo e da taxa de bits média.

O mecanismo de ajuste elástico proposto foi aplicado ao formato MPEG-2 AAC ADTS. Os campos *cabeçalho de tamanho fixo*, *cabeçalho de tamanho variável* e *CRC* do quadro ADTS foram mapeados para o campo de *cabeçalho* do formato genérico e os demais campos foram mapeados para a *unidade lógica de dados*.

O ADIF e ADTS são apenas exemplos de protocolo de transporte. Outros formatos podem ser utilizados para encapsular os blocos de dados AAC, como, por exemplo, o protocolo do MPEG-4 Sistemas (ISO, 2001a).

É importante ainda ressaltar que um novo problema surge nas regras de seleção e processamento de quadros em fluxos do perfil principal do AAC, pois existe uma ferramenta de codificação que permite que o codificador utilize valores de amostras de blocos anteriores para prever os valores de amostras do bloco atual. No pior caso, pode demorar até 240 quadros para o codificador gerar um quadro totalmente independente dos anteriores (Purnhagen, 2004). O algoritmo

proposto não funciona com tamanha dependência entre quadros, ficando a generalização do algoritmo para cobrir este caso proposta como trabalho futuro.

3.4.4. MPEG-4 AAC

O padrão MPEG-4 áudio (ISO, 2001b) codifica sinais de voz humana e áudio multicanal com alta qualidade e também oferece suporte a áudios naturais e sintéticos, como explicado na Subseção 2.2.8. Para os sinais naturais de áudio, existe suporte à codificação entre *2*kbit/s até *64*kbit/s, existindo *3* tipos de codificação, como ilustrado na Figura 28.

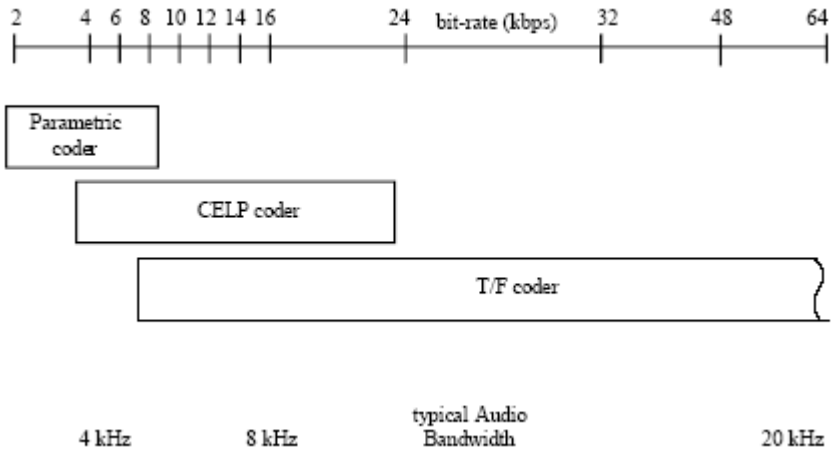


Figura 28 - Tipos de codificação para diferentes taxas de bits em MPEG-4 natural - fonte: (ISO, 2001b).

O codificador MPEG 4 utiliza técnicas paramétricas para sinais com taxas de bits entre *2* e *8*kbit/s. Essa técnica é utilizada principalmente para voz humana com taxa de amostragem *8*kHz. Para taxas de *4* a *24*kbit/s, o codificador utiliza técnicas CELP (*Coding Excited Linear Predictive*). Essa codificação pode ser utilizada para voz ou outros tipos de áudio com taxas de amostragem de *8*kHz e *16*kHz. Para taxas de bits mais altas (em geral, acima de *16*kbps) e taxas de amostragem a partir de *8*kHz, são utilizadas técnicas que se baseiam no MPEG-2 AAC.

O formato de dados do fluxo MPEG-4 AAC é similar ao MPEG-2 AAC, embora existam várias ferramentas de codificação novas. O MPEG-4 pode ser transmitido por diferentes protocolos de transporte, assim como o MPEG-2. O

mecanismo de ajuste elástico foi aplicado apenas aos fluxos MPEG-4 AAC ADTS.

3.4.5. AC-3

O *AC-3* (*Audio Code-3*) (ATSC, 1995) é um formato de áudio proprietário da *Dolby Laboratories Inc* conhecido também pelos seguintes nomes: Dolby Digital, DD e ATSC A/52.

O formato permite codificar de *1* a *5* canais de *20*Hz até *20*kHz e um canal de baixas freqüências de *20*Hz até *120*Hz. *Dolby Digital* utiliza taxas de dados de *32*kbps até *640*kbps, atingindo uma taxa de compressão de aproximadamente *13:1*. As taxas de amostragem possíveis são *32*, *44.1* e *48*kHz.

Um fluxo AC-3 é composto por uma seqüência de quadros. Um quadro AC-3, ilustrado na Figura 29, é formado pelos seguintes campos:

- *Synchronization Information* (SI): inicia com a palavra de sincronismo "*0000 1011 0111 0111*" e contém dados de descrição do quadro, como taxa de amostragem e tamanho do quadro.
- *Bit Stream Information* (BSI): contém parâmetros da codificação das amostras do quadro.
- *Audio blocks* (AB): Seis blocos de áudio, cada um contendo *256* amostras de áudio. Os blocos não são independentes.
- *Auxiliary data* (Aux): campo opcional que contém bits de PAD.
- *CRC*: campo obrigatório para detectar erros.

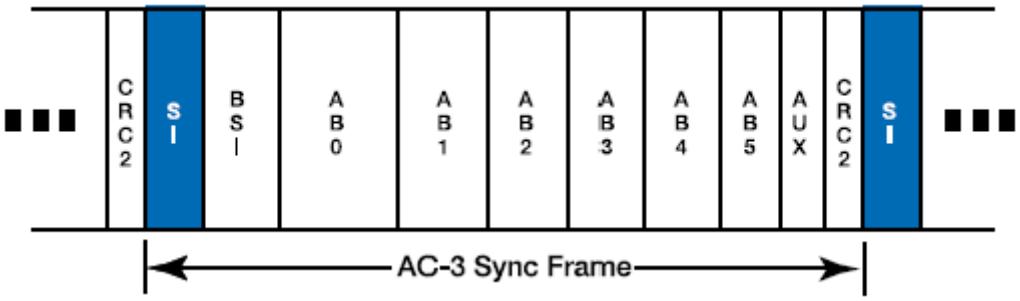


Figura 29 - Quadro do formato AC-3.

O campo *SI* foi mapeado para o campo de *cabeçalho* do formato genérico e os demais campos foram mapeados para a *unidade lógica de dados*.

Contando as amostras dos *6* blocos, cada quadro possui *1536* amostras. Isso significa que o tempo necessário para exibição de um quadro varia entre *32* e *48*ms.

Em um fluxo AC-3 não existe reservatório de bits nem qualquer dependência entre quadros. Conseqüentemente, o mecanismo de ajuste pode ser utilizado de modo similar ao usado no MP2.

3.5. Ajuste Elástico em Fluxos de Sistemas

Alguns formatos de mídia agrupam vários fluxos elementares juntamente com metadados no que se chama de *fluxo de sistemas*. Exemplos de fluxos elementares são fluxos de áudio, vídeo e dados; exemplos de metadados são marcas para manter o sincronismo temporal entre diferentes fluxos elementares.

Para realizar ajuste elástico em fluxos de sistemas pode ser necessário ajustar todos os fluxos elementares e recalcular alguns metadados. Para ajustar um fluxo elementar de áudio, o algoritmo proposto nas subseções anteriores pode ser plenamente utilizado. Para ajustar o vídeo, pode-se utilizar, por exemplo, o algoritmo proposto por Cavendish (Cavendish, 2005). O ajuste de fluxo de dados, quando necessário, é deixado como trabalho futuro.

O modelo do fluxo de sistemas proposto neste trabalho é composto por quadros chamados PACKETS. Um PACKET é formado por um cabeçalho seguido por um conjunto de bits de um fluxo elementar. No cabeçalho, podem existir marcas de tempo (*timestamp*) cujos valores foram atribuídos pelo codificador com base no valor de seu relógio, e outros campos dependentes do formato de mídia. O conjunto de bits de cada PACKET pode conter dados de um ou mais quadros de um fluxo elementar (comum em fluxos de áudio, que possuem quadros pequenos), podendo inclusive conter apenas parte dos dados de um quadro (caso comum para quadros grandes de vídeo). De modo análogo ao caso do fluxo de áudio, é válido destacar que o tamanho dos PACKETs não é necessariamente constante e seu cabeçalho sempre inicia com uma palavra de sincronismo.

As figuras abaixo ilustram exemplos de fluxos de sistemas, cada um com dois fluxos elementares, um de áudio e outro de vídeo. A Figura 30 contém PACKETs de áudio e vídeo uniformemente distribuídos. Entretanto, no caso

geral, existem vários PACKETs de vídeo para um PACKET de áudio, já que é comum que quadros de vídeo possuam muito mais bits do que quadros de áudio. A Figura 31 ilustra um fluxo típico de sistemas.

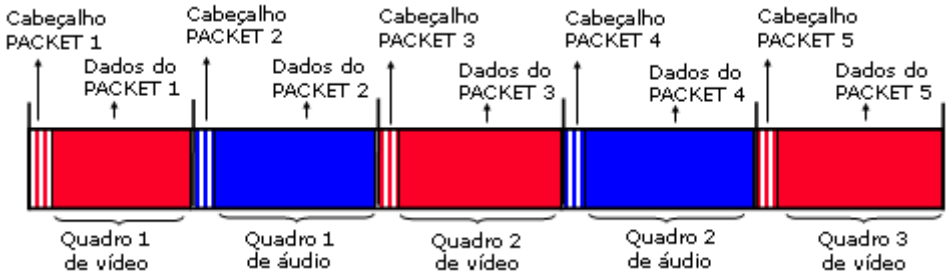


Figura 30 - Fluxo de sistemas com fluxos elementares de áudio e vídeo uniformemente distribuídos.

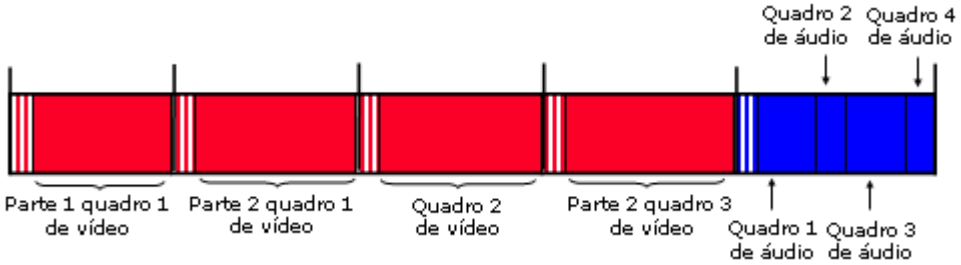


Figura 31 - Fluxo de sistemas com fluxos elementares de áudio e vídeo com distribuição não-uniforme.

Os metadados de sistemas, cabeçalhos dos PACKETS, podem ser tratados como novos atributos dos quadros dos fluxos elementares e devem ser atualizados por um algoritmo de resincronização, que é constituído pelos algoritmos extrator de estruturas de sistemas, de demultiplexação, de conversão de formatos, de controle de fator de ajuste, de verificação do sincronismo intermídia, de multiplexação e de serialização, todos apresentados na Figura 32 e detalhados no decorrer desta seção.

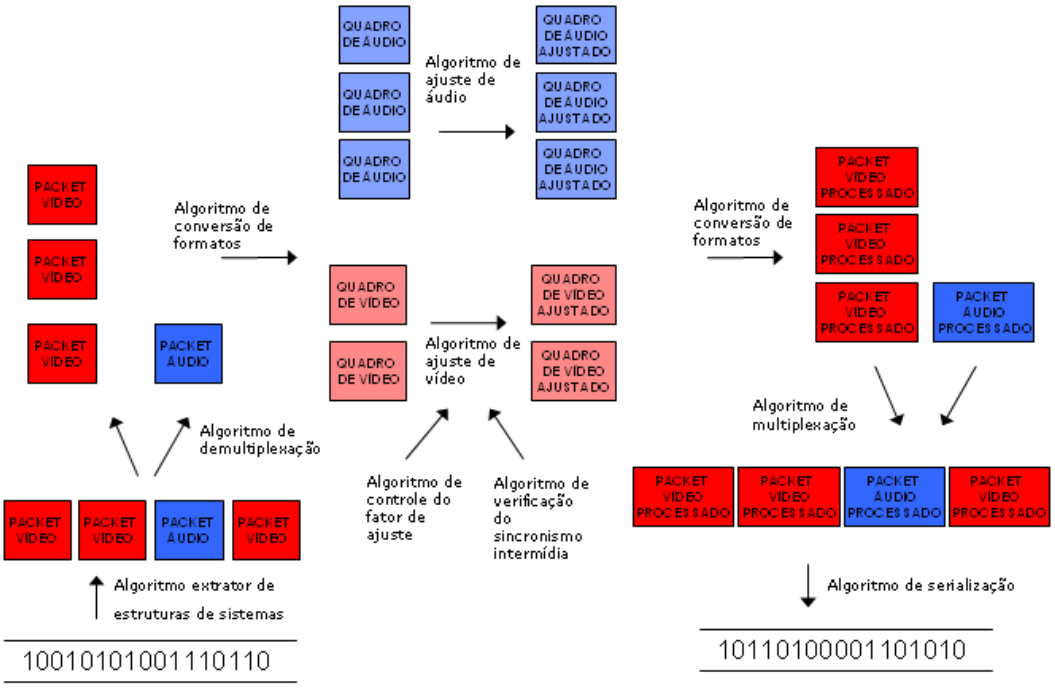


Figura 32 – Conjunto de algoritmos para realizar ajuste elástico em fluxos de sistemas.

O algoritmo extrator de estruturas de sistemas é responsável por reconhecer e criar os PACKETs. O primeiro PACKET do fluxo inicia quando a primeira palavra de sincronismo for encontrada, alinhada em bytes[7]. A partir de então, os bits seguintes ao final desse PACKET podem ser uma nova palavra de sincronismo, caso outro PACKET seja encontrado, ou não, quando o fluxo de sistemas termina. Durante a criação, os PACKETs reconhecidos são numerados com o objetivo de manter a informação da ordem em que estavam no fluxo original.

O próximo passo é chamado de algoritmo de demultiplexação. É necessário separar os PACKETs de cada fluxo para então aplicar o algoritmo de ajuste em cada fluxo elementar. A separação é baseada no fato de cada PACKET identificar univocamente um fluxo elementar.

Depois da demultiplexação, é necessário extrair os dados dos PACKETs para montar os quadros de cada fluxo elementar, sendo esses os dados esperados pelo algoritmo de ajuste do fluxo elementar. Para transformar o modelo de fluxo de sistema no novo modelo de fluxo de áudio - e no de vídeo proposto por

[7] É preciso que exista um tratamento de erros para verificar que a palavra de sincronismo encontrada é válida, ou seja, não faz parte dos dados transportados.

Cavendish (Cavendish, 2005) - é necessário aplicar um algoritmo de conversão de formatos.

A Figura 33 ilustra como o algoritmo deve montar quadros do fluxo elementar a partir dos dados contidos nas estruturas de sistemas. O algoritmo de montagem de quadros de áudio já foi descrito na Seção 3.3. A novidade é que também é necessário criar os metadados, que são os cabeçalhos dos PACKETs do fluxo de sistemas. Entretanto, nem todos os quadros gerados dos fluxos elementares precisam ter metadados, apenas aqueles que contêm o primeiro byte de um PACKET (ver quadro *2* do fluxo elementar na Figura 33). É valido destacar que, após a realização de ajuste, é preciso realizar a conversão inversa, ou seja, montar as estruturas de sistemas a partir dos quadros do fluxo elementar.

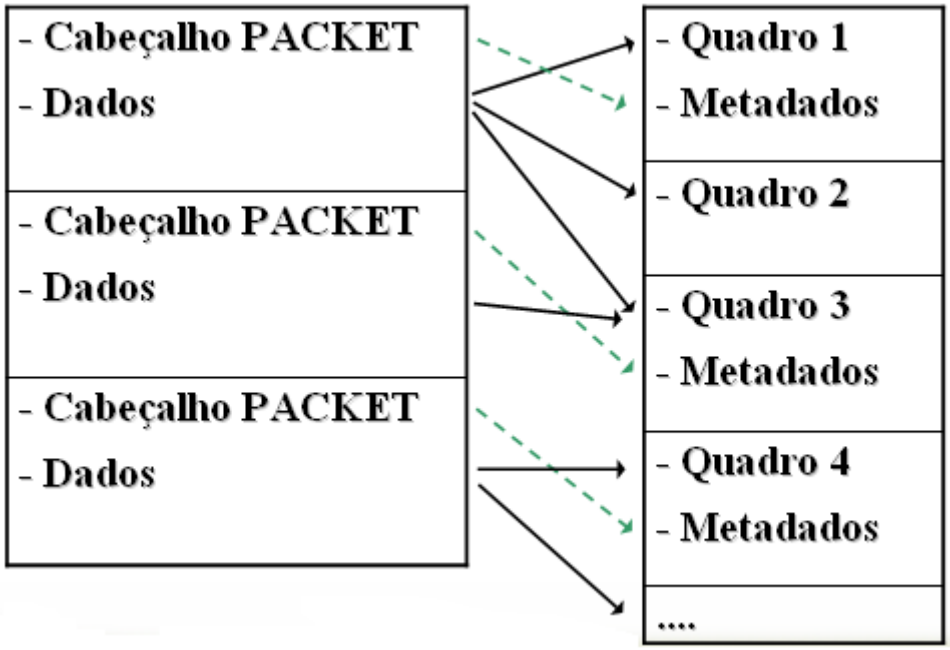


Figura 33 - Transformação de um fluxo de sistemas em um fluxo elementar para aplicar algoritmo de ajuste elástico.

Para o tratamento do fluxo de sistemas, algumas mudanças são necessárias nos algoritmos de ajuste dos fluxos elementares. Em primeiro lugar, a modelagem do fluxo elementar deve conter os metadados de sistemas, como já mencionado, o que afeta a modelagem do fluxo de áudio proposta na Seção 3.2. Cada quadro do fluxo elementar deve poder conter um conjunto de metadados. Além disso, também é necessário armazenar a posição em que os bits desses metadados precisam ser inseridos nos bits do quadro do fluxo elementar no momento da reconstrução dos quadros de sistemas.

Outras mudanças nos algoritmos de ajuste dos fluxos elementares com objetivo de manipular os metadados de sistemas são listadas abaixo:

- Durante a remoção de um quadro, pode ser necessário transferir os metadados contidos de um quadro para o seguinte;
- Durante a inserção de um quadro, pode ser necessário criar estruturas de sistemas (metadados) no novo quadro;
- Para todos os quadros do fluxo resultante é necessário recalcular marcas de tempo presentes nos metadados.

O recálculo das marcas de tempo merece ser mais detalhado. A inserção ou o descarte de quadros invalidam os cálculos realizados no codificador. O algoritmo de ajuste elástico precisa recalcular novos valores para as marcas de tempo do fluxo de mídia acrescentando ou diminuindo o valor da duração de quantos quadros já foram inseridos ou removidos, respectivamente. Somente com essa atualização, os valores das marcas de tempo continuam válidos e o sincronismo intra e intermídia pode ser mantido.

Um outro problema relacionado à transmissão de marcas de tempo é que deve existir um intervalo máximo de tempo entre o envio de duas amostras consecutivas da referência de relógio. Ao realizar ajuste elástico com fator $f>1$, o intervalo máximo, no entanto, pode ser ultrapassado, fazendo com que seja necessário determinar e inserir, no fluxo de sistemas, novas amostras de relógio, através de interpolação entre as existentes. Mais ainda, pode ser necessário criar novas estruturas de sistemas para transmitir as novas amostras de relógio.

A última mudança nos algoritmos de ajustes de fluxos elementares é necessária para viabilizar o algoritmo de controle do fator de ajuste, como explicado a seguir. No algoritmo de ajuste proposto, um fluxo de sistemas é processado por um pequeno trecho (o que equivale a um pequeno intervalo de tempo da mídia) e seguido por um algoritmo de controle de fator. Isso se repete de trecho em trecho até acabar o fluxo. Obviamente, no primeiro trecho o valor do fator de ajuste aplicado em cada fluxo elementar é igual ao fator originalmente aplicado no fluxo de sistemas.

Em cada trecho, os algoritmos de ajuste de fluxos elementares devem executar até acabarem os quadros de fluxos elementares criados a partir dos bits extraídos do fluxo de sistemas. Ao acabar a execução dos ajustes em todos os fluxos elementares presentes naquele trecho do fluxo de sistemas, o algoritmo de

controle de fator é executado. O algoritmo de controle deve analisar o fator obtido por cada fluxo elementar até o momento atual e decidir, sempre tentando aproximar o valor obtido do fator originalmente aplicado ao fluxo de sistemas, qual será o fator de ajuste a ser utilizado no próximo trecho em cada fluxo elementar. A Figura 34 ilustra um fluxo de sistemas sendo divido em fluxos elementares e a delimitação de trechos de processamento.

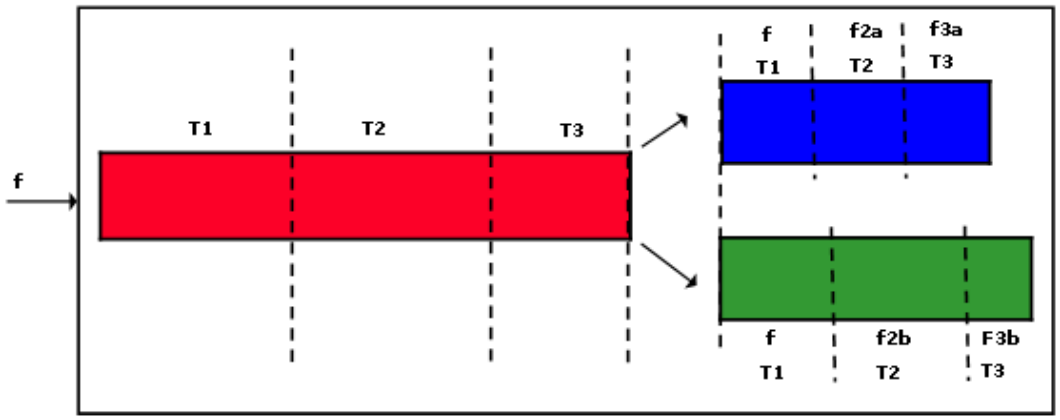


Figura 34 - Representação de divisão dos fluxos elementares em trechos para recálculo de fator de ajuste durante aplicação de ajuste elástico.

Sabendo-se que n é o número de trechos em que o ajuste já foi aplicado, f é o fator de ajuste originalmente aplicado ao fluxo de sistemas e f_{atual} é o fator de ajuste obtido pelo algoritmo de ajuste na parte do fluxo elementar já processada, a seguinte fórmula calcula o fator adequado para se aplicar no novo trecho de cada fluxo elementar com objetivo de convergir f_{atual} para f .

$$f_{i+1} = (n+1) \cdot f - n \cdot f_{atual}$$

Figura 35 - Cálculo do fator de ajuste a ser aplicado no trecho i+1 dos fluxos elementares componentes da mídia original.

Por exemplo, supondo que seja solicitado um ajuste com fator *0.9*, e que nos dois primeiros trechos de um fluxo elementar não tenha sido atingido o fator de *0.9*, mas apenas de *0.95*. Nesse caso, o valor do fator de ajuste calculado para o terceiro trecho desse fluxo elementar deve ser *3*0.9-2*0.95=0.8*. Utilizando um fator de *0.8* no terceiro trecho é possível compensar o erro de não ter atingido o fator *0.9* nos dois primeiros trechos.

O exemplo do parágrafo acima ilustra que, embora o algoritmo de ajuste tenha sido criado pensando em ajustar a mídia em até 10% pode ser necessário

ultrapassar esses valores em determinados trechos da mídia. Vale ainda mencionar que caso ocorra uma solicitação de mudança no fator de ajuste originalmente aplicado, deve-se recalcular o fator de ajuste efetivo do momento da solicitação em diante, desconsiderando os ajustes anteriores.

O algoritmo de controle de fator de ajuste acima, no entanto, não considera que a sincronização intermídia pode ser comprometida. Surge então a necessidade de verificar se o sincronismo intermídia está sendo mantido e, em caso negativo, tomar as providências necessárias. Um possível algoritmo é descrito a seguir:

a) Calcular o instante atingido por cada fluxo elementar até um determinado trecho;

b) Identificar qual o fluxo que está mais adiantado f_1 e qual está mais atrasado f_2 dentre todos os fluxos elementares. Calcular a diferença existente entre os instantes de f_1 e f_2 considerando que não existe ajuste (obtendo-se o valor *dmo*) e considerando que existe (obtendo-se o valor *dmp*). A seguir, calcular o seguinte valor: $dif=|dmo-dmp|$;

c) Verificar se o valor de *dif* é maior do que um determinado valor limite. Se sim, deve-se considerar que o fator aplicado ao fluxo de sistema é o fator do fluxo elementar que está mais atrasado (caso o $f<1$) ou adiantado (caso contrário).

O valor limite mencionado no passo (c) é estabelecido dependendo do quanto se deseja manter o sincronismo intermídia. Segundo Youssef (Aly & Youssef, 2002), diferenças superiores a *160*ms já são perceptíveis aos usuários. Se a condição do passo (c) for verdadeira em algum instante significa que o ajuste elástico não será realizado com o fator originalmente aplicado, mas sim com um fator cujo valor é um número entre o fator original e *1*. Vale ainda observar que é importante que os trechos de processamento sejam pequenos de modo que o dessincronismo intermídia seja pequeno.

Depois de realizar os ajustes nos fluxos elementares e de recriar as estruturas de sistemas, é necessário multiplexar os PACKETs de diferentes fluxos elementares com objetivo de recriar o fluxo de sistemas. A ordem em que os PACKETs devem ser multiplexados obedece ao número de ordem definido durante a criação da estrutura. Por fim, o algoritmo de serialização transforma os PACKETS nos dados binários do fluxo de sistemas ajustado.

3.5.1.
Ajuste Elástico em Fluxo MPEG-2 de Sistemas

Esta subseção descreve a instanciação do algoritmo genérico de ajuste elástico para fluxos de sistemas para a realidade do padrão MPEG-2 sistemas (ISO, 2000a).

A estrutura de um fluxo definida pelo padrão MPEG-2 pode ser visualizada na Figura 36 em dois formatos distintos: o Fluxo de Transporte (TS), que contém um ou mais programas e é apropriado para a transmissão e o armazenamento em ambientes ruidosos onde a ocorrência de erros é freqüente; e o Fluxo de Programa (PS), que contém apenas um programa e é adequado para uso em ambientes com baixas taxas de erros. Cada programa é definido como um conjunto de fluxos elementares que podem ou não ter algum relacionamento temporal entre si. A codificação dos elementos sincronizados entre si utiliza uma mesma base de tempo, ou referência de relógio. A proposta desta dissertação atualmente só manipula Fluxos de Programas, sendo o Fluxo de Transporte deixado como trabalho futuro. O texto a seguir refere-se, assim, sempre ao Fluxo de Programas.

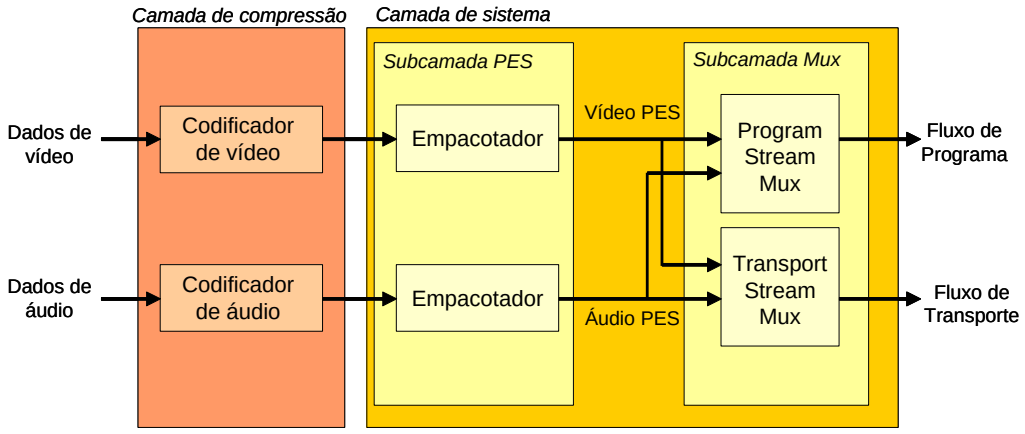


Figura 36 - Estrutura de fluxo MPEG-2 - fonte: (Cavendish, 2005).

Como ilustrado na Figura 36, a estrutura do MPEG-2 está dividida em duas camadas: a camada de compressão e a camada de sistema. A camada de sistema, definida no padrão MPEG-2 sistemas, é responsável pela divisão e encapsulamento de cada fluxo comprimido em pacotes; pela inserção de informações de sincronização entre fluxos de mídias diferentes; pela multiplexação dos fluxos encapsulados; e pelo transporte da informação de referência do relógio utilizado no codificador. A camada de compressão refere-se

à codificação de cada um dos dados audiovisuais, conforme especificado nos padrões MPEG-2 áudio e vídeo.

Os dados individuais de cada mídia, após sofrerem o processo de compressão, são denominados de fluxos elementares e são divididos em PACKETs na subcamada PES (*Packetized Elementary Stream*). O termo PACKET aqui deve ser interpretado como descrito na Seção 3.5. Os PACKETs são empacotados na subcamada de multiplexação em estruturas chamadas PACKs. Um PACK inicia com uma palavra de sincronismo, contém um cabeçalho e é seguido por zero ou mais PACKETS. A Figura 37 ilustra parte de um fluxo de sistemas contendo um PACK com quatro PACKETS e o cabeçalho do próximo PACK.

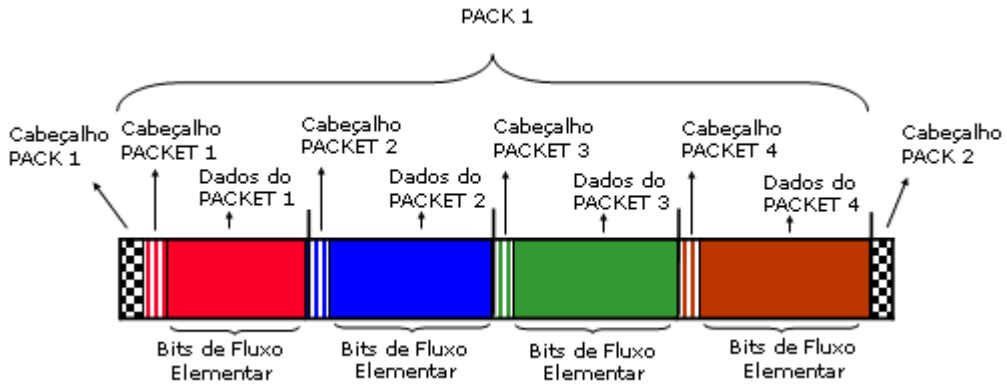


Figura 37 - Representação de parte de um fluxo de sistemas, com PACKs, PACKETs e bits de fluxos elementares.

O algoritmo extrator de estruturas de sistemas, descrito genericamente na Seção 3.5, é adaptado para contemplar a estrutura PACK, utilizando as seguintes regras:

a) Reconhecer e criar o cabeçalho do primeiro PACK. O primeiro PACK do fluxo inicia quando a primeira palavra de sincronismo for encontrada alinhada em bytes.

b) Analisar os bits seguintes ao atual

- Se for a palavra de sincronismo de outro PACK, voltar ao passo (a).
- Se for a palavra de sincronismo de um PACKET, reconhecer a estrutura e voltar ao passo (b).

- Se não for nenhum dos casos acima, indicar que fluxo de sistemas terminou.

Os principais campos do cabeçalho de um PACK para o contexto deste trabalho são: a palavra de sincronismo (cujo valor em hexadecimal é "*0x000001BA*") e o SCR (*System Clock Reference*). Os valores das marcas de tempo SCR significam o instante de tempo em que o último bit desses campos deve entrar no decodificador. Esse campo permite que o decodificador recupere a referência do relógio utilizado pelo codificador. O SCR é divido em duas partes, como indica a fórmula a seguir:

$$SCR(i) = SCR\_base(i) \cdot 300 + SCR\_ext(i)$$

Figura 38 - Cálculo do SCR em fluxos MPEG-2.

As duas partes são transmitidas no cabeçalho do PACK, sendo que a primeira delas (*SCR base*) contém *33* bits e a segunda (*SCR extension*) contém *9*. As fórmulas para calcular tais campos são apresentadas abaixo:

$$SCR\_base(i) = \left[\frac{((system\_clock\_frequency) \cdot t(i))}{300} \right] \% \, 2^{33}$$

Figura 39 - Cálculo do SCR base em fluxos MPEG-2.

$$SCR\_ext(i) = \left[(system\_clock\_frequency) \cdot t(i) \right] \% \, 300$$

Figura 40 - Cálculo do SCR extension em fluxos MPEG-2.

O valor $t(i)$ é um instante de tempo do relógio do codificador, o operador *%* representa o resto da divisão inteira (comum em linguagens de programação) e o valor do $system\_clock\_frequency$ é dado pela seguinte equação:

$$27000000 - 810 \leq system\_clock\_frequency \leq 27000000 + 810$$

Figura 41 - Intervalo de variação do *system_clock_frequency*.

Este trabalho assume que $system\_clock\_frequency = 27000000$.

Em relação à camada PES, os principais campos para o contexto deste trabalho são listados abaixo:

- *Packet_start_code_prefix*: palavra de sincronismo que possui o valor em hexadecimal igual a "*0x000001*".
- *Stream_id*: especifica o tipo e o número do fluxo elementar. Fluxos de áudio e vídeo possuem o número "*110xxxxx*" e "*1110xxxx*", sendo que a

seqüência de ‘*x*’s indica o número do fluxo elementar. Outros tipos de fluxos possuem outros números.

- *PES_packet_length*: indica o número de bytes contidos no PACKET contando a partir do final desse campo.
- *PTS_DTS_flags*: indica a presença ou não das marcas de tempo PTS e DTS, explicadas a seguir.
- *PTS (Presentation time stamp)*: marca de tempo de *33* bits que indica o tempo de apresentação de uma unidade de apresentação (imagens, amostras de áudio ou dados) no decodificador e, desse modo, mantém o sincronismo temporal entre diferentes fluxos elementares. O PTS é calculado utilizando a equação a seguir.

$$PTS(i) = \left[\frac{(system\_clock\_frequency) \cdot tp_n(i)}{300} \right] \% \, 2^{33}$$

Figura 42 - Cálculo do PTS em fluxos MPEG-2.

A fórmula acima é bastante parecida com a Figura 39 de calculo do *SCR base*. O *system_clock_frequency* é definido como antes e o $tp_n(i)$ é o instante de apresentação de uma unidade de apresentação presente dentro do PACKET.

- *DTS (Decoding time stamp)*: marca de tempo de *33* bits que indica o instante de decodificação de uma unidade de apresentação no decodificador e é calculada segundo a equação a seguir.

$$DTS(i) = \left[\frac{(system\_clock\_frequency) \cdot td_n(i)}{300} \right] \% \, 2^{33}$$

Figura 43 - Cálculo do DTS em fluxos MPEG-2.

Novamente, a fórmula acima é bastante parecida com a Figura 39 de calculo do *SCR base*. O *system_clock_frequency* é definido como antes e o $td_n(i)$ é o instante de decodificação de uma unidade de apresentação presente dentro do PACKET.

Como anteriormente mencionado, o algoritmo de ajuste elástico proposto deve recalcular as marcas de tempo sendo transmitidas, no caso SCR, PTS e DTS. Uma vez que os instantes de tempo são baseados no relógio do codificador, a

atualização desses valores se dá primeiramente extraindo o valor do instante de tempo original (dado por $t(i)$ no SCR, por $tp_n(i)$ no PTS e por $td_n(i)$ no DTS), acrescentando a este a variação de tempo decorrente do ajuste elástico para enfim gerar as novas marcas de tempo. O intervalo máximo permitido entre duas marcas do relógio consecutivas do SCR é de *0.7* segundos e por isto pode ser necessário inserir amostras de relógio quando o fator *f>1*. Um outro metadado que pode precisar ser alterado pelo algoritmo de ajuste é o campo *PES_packet_length.*

Por fim, vale mencionar que o tamanho das estruturas PACKs e PACKETs não são fixos (lembrando que esta subseção trata apenas de Fluxo de Programas), assim, teoricamente, nunca é preciso duplicar uma estrutura PACKET ao inserir um novo quadro do fluxo elementar. No entanto, no caso de quadros grandes, que são divididos em vários PACKETs, pode ser adequado criar novos PACKETs para transportar os quadros duplicados para não gerar um PACKET de tamanho muito diferente dos existentes no fluxo original.

4
Ferramenta de Ajuste Elástico

A ferramenta de ajuste foi desenvolvida com o objetivo de dar suporte às necessidades de ajuste das aplicações descritas na Seção 1.1. A Figura 44 ilustra o contexto de utilização da ferramenta e seus componentes principais.

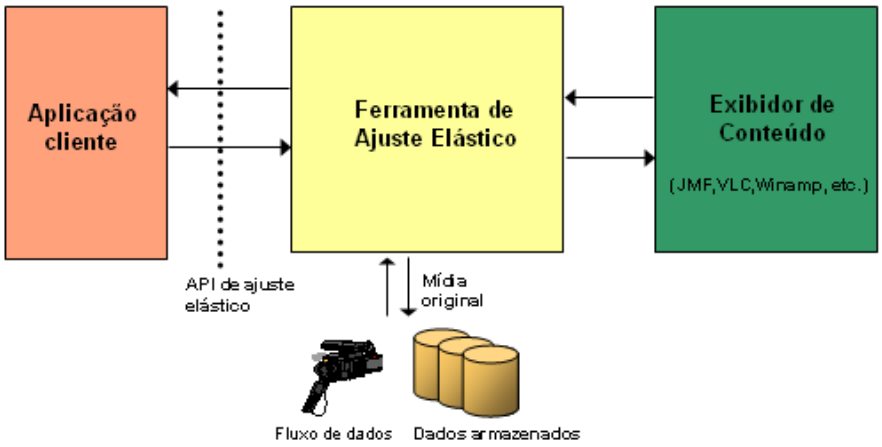


Figura 44 - Contexto de utilização da ferramenta de ajuste elástico.

A ferramenta de ajuste elástico recebe solicitações de aplicações cliente através de APIs (*Application Programming Interface*) de ajuste. Os dados audiovisuais de entrada podem ser buscados pela ferramenta (aplicações *pull*) ou recebidos como um fluxo de mídia (aplicações *push*). O fluxo processado utilizando o algoritmo de ajuste pode ser armazenado, disponibilizado como fluxo para a aplicação cliente ou enviado para exibição.

Quando o processamento de ajuste é realizado em *batch*, utilizando toda a capacidade de processamento oferecida pela máquina, diz-se que ocorre um ajuste em *tempo de compilação*. Se, por outro lado, o ajuste deve ser realizado periodicamente, mantendo-se razoavelmente constante a produção de quadros de mídia ajustados, diz-se que o ajuste ocorre em *tempo de execução*.

Todo o código fonte da ferramenta de ajuste foi desenvolvido utilizando a linguagem de programação Java, almejando independência de plataforma e fácil integração com outras ferramentas, que também estavam codificadas em Java (ver

Capítulo 5). As subseções a seguir detalham propostas de APIs para solicitação de serviços de ajuste elástico e suas implementações, uma visão geral da implementação da ferramenta de ajuste, a implementação do algoritmo de ajuste em fluxos de áudio e de sistemas e as mudanças no algoritmo de ajuste de vídeo utilizado.

4.1. APIs de ajuste elástico

As APIs de ajuste elástico definem como aplicações clientes podem solicitar serviços da ferramenta. Considerando que o modo de solicitação de ajuste em tempo de compilação e de execução possui características diferentes, uma API foi desenvolvida para cada um desses modos.

A Tabela 3 descreve uma proposta de interface genérica para solicitar processamento em tempo de compilação a ferramentas de ajuste, seguida, em particular, pela ferramenta proposta.

Método	**Dados de entrada**	**Descrição**
`config`	Mídia original, {fator de ajuste, trecho da mídia}, URI do arquivo de saída	Configura a ferramenta de ajuste definindo os parâmetros necessários antes de iniciar o processamento.
`start`	-	Inicia o processamento de ajuste.
`stop`	-	Suspende o processamento de ajuste.
`addTimescaleListener`	Observador do evento	Adiciona um observador à lista de observadores do evento de finalização do processamento de ajuste.
`getTimescaleInstant`	Instante de tempo	Retorna o novo valor de um determinado instante de tempo considerando a operação de ajuste realizada.
`getOutputTools`	-	Retorna a URI do arquivo de mídia gerado.
`getReport`	-	Retorna informações estatísticas sobre a operação de ajuste realizada.

Tabela 3 - Descrição dos métodos da API para realizar ajuste elástico em tempo de compilação.

O método `config` deve receber a mídia original, um conjunto formado por uma tupla de um fator de ajuste e um trecho da mídia a aplicar o fator e a URI do arquivo de saída a ser gerado.

O método `start` inicia e o `stop` pára o processamento do ajuste. A aplicação cliente pode se registrar como observadora[8] do evento de finalização do processamento de ajuste utilizando o método `addTimescaleListener`. Quando o ajuste terminar, é possível chamar o método `getTimescaleInstant` para descobrir o novo valor de um instante após a realização de um ajuste elástico e o método `getOutputTools` para obter a URI do arquivo gerado.

Por fim, a aplicação cliente pode obter estatísticas sobre um ajuste realizado (informações como quantidade de quadros do fluxo original, quantidade de quadros processados, tempo de processamento etc) chamando o método `getReport`.

A Tabela 4 descreve uma proposta de interface genérica similar à Tabela 3, mas para solicitar processamento em tempo de execução.

Método	**Dados de entrada**	**Descrição**
`config`	Mídia original, fator de ajuste, {instantes de tempo}	Configura a ferramenta de ajuste definindo os parâmetros necessários antes de iniciar o processamento.
`setFactor`	Fator de ajuste	Modifica o fator de ajuste a ser aplicado durante o processamento.
`start`	-	Inicia o processamento de ajuste.
`stop`	-	Suspende o processamento de ajuste.
`addTimescaleInstantListener`	Observador do evento	Adiciona um observador à lista de observadores do evento de encontrar o novo valor de instante de tempo específico.
`getOutputTools`	-	Retorna fluxo de bytes processado pela operação de ajuste.
`getReport`	-	Retorna informações estatísticas sobre a operação de ajuste realizada.

Tabela 4 - Descrição dos métodos da API para realizar ajuste elástico em tempo de execução.

O método `config` agora deve receber a mídia original, o fator de ajuste a ser utilizado, e, opcionalmente, um conjunto de instantes de tempo a monitorar durante a realização do ajuste.

O método `setFactor` deve poder ser chamado durante a realização de ajuste elástico para modificar o fator sendo utilizado. O método `start` inicia e o `stop` pára o processamento e a possível exibição do fluxo de dados.

[8] A semântica de observadores neste trabalho é similar ao descrito no padrão de projeto Observer (Gamma et al., 1995).

Caso a aplicação cliente tenha indicado no método `config` que deseja monitorar os instantes de tempo, ela deve se registrar como observadora dos eventos da computação do novo valor de um instante de tempo utilizando o método `addTimescaleInstantListener`.

O método `getOutputTools` deve ser chamado se a aplicação cliente deseja obter o fluxo de dados processado e o método `getReport` possui finalidade idêntica ao que já foi descrito na API em tempo de compilação.

4.1.1.Detalhes de Implementação das APIs de ajuste elástico

As APIs de ajuste elástico foram implementadas como duas interfaces em Java, uma para o ajuste em tempo de compilação (`ICompileTimeTimescalePlayer`) e outra para o tempo de execução (`IExecutionTimeTimescalePlayer`). Além dessas, uma terceira interface, chamada `ITimescalePlayer`, herda a definição das duas APIs, reunindo todos os métodos oferecidos pela ferramenta de ajuste. Todas essas classes são ilustradas na Figura 45.

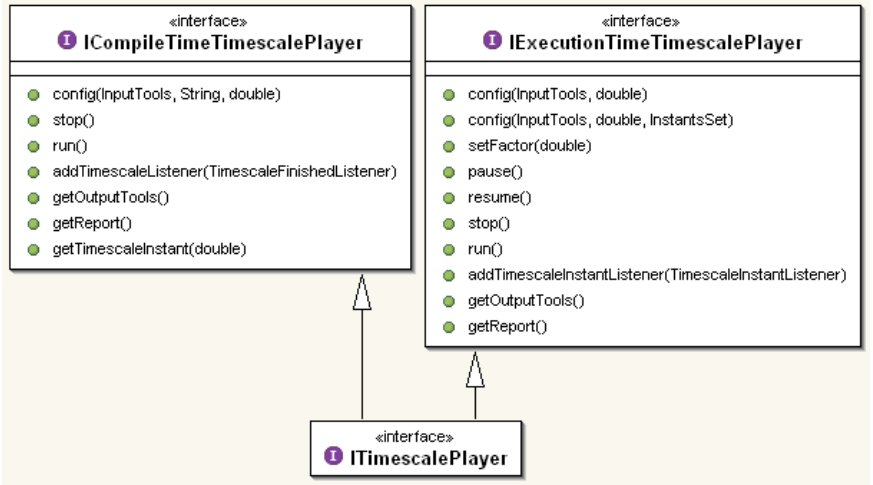


Figura 45 - Implementação das APIs de ajuste elástico.

Alguns comentários merecem ser realizados nessa implementação. Em primeiro lugar, o método `config` foi implementado como três métodos, variando os parâmetros recebidos. Por simplificação, o método de configuração para ajuste em tempo de compilação foi implementado podendo receber apenas um fator de ajuste (e não vários, conforme previsto na Tabela 1). Os possíveis parâmetros são: uma abstração do fluxo de mídia entrada (`InputTools inputTools`), o fator de ajuste (`double factor`), a URI do arquivo de saída (`String outputFile`) e um

conjunto de instantes de tempo para o algoritmo de ajuste monitorar (`InstantsSet instants`). A Tabela 5 descreve a diferença entre os métodos de configuração e quando utilizar cada um deles.

Assinatura do método	**Utilização**
`config (InputTools inputTools, String outputFile, double factor)`	Para ajuste em tempo de compilação. Utilizado quando se deseja gerar um arquivo contendo o processamento de dados de mídia de entrada.
`config (InputTools inputTools, double factor)`	Para ajuste em tempo de execução. Utilizado quando se deseja gerar em memória dados referentes ao processamento de mídia de entrada. Nesse caso, a aplicação cliente deverá paralelamente consumir os dados.
`config (InputTools inputTools, double factor, InstantsSet instants)`	Para ajuste em tempo de execução. Utilizado quando se deseja gerar em memória dados referentes ao processamento de mídia de entrada e paralelamente enviá-los para uma aplicação cliente. Além disso, deve-se monitorar um conjunto de instantes de tempo.

Tabela 5 - Descrição dos métodos de configuração da ferramenta de ajuste elástico.

Uma segunda observação refere-se aos métodos para iniciar o processamento. Uma vez que as APIs de ajuste herdam a definição da interface `Runnable`, é possível executar a ferramenta de ajuste elástico na mesma *thread* do processo da aplicação cliente ou em uma nova. A primeira opção é obtida simplesmente chamando o método `run` e a segunda chamando o método `start` de um objeto Java `Thread` que foi criado com uma interface da API de ajuste.

Os métodos `pause` e `resume` de `IExecutionTimeTimescalePlayer`, respectivamente, pára e retorna a realização do ajuste durante a exibição da mídia e são implementados, respectivamente, modificando o fator de ajuste para *1* e retornando o fator de ajuste para o valor que estava imediatamente antes do pause.

O Apêndice A apresenta trechos de código necessários para uma aplicação cliente utilizar a ferramenta de ajuste.

4.2. Visão Geral da Implementação da Ferramenta de Ajuste Elástico

Esta seção apresenta uma visão geral da implementação da ferramenta de ajuste elástico. A Figura 46 ilustra os componentes da ferramenta de ajuste elástico detalhando Figura 44. Na nova figura, o componente algoritmo de ajuste elástico foi detalhado nos componentes de ajuste para áudio, vídeo e sistemas. O algoritmo de sistemas utiliza os algoritmos de áudio e vídeo.

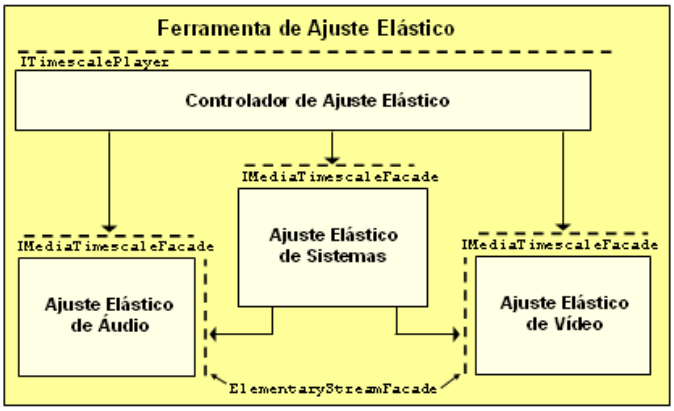


Figura 46 - Detalhes da ferramenta de ajuste elástico.

A Figura 47 ilustra a definição das classes principais considerando o propósito desta seção. A classe `TimescalePlayer` implementa `ITimescalePlayer`, e, como tal, contém todos os métodos oferecidos pela ferramenta de ajuste. `TimescalePlayer` encaminha as solicitações recebidas aos controladores de mídia específicos, no caso áudio, vídeo e sistemas, através de uma interface comum definida pela interface `IMediaTimescaleFacade`[9].

A classe `AbstractTimescaleFactory` é uma fábrica[10] de objetos necessários para a realização do ajuste. `AbstractTimescaleFactory` se especializa em `AudioTimescaleFactory`, `VideoTimescaleFactory` e `SystemTimescaleFactory`. São essas classes que identificam o formato da mídia recebida e instanciam as classes responsáveis por manipular o formato encontrado. Ter uma interface única para a criação de objetos facilita a uniformização do processamento da mídia.

[9] A semântica do termo Facade (em Português, fachada) neste trabalho é análoga ao descrito no padrão de projeto Facade (Gamma et al., 1995).

[10] A semântica do termo factory neste trabalho é análoga ao descrito no padrão de projeto AbstractFactory (Gamma et al., 1995).

Figura 47 - Realização das APIs de ajuste elástico pela ferramenta de ajuste.

Algumas classes mais genéricas são utilizadas em vários momentos durante o processamento e por isso são resumidas nas subseções a seguir.

4.2.1. Subsistema de Entrada

O subsistema de entrada é responsável por obter os dados de mídia a partir da leitura de um arquivo binário ou da recepção de um fluxo de dados. Os clientes do subsistema de entrada não sabem como os dados estão sendo obtidos. A implementação desse subsistema é composta pelas classes ilustradas na Figura 48.

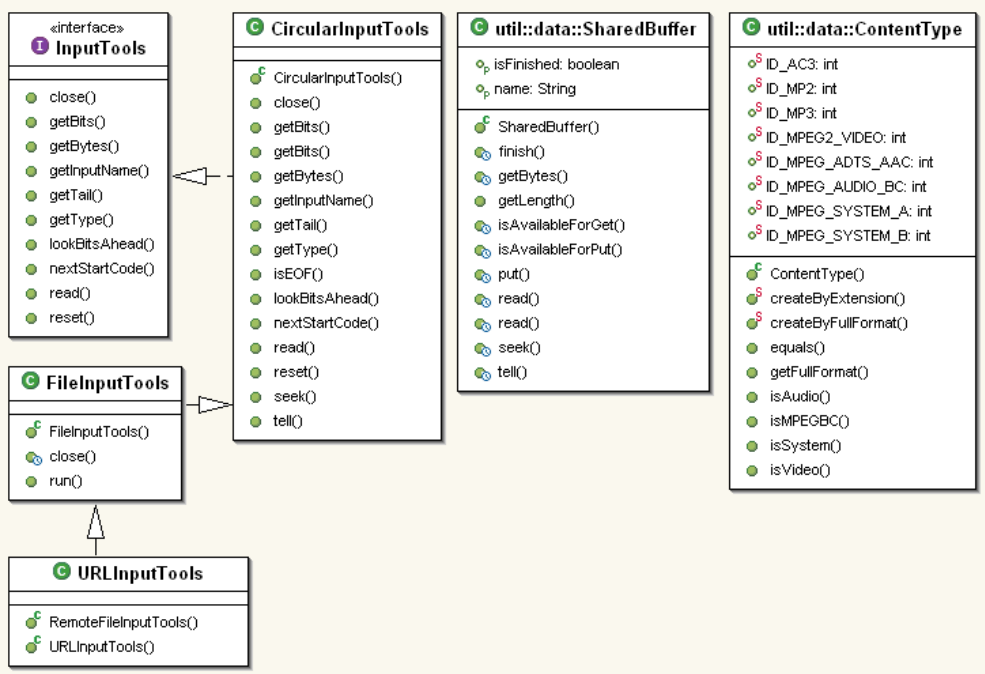


Figura 48 - Classes do subsistema de entrada.

A interface `InputTools` é instanciada como `CircularInputTools`, `FileInputTools` ou `RemoteFileInputTools`. A classe `CircularInputTools` é capaz de realizar a escrita e leitura de dados em um buffer circular de dados, representado pela classe `SharedBuffer`.

A classe `FileInputTools` modela a leitura do arquivo e armazenamento dos dados lidos em memória e a classe `URLInputTools` encapsula a leitura do conteúdo de uma URL e seu armazenamento em memória. Essas duas classes herdam da classe `CircularInputTools` porque a estrutura de dados interna para armazenar os dados lidos é circular, denominada `SharedBuffer`. Enquanto a *thread* de leitura armazena os dados nessa estrutura, uma outra *thread* pode consumir os dados.

Por fim, a classe `ContentType` deve ser instanciada com o tipo de dados da mídia sendo manipulada. Atualmente, a identificação do tipo de dados pode se basear na extensão da URL do arquivo de entrada ou na análise do fluxo de dados.

4.2.2. Subsistema de Saída

O subsistema de saída é responsável por escrever os dados recebidos em um arquivo binário ou alimentar uma estrutura de dados em memória. A implementação desse subsistema é composta pelas classes ilustradas na Figura 49.

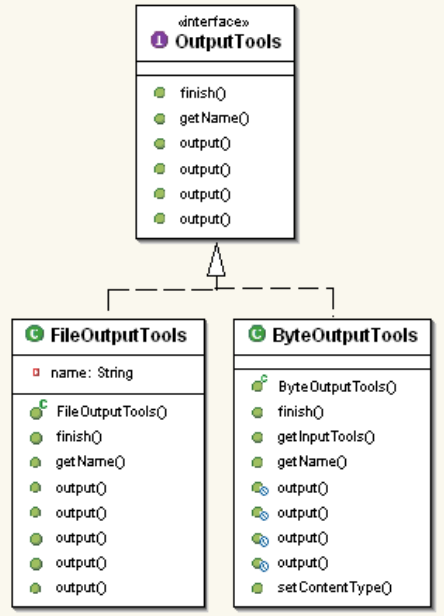


Figura 49 - Classes do subsistema de entrada.

Os clientes do subsistema de saída solicitam os serviços através da interface `OuputTools` e não conhecem o mecanismo de saída dos dados usado. As classes `FileOutputTools` e `ByteOutputTools` implementam `OutputTools` e encapsulam a lógica de escrever em um arquivo ou alimentar um buffer circular em memória, representado pela classe `SharedBuffer`. A classe `ByteOutputTools` conhece classes do subsistema de entrada, pois alimenta um buffer do tipo `SharedBuffer` e, através do método `getInputTools`, é capaz de retornar um `InputTools` que lê dados do mesmo buffer.

4.2.3. Outras classes

Algumas classes importantes que podem ser utilizadas pelos subsistemas de áudio, vídeo e sistemas merecem destaque. `IAssembler` é a interface genérica a ser implementada por classes responsáveis por montar um conjunto de objetos para representar o fluxo de dados a partir de um fluxo binário e `IDisassembler` é a interface que deve ser implementada por classes que possuem a responsabilidade inversa, isto é, extrair um fluxo binário a partir de um conjunto de objetos que representa o fluxo de dados.

Algumas classes definem eventos específicos da regra de negócios da aplicação e interfaces de observadores desses eventos. Atualmente, existem eventos da finalização da realização do ajuste (classes `TimescaleFinishedEvent`

e `TimescaleFinishedListener`) e de descoberta de um novo valor para um determinado instante de tempo que foi indicado na configuração da ferramenta de ajuste (classes `TimescaleInstantEvent` e `TimescaleInstantListener`).

Existem classes para calcular novos valores de determinados instantes de tempo decorrente da realização de ajuste elástico. A classe `InstantsRetrieval` é chamada para calcular tais valores durante a realização do ajuste e `OfflineInstantsRetrieval` é utilizada quando o ajuste já foi realizado.

Outras classes representam parâmetros e resultados do processamento de ajuste elástico. As principais são: `Factor` (representa o fator de ajuste), `ProcessmentParameters` (representa parâmetros de processamento na configuração do ajuste elástico), `FinalReport` (dados estatísticos sobre a realização do ajuste) e `InstantsSet` (conjunto de instantes de tempo a monitorar durante realização de ajuste e os novos valores encontrados para tais instantes).

4.3. Implementação do Algoritmo de Ajuste Elástico de Áudio

A implementação do ajuste elástico de áudio é composta por controladores, subsistemas de montagem, processamento e desmontagem de dados e classes que modelam os quadros do fluxo de áudio. A Figura 50 apresenta uma visão geral da dependência entre os componentes no algoritmo de ajuste.

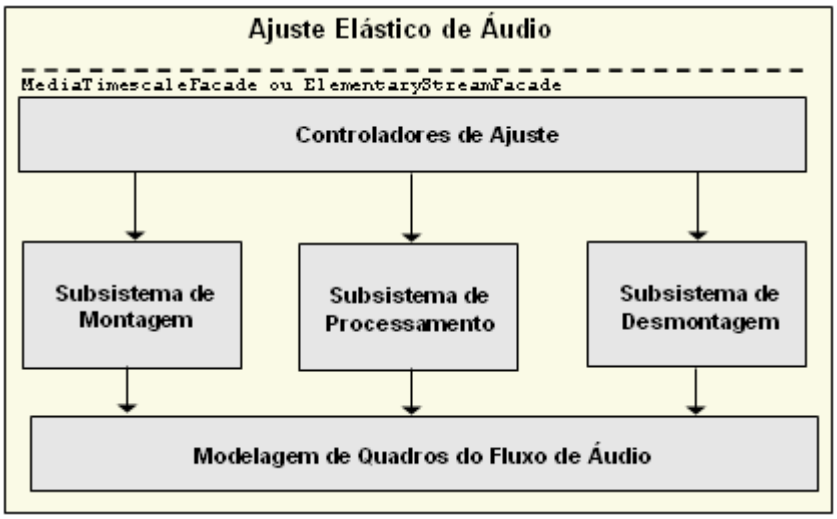


Figura 50 - Visão geral da implementação do algoritmo de ajuste de áudio.

Os controladores contêm a lógica de orquestração do serviço solicitado, repassando ações para os subsistemas adequados. Durante a realização do ajuste, cada um desses subsistemas manipula uma parte do fluxo de áudio, fazendo com

que o algoritmo seja aplicado de forma incremental. Todos os subsistemas dependem do modelo de quadros do fluxo de áudio. As subseções a seguir detalham cada componente apresentado.

4.3.1. Modelagem dos Dados de Áudio com Compressão

Os dados do fluxo de áudio são modelados por um conjunto de classes genéricas (independente do formato) e outras específicas para cada formato suportado. A Figura 51 apresenta as principais classes genéricas.

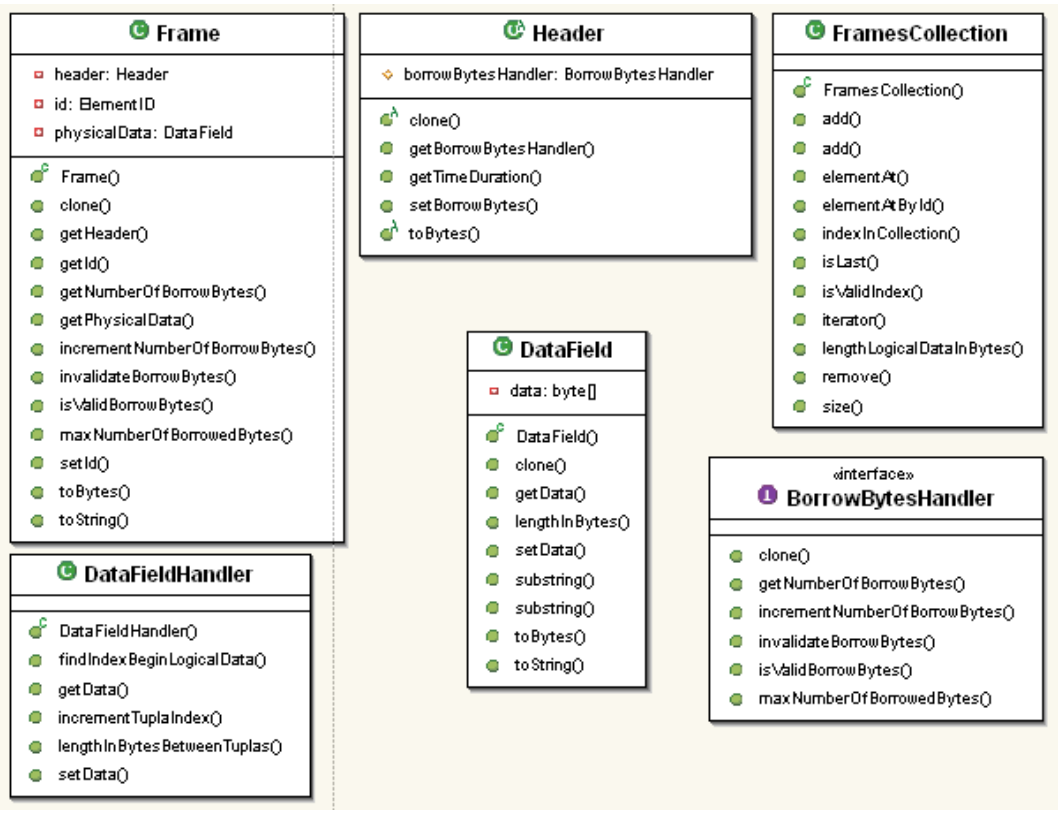


Figura 51 - Principais classes do formato genérico de áudio.

A classe `Frame` representa um quadro do fluxo. As classes `Header` e `DataField` representam o cabeçalho e campo de dados de um quadro, respectivamente. As classes `DataFieldHandler` e `BorrowBytesHandler` manipulam os bits do campo de dados do quadro e a classe `FramesCollection` modela uma coleção de quadros.

Além das classes genéricas, existem classes para modelar as particularidades de cada formato, que herdam das classes de formato genérico. Por exemplo, para o formato MP3, foram criadas as classes da Figura 52, além de outras que são para quaisquer formatos de áudio MPEG.

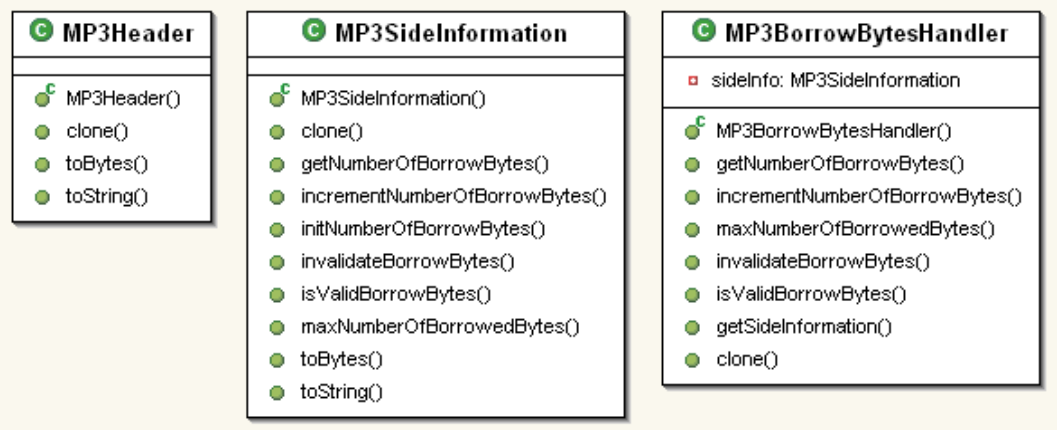


Figura 52 - Principais classes do formato MP3.

4.3.2. Controladores de Ajuste

Os controladores do ajuste de áudio são ilustrados na Figura 53. A classe `AudioTimeScaleFacade` é a fachada do algoritmo de ajuste de áudio, ou seja, o ponto único de acesso a todas as classes responsáveis pela realização do processamento.

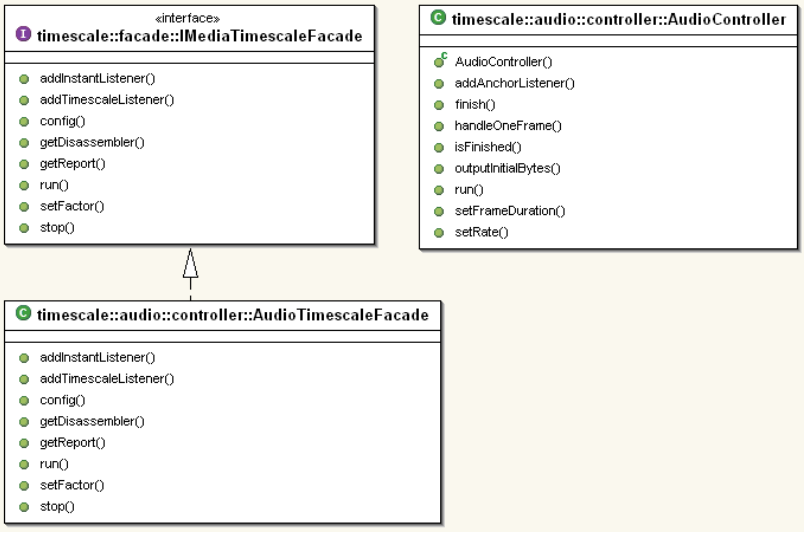


Figura 53 - Controladores do ajuste elástico de áudio.

Sendo a classe que disponibiliza os serviços de ajuste de áudio, `AudioTimeScaleFacade` implementa a interfaces `IMediaTimescaleFacade` (classe que especifica serviços de ajustes dos controladores de áudio, vídeo e sistemas) e `Runnable` (para possibilitar executar o ajuste em uma nova *thread*).

A fachada executa o ajuste elástico utilizando os dados que foram passados para a classe no seu construtor e solicitando serviço da classe `AudioController`.

Para efetuar o ajuste elástico, a classe `AudioController` solicita serviços dos subsistemas de montagem, processamento e desmontagem. Essa classe contém a lógica de orquestração dos serviços realizados por esses subsistemas. Durante a realização do ajuste, cada um desses subsistemas processa uma parte do fluxo de áudio, fazendo com que a montagem dos quadros de áudio, o processamento e o envio para saída dos bytes extraídos dos quadros processados ocorram de forma incremental, como ilustrado na Figura 54 para um instante qualquer no meio do processamento de um fluxo.

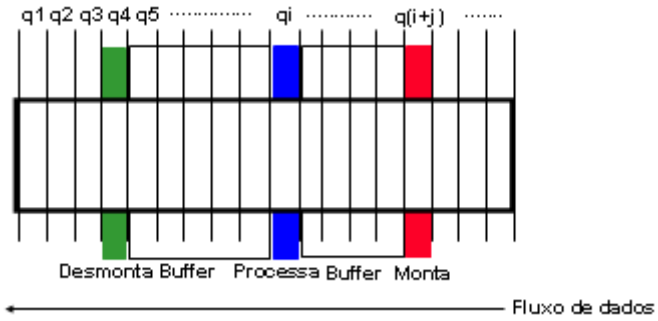


Figura 54 - Montagem (incluindo leitura), processamento e desmontagem (incluindo escrita) de dados de áudio.

Na Figura 54, cada *qi* é um quadro do fluxo. No exemplo, os quadros *q1*, *q2* e *q3* já passaram por todo o algoritmo de ajuste e estão em formato binário; o quadro *q4* já passou pelo subsistema de processamento e está sendo desmontado; o quadro *qi* está sendo processado; o quadro *q(i+i)* está sendo ainda montado para futuro processamento do algoritmo de ajuste e os quadros posteriores a *q(i+j)* ainda estão no formato binário original. Os *buffers* são necessários se existe dependência entre um quadro de áudio e os quadros anteriores e posteriores, uma vez que, nesse caso, o processamento do quadro *qi* deve modificar bits de quadros vizinhos.

4.3.3. Subsistemas de Montagem e Desmontagem

O subsistema de montagem é responsável por criar as classes do formato de áudio descritos na Subseção 4.3.1 a partir de dados binários do fluxo de mídia fornecidos pelo subsistema de entrada (ver Subseção 4.2.1). Tais operações são realizadas através de um conjunto de classes que herdam da classe `IAssembler`, (ver Subseção 4.2.3).

A interface `IAudioAssembler` define os serviços oferecidos pelo subsistema de montagem de áudio. A classe `AudioAssembler` implementa `IAudioAssembler`, fazendo uso da classe `GenericAudioAssembler`, que contém as regras de montagem de um quadro do formato genérico de áudio. No entanto, a classe `GenericAudioAssembler` deve ser especializada para manipular as particularidades de cada formato suportado, como ilustra a Figura 55.

O subsistema de desmontagem é responsável por extrair e enviar ao subsistema de saída dados binários a partir das classes de formatos de áudio. Tais operações são realizadas através de um conjunto de classes que herdam da classe `IDisassembler` (ver Subseção 4.2.3). Uma vez que as classes do modelo genérico de áudio possuem um método chamado `getBytes`, uma única classe, chamada `AudioDisassembler`, é capaz de realizar a desmontagem de todos os formatos de áudio. `AudioDisassembler` implementa os serviços de desmontagem especificados pela interface `IAudioDisassembler`. As classes de desmontagem também estão ilustradas na Figura 55.

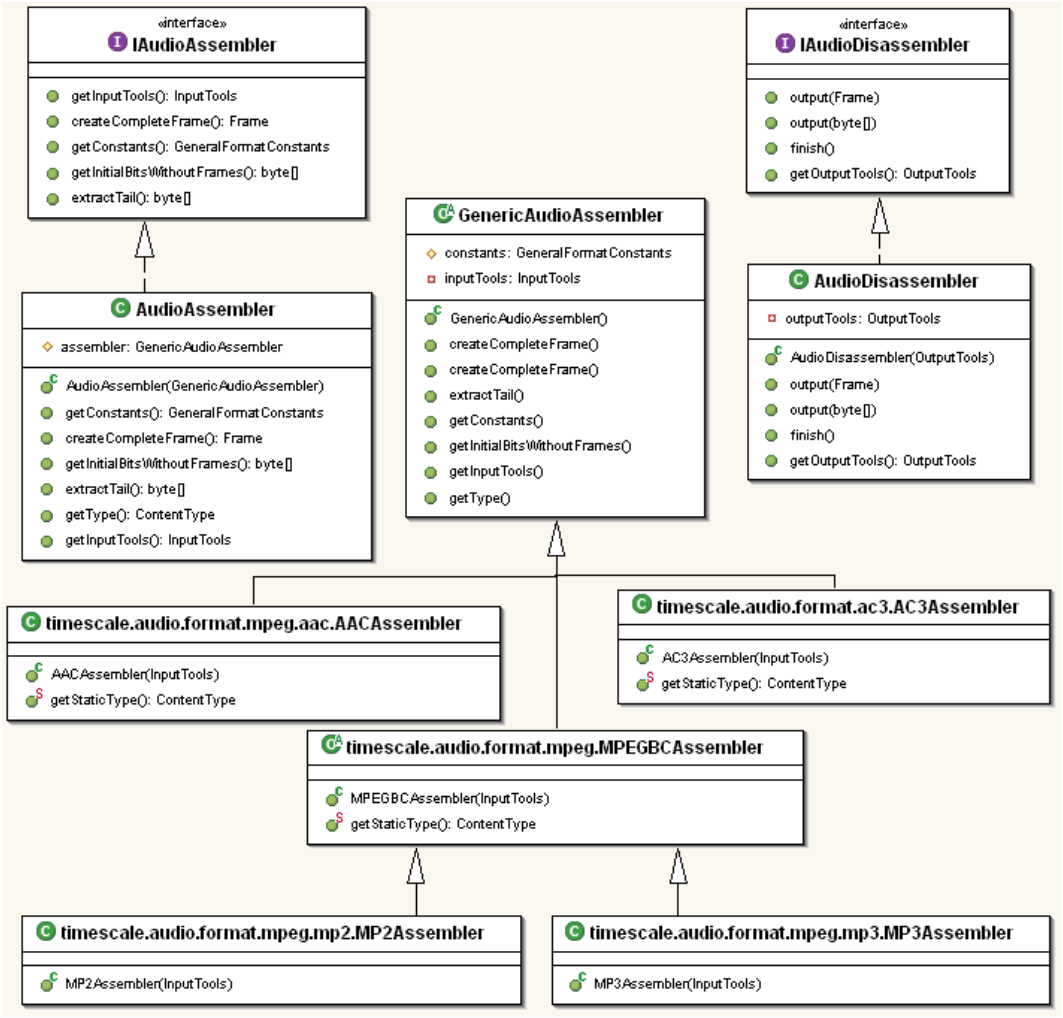


Figura 55 - *Assemblers* e *disassembler* de objetos de formatos de áudio.

4.3.4. Subsistema de Processamento

O subsistema de processamento aplica o algoritmo de ajuste nas classes do formato genérico de áudio e é independente do formato de áudio utilizado. As principais classes desse subsistema estão ilustradas na Figura 56.

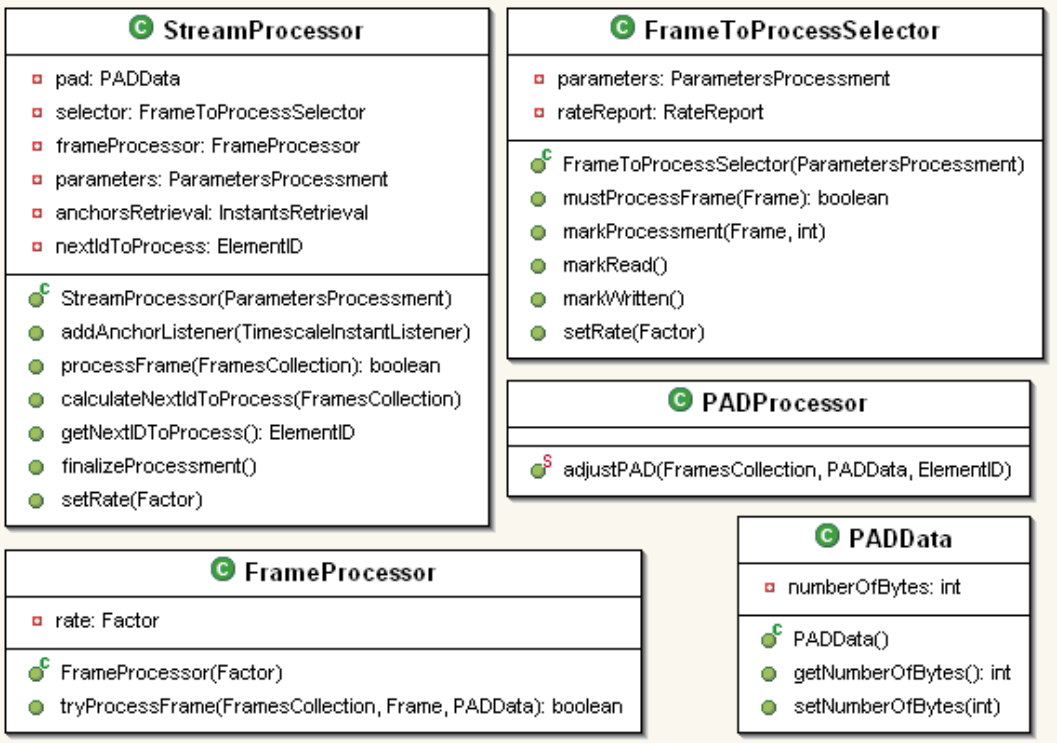


Figura 56 - Principais classes do subsistema de processamento de áudio.

A classe `StreamProcessor` é responsável por processar um fluxo de áudio. Para realizar essa tarefa, essa classe percorre os quadros do fluxo e para cada um utiliza a classe `PADProcessor` para deslizar o PAD, a classe `FrameToProcessSelector` para verificar se este quadro deve ser processado (levando em consideração o fator de ajuste originalmente aplicado e o fator já atingido) e, em caso positivo, utiliza a classe `FrameProcessor` para tentar processá-lo (uma vez que nem todo quadro pode ser processado).

A classe `FrameProcessor` encapsula o algoritmo de ajuste utilizado no quadro recebido. O algoritmo para diminuir ou aumentar a duração de um fluxo de áudio é, respectivamente, remover ou duplicar o quadro do fluxo de dados. O processamento de ajuste elástico pode gerar um conjunto de bytes de PAD no

fluxo de mídia (ver Seção 3.3). Um objeto da classe `PADData` armazena informações sobre esses bytes. O deslizamento do PAD executado pela classe `PADProcessor` é realizado transferindo o PAD de um quadro para o seguinte.

Como já mencionado, a classe `FrameToProcessSelector` seleciona quais quadros devem ser processados. Para tal, é necessário calcular qual o fator de ajuste gerado até o momento e comparar com o fator a ser atingido. Caso ocorra uma mudança do fator, o algoritmo de seleção ignora o fator anteriormente aplicado e realiza os novos cálculos considerando que iniciou um novo fluxo de dados que precisa ser ajustado com um novo fator.

4.4. Implementação do Algoritmo de Elástico em Fluxos de Sistemas

A implementação do algoritmo de ajuste elástico em fluxos de sistemas é composta por uma fachada, subsistemas de demultiplexação, montagem, processamento, multiplexação e desmontagem, e classes que modelam as estruturas do fluxo de sistemas. A Figura 57 apresenta uma visão geral da dependência entre os componentes citados.

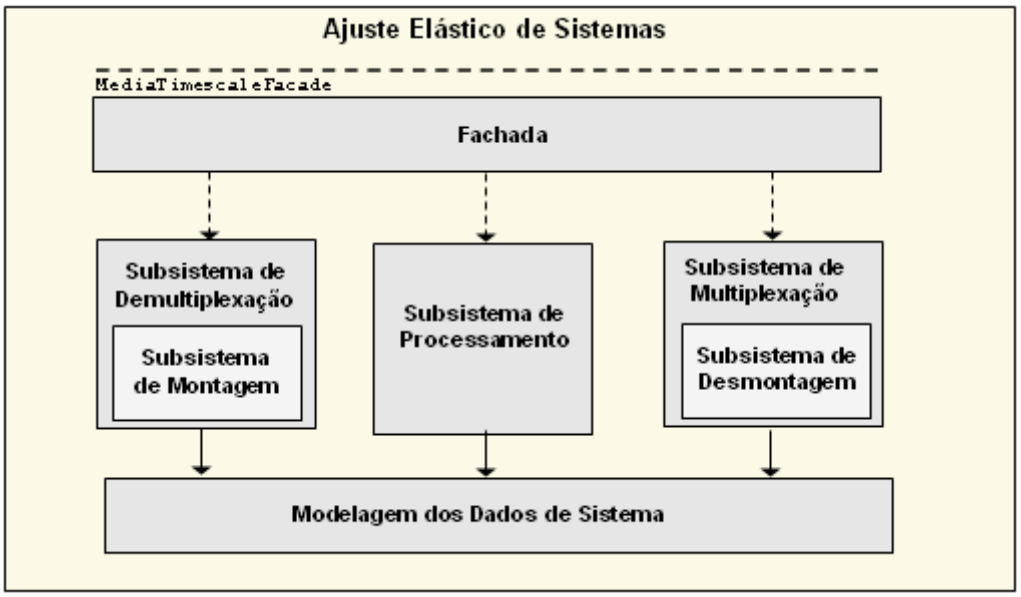


Figura 57 - Visão geral da implementação do algoritmo de ajuste em fluxos de sistemas.

A fachada implementa a interface `IMediaTimescaleFacade` e possui um relacionamento com os demais subsistemas através de setas tracejadas. Isso acontece porque a fachada, representada pela classe `SystemTimescaleFacade`,

não controla o fluxo desses subsistemas, mas apenas instancia suas classes em diferentes *threads*.

A maneira que as *threads* desses três subsistemas interagem entre si e com os subsistemas que realizam ajuste em fluxos elementares é apresentada na Figura 58. O fluxo original é demultiplexado em fluxos elementares, processado por algoritmos de ajuste sob o controle do subsistema de processamento e, finalmente, os fluxos elementares processados são multiplexados para compor um novo fluxo de sistema.

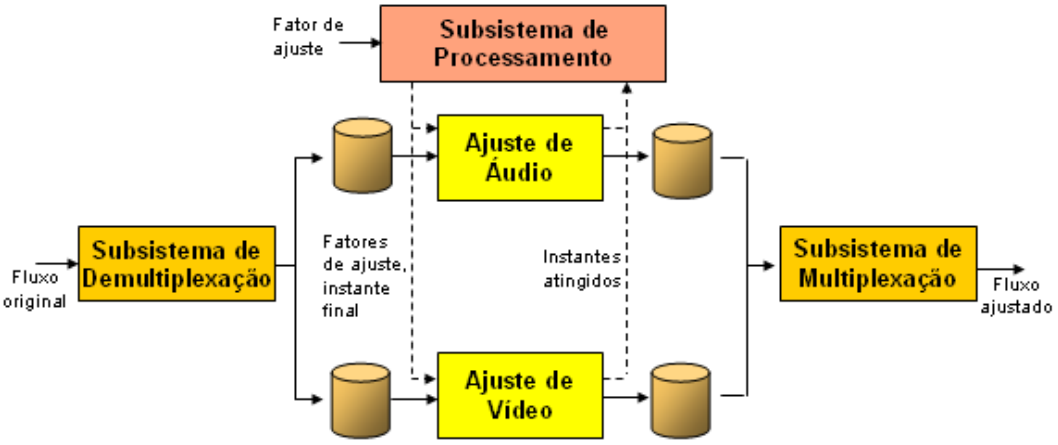


Figura 58 - Algoritmo de ajuste de dados audiovisuais.

O subsistema de processamento recebe o fator de ajuste desejado pela aplicação cliente e chama os algoritmos dos fluxos elementares instanciando novas *threads* configurando o fator de ajuste a ser aplicado até um determinado instante final em relação à mídia original. Os algoritmos dos fluxos elementares realizam o processamento e indicam o instante de tempo no fluxo processado equivalente ao instante recebido. O subsistema de processamento então recalcula o fator de ajuste e solicita a aplicação do ajuste no próximo trecho da mídia original (ver Seção 3.5). A Figura 59 ilustra o tempo de vida das *threads* no decorrer da execução do ajuste.

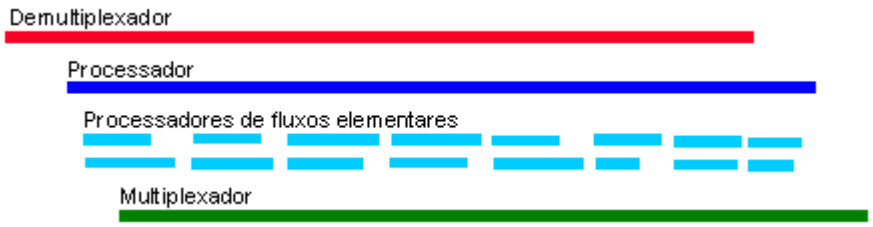


Figura 59 - Tempo de vida das *threads* responsáveis pelo ajuste elástico.

As subseções a seguir detalham a modelagem de dados de sistema, os subsistemas de montagem e desmontagem, de multiplexação e demultiplexação e de processamento, e as mudanças necessárias nos algoritmos de ajuste de fluxos elementares.

4.4.1. Modelagem dos Dados de Sistemas

Os dados do fluxo de sistemas são modelados por uma interface genérica de metadados chamada `IMediaMetadata` e um conjunto de classes especificamente desenvolvidas para representar as estruturas PACK e PACKET do MPEG-2. A Figura 60 apresenta tais classes.

A interface `IMediaMetadata` é a única conhecida pelos fluxos elementares. Seu método `adjustClock` deve ser chamado para atualizar as marcas de tempo inseridas no fluxo. Os métodos `getMetadataIndex` e `setMetadataIndex` servem para obter e modificar o início dos metadados nos bits do quadro do fluxo elementar.

A classe `MediaMetadata` implementa os serviços da interface `IMediaMetadata` para o fluxo MPEG-2 sistemas. As classes `SystemPacket` e `SystemPack` representam as estruturas PACKET e PACK de sistemas. A classe `SystemPacket` é abstrata e deve ser instanciada como `AVSystemPacket`, se for um PACKET de áudio ou vídeo, ou como `DataSystemPacket`, se for um PACKET de outro tipo. A classe `SystemHeader` é uma subestrutura do `SystemPack` e a classe `CollectionPacket` modela um conjunto de estruturas PACKET de um mesmo fluxo elementar.

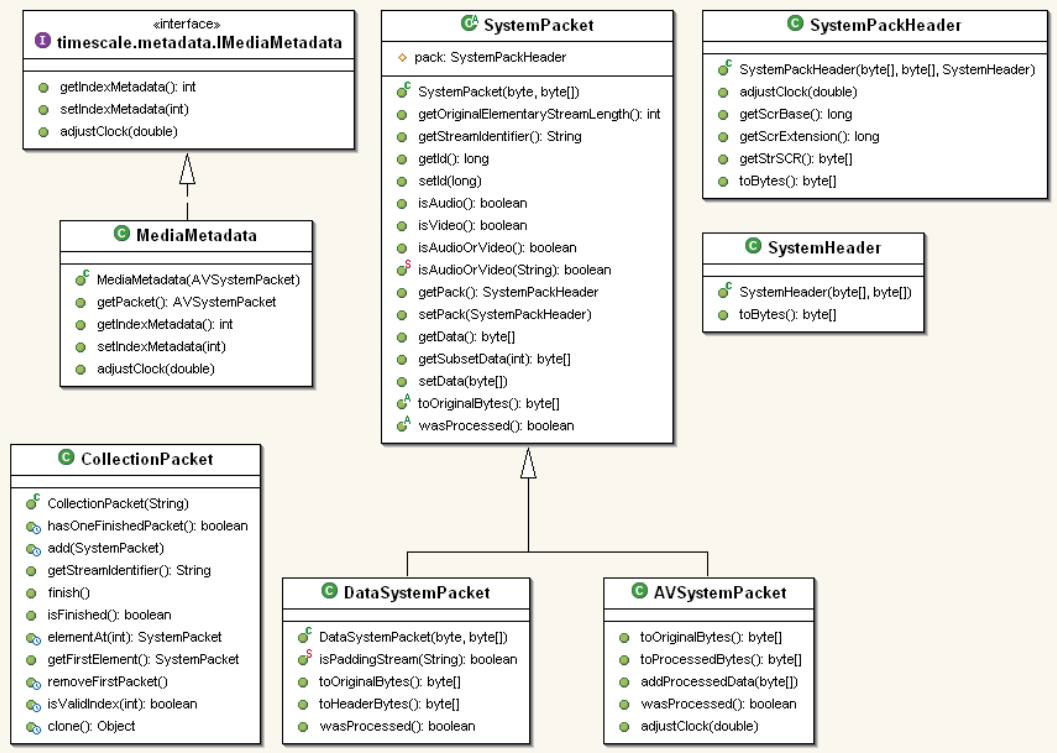


Figura 60 - Classes do formato de MPEG-2 sistemas.

4.4.2. Subsistemas de Montagem e Desmontagem

Os subsistemas de montagem e desmontagem são responsáveis por criar classes que descrevem as estruturas de sistemas descritas na Subseção 4.4.1 (`SystemAssembler`) e por retirar o fluxo binário a partir dessas classes (`SystemDisassembler`). Para tais atividades, os subsistemas utilizam respectivamente o subsistema de entrada (ver Subseção 4.2.1) e de saída (ver Subseção 4.2.2). A Figura 61 apresenta as classes citadas.

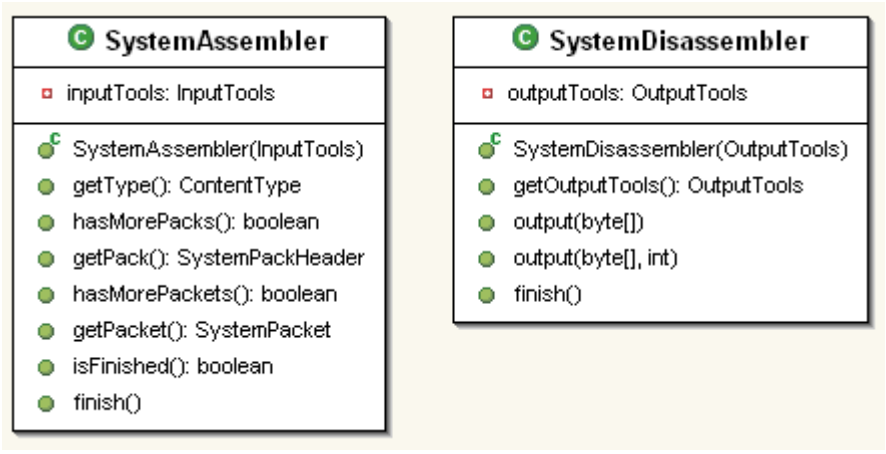


Figura 61 - Classes dos subsistemas de montagem e desmontagem.

4.4.3. Subsistemas de Multiplexação e Demultiplexação

O subsistema de demultiplexação utiliza o subsistema de montagem (ver Subseção 4.4.2) para criar estruturas PACKETs do fluxo de sistemas original e é responsável por ordenar e separar os PACKETs dos vários fluxos elementares. Ao encontra um novo fluxo elementar, esse subsistema deve instanciar um subsistema de fluxo elementar para processá-lo. A classe `SystemDemultiplex` implementa tais atividades.

A referência aos dados de PACKETs dos diferentes fluxos elementares e os subsistemas que estão manipulando cada fluxo são armazenados na classe `ElementaryStreams`. Além disso, essa classe contém semáforos que sinalizam quando o subsistema de processamento e de multiplexação devem realmente começar a executar (ver Figura 59).

O subsistema de multiplexação, implementado pela classe `SystemMultiplex`, coleta os dados processados dos diferentes fluxos elementares e realiza a multiplexação baseando-se na ordenação realizada durante a demultiplexação. O multiplexador inicia seu processamento quando o primeiro PACKET é processado. A *thread* de execução de multiplexação deve dormir toda vez que o número do próximo PACKET ainda não estiver disponível para não consumir recursos de processamento desnecessários.

A Figura 62 ilustra as classes discutidas nesta subseção.

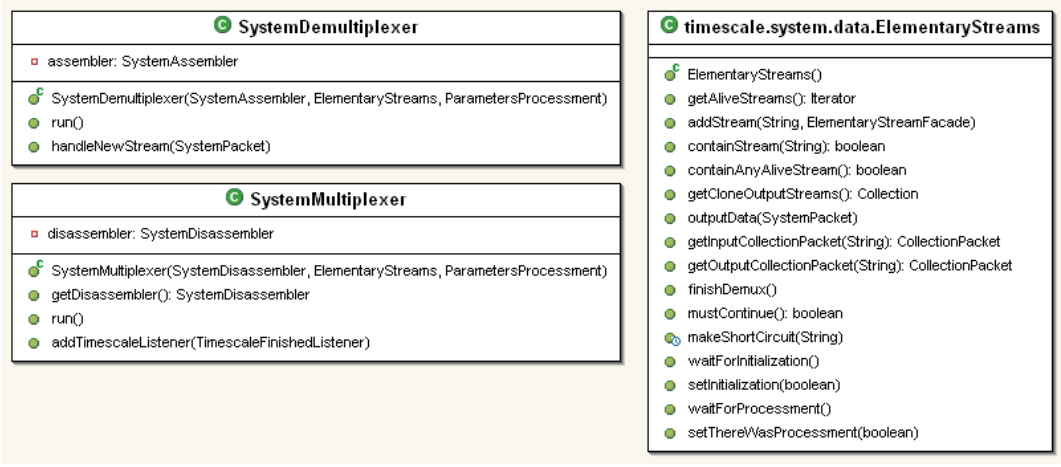


Figura 62 - Classes dos subsistemas de multiplexação e demultiplexação.

4.4.4. Subsistemas de Processamento

O subsistema de processamento é responsável por monitorar a execução da realização de ajuste elástico nos fluxos elementares. A Figura 63 ilustra suas classes principais.

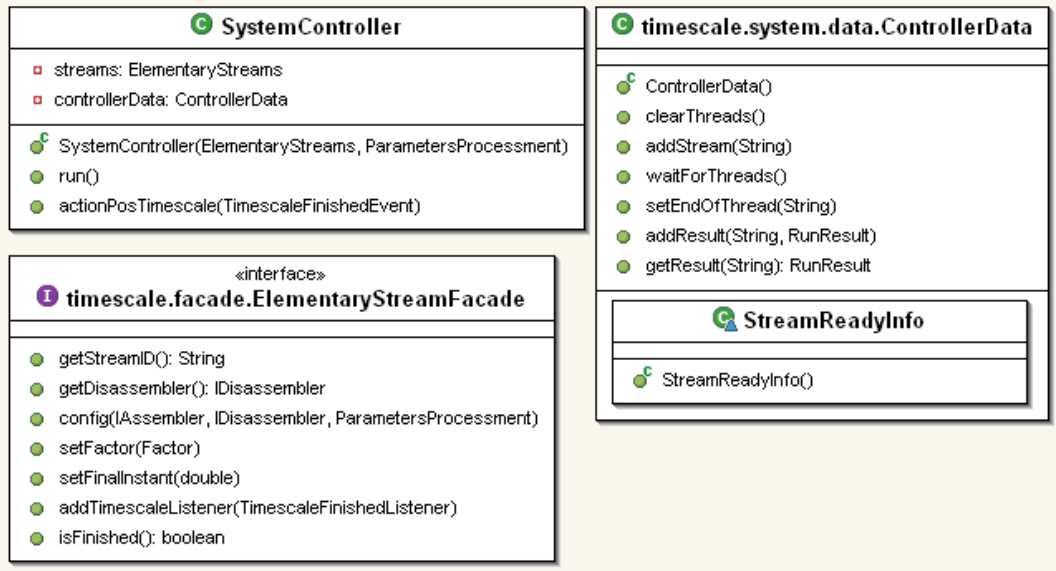


Figura 63 - Controladores do algoritmo de ajuste em fluxos de sistemas.

A classe `SystemController` mantém a referência dos subsistemas de ajuste dos fluxos elementares e ordena a execução de todos eles em um pequeno trecho da mídia original, de trecho em trecho, até o final da mídia. Em cada trecho da mídia, o controlador percorre a lista de todos os subsistemas de ajuste e cria uma *thread* para a execução de cada um configurando seu instante final e fator de ajuste a ser utilizado, adiciona o subsistema a um objeto da classe `ControllerData` e espera que a execução de todas as *thread* iniciadas se encerre. A interface `ElementaryStreamFacade` especifica como os serviços dos subsistemas de ajuste de fluxos elementares devem ser solicitados (ver Figura 46).

Alguns algoritmos descritos na Seção 3.5 ainda não foram implementados. São eles: verificação de sincronismo intermídia, possibilidade de criar ou remover quadros PACKETs e inserção de novas amostras de relógio se ultrapassado tempo máximo entre transmissões consecutivas.

4.4.5. Mudanças nos Algoritmos de Ajuste de Fluxos Elementares

Algumas mudanças no algoritmo de ajuste de áudio e vídeo são necessárias para estes serem executados como fluxos elementares. Este trabalho desenvolveu apenas as mudanças para o fluxo de áudio, deixando as mudanças do fluxo de vídeo para trabalho futuro. As classes da Figura 64 foram criadas para esse propósito.

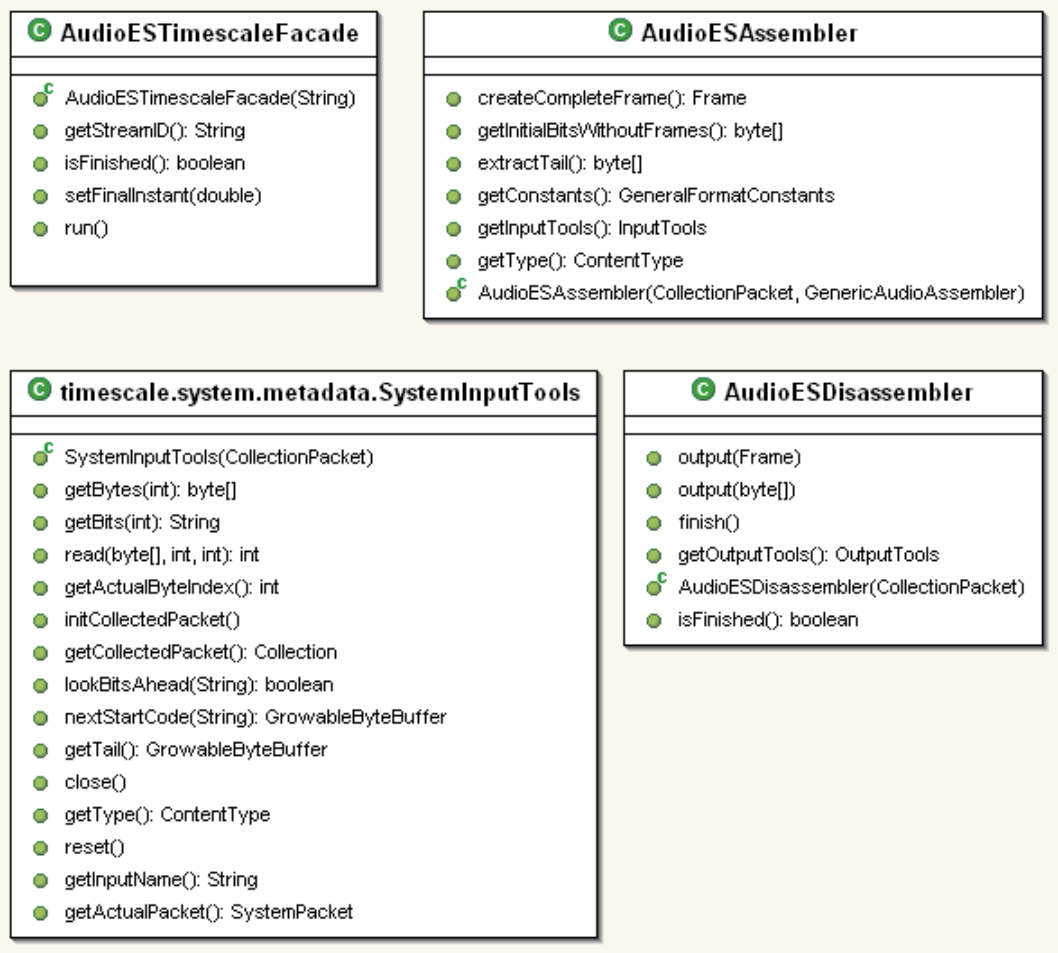


Figura 64 - Classes para realizar ajuste em fluxo elementar de áudio.

A classe `AudioESTimescaleFacade` herda da classe `AudioTimescaleFacade` e adiciona métodos para tratar especificidades do ajuste como um fluxo elementar, como o método `getStreamID` (para retornar o identificador único do fluxo elementar do fluxo de sistemas a que pertence) e o `setFinalInstant` (para configurar o instante da mídia original até quando deve ocorrer a execução do ajuste).

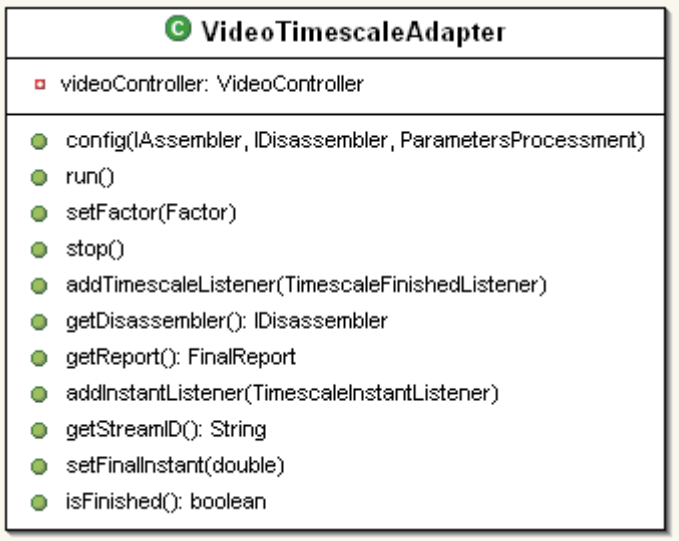
A classe `AudioESAssembler` implementa `IAudioAssembler` e, como tal, `é` responsável por criar objetos do modelo de dados de áudio. `AudioESAssembler`

extrai os dados binários para montagem a partir das estruturas de sistema. Para tal, utiliza a classe `SystemInputTools`, que é capaz de extrair bytes dos dados dos PACKETS do fluxo de sistemas. `AudioESAssembler` transforma o cabeçalho das estruturas de sistemas em metadados do fluxo de áudio. A classe `AudioESDisassembler` implementa `IAudioDisassembler` e, como tal, deve extrair dados binários do modelo de dados de áudio. `AudioESDisassembler` utiliza os dados extraídos e os metadados para compor as estruturas do fluxo de sistemas.

Outras mudanças no ajuste elástico dos fluxos elementares já foram comentadas, como a necessidade de ajustar os relógios de fluxos de sistemas (ver Subseção 4.4.1).

4.5. Mudanças no Algoritmo de Ajuste Elástico de Vídeo para Integração

O algoritmo de ajuste elástico de vídeo utilizado foi desenvolvido por Cavendish (Cavendish, 2005). Entretanto, a interface de serviços disponibilizada por esse subsistema não obedecia ao especificado pelas interfaces `IMediaTimescaleFacade` e `ElementaryStreamFacade`. Para resolver o impasse, foi implementada uma classe para adaptar[11] a interface oferecida à interface esperada pela ferramenta de ajuste. A classe foi chamada `VideoTimescaleAdapter` e está apresentada na Figura 65.



[11] A semântica do termo *adapter* neste trabalho é análoga ao descrito no padrão de projeto Adapter

Figura 65 - Classe para adaptar serviços oferecidos pelo subsistema de ajuste de vídeo à ferramenta de ajuste elástico.

Além dessa classe, outras transformações elaboradas no subsistema de ajuste de vídeo foram a inclusão do métodos `addTimescaleListener` e `addIntantListener` e a transformação de algumas classes para utilizar/implementar classes/interfaces genéricas da ferramenta de ajuste (como `IAssembler` e `IDisassembler`). Entretando, atualmente o subsistema de ajuste de vídeo ainda não está pronto para processar um fluxo elementar constituinte de um fluxo de sistema. Duas modificações ainda necessárias são alterar o algoritmo de ajuste para funcionar em tempo de execução e implementar alguns métodos do `ElementaryStreamFacade` juntamente com classes similares as da Subseção 4.4.5.

5
Integração da Ferramenta de Ajuste com Exibidores de Conteúdo

Conforme explicado no Capítulo 4, a ferramenta de ajuste processa dados de áudio para futuro armazenamento ou disponibilização do fluxo processado diretamente para uma outra aplicação. Deve existir um mecanismo de comunicação capaz de integrar exibidores de conteúdo com a ferramenta de ajuste, seja utilizando um arquivo gerado ou recebendo o fluxo de bytes produzido. Este Capítulo descreve a implementação realizada com o objetivo de integrar o algoritmo de ajuste elástico a diferentes exibidores de conteúdo. A Figura 66 ilustra as classes principais para esse propósito.

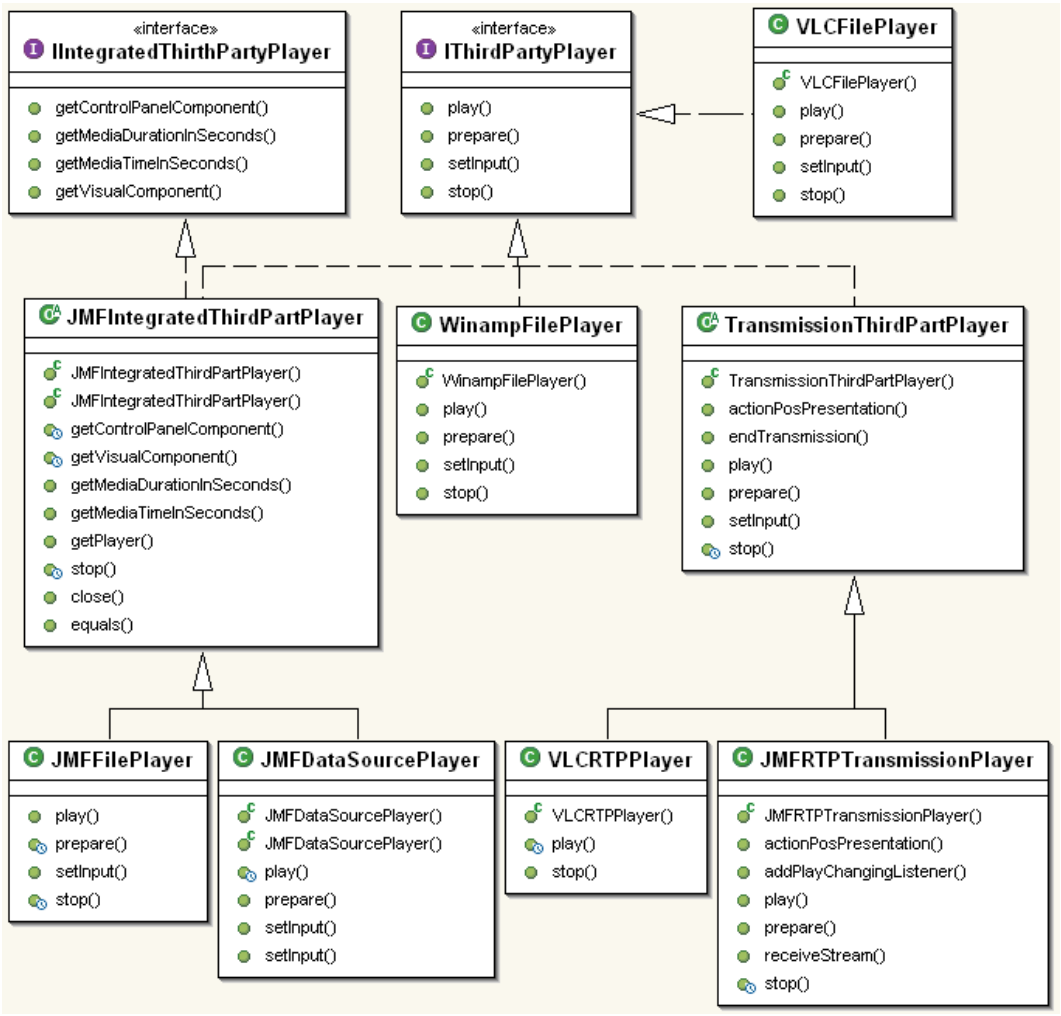


Figura 66 - Diagrama das classes dos exibidores de conteúdo.

Todas as classes que realizam a comunicação da ferramenta de ajuste com exibidor implementam a interface `IThirdPartyPlayer`. As demais classes da Figura 66 serão explicadas nas próximas subseções, que detalham o mecanismo de comunicação utilizado quando ajuste ocorre em tempo de compilação e de execução.

5.1. Mecanismo de Comunicação utilizado com Ajuste Elástico em Tempo de Compilação

Quando o ajuste é realizado em tempo de compilação, a integração com o exibidor de conteúdo é realizada utilizando a URI do arquivo contendo a mídia processada. As classes `JMFFilePlayer`, `VLCFilePlayer` e `WinampFilePlayer` foram desenvolvidas para prover a comunicação em tempo compilação com os exibidores JMF (Sun, 1999), VLC (Videolan, 2005) e Winamp (Nullsoft, 2005), respectivamente.

A classe `JMFFilePlayer` instancia um objeto do exibidor de conteúdo do JMF utilizando a URL do arquivo ajustado. O exibidor do JMF é capaz de reproduzir arquivos nos formatos MP2, MP3, MPEG-1/2 vídeo e sistemas no sistema operacional Windows. Já no sistema operacional Linux, o JMF somente oferece suporte à reprodução de arquivos MP2. As solicitações ao JMF são realizadas via programação Java. Para o JMF fornecer suporte a outros formatos de mídia, é possível acrescentar *plug-ins* capazes de manipular tais formatos. Por exemplo, existe um *plug-in* da empresa SUN (Sun, 1999) que fornece suporte a arquivos MP3 no Linux. Este trabalho utilizou o *plug-in* Jffmpeg (Jffmpeg, 2005), que acrescenta ao JMF a capacidade de manipular vários arquivos e fluxos de mídia, como arquivos MP3 e MPEG-1 vídeo e sistemas no Linux. Grande parte das funções do Jffmpeg é resultado da tradução de funções do conhecido codificador Ffmpeg (Ffmpeg, 2005) (escrito em linguagem C), utilizando JNI (*Java Native Interface*).

O VLC é um exibidor de conteúdo gratuito que oferece suporte a diversos formatos de arquivos, incluindo todos os formatos manipulados por este trabalho. A interação com o VLC para solicitar a exibição de um arquivo foi realizada pela classe `VLCFilePlayer` via um processo externo. O exibidor Winamp é capaz de

reproduzir todos os formatos manipulados por este trabalho, exceto o AC-3. Para interagir com o Winamp, a classe `WinampFilePlayer` realiza uma chamada via um aplicativo chamado CLAmp.exe[12], que interpreta um comando escrito e envia a ação correspondente a esse exibidor de conteúdo.

5.2. Mecanismo de Comunicação utilizado com Ajuste Elástico em Tempo de Execução

As classes que interagem com exibidores em tempo de execução recebem um fluxo de dados e solicitam imediatamente sua reprodução. Dois modos diferentes para entregar o fluxo ao exibidor de conteúdo foram utilizados: via programação e via RTP.

A interação via programação foi obtida através da classe `JMFDataSourcePlayer`, que envia dados de mídia ajustados para um exibidor de conteúdo do JMF. Nessa interação, algumas classes precisaram ser criadas para definir uma fonte de dados, a ser alimentada pelo algoritmo de ajuste, com a interface que o exibidor de conteúdo espera para buscar a mídia, conforme definido pelo padrão JMF. As classes implementadas para esse fim são ilustradas na Figura 67.

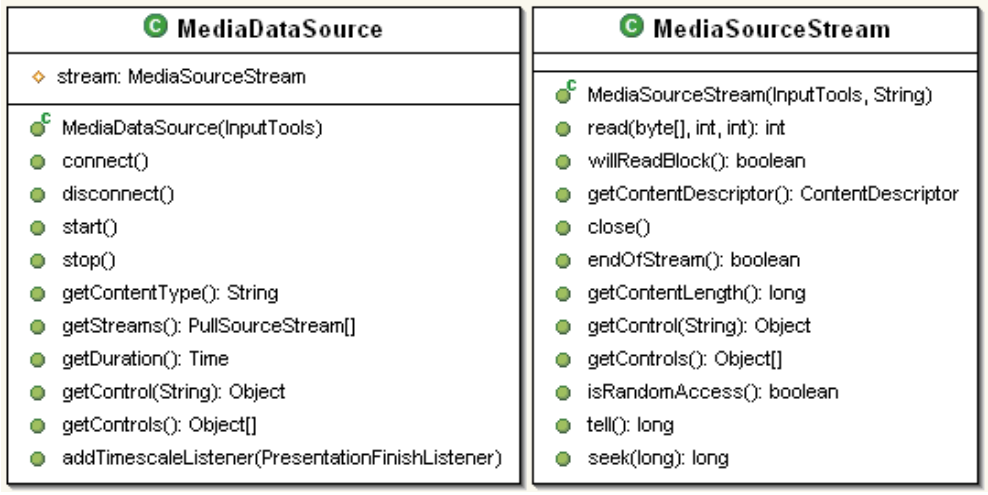


Figura 67 - Diagrama das classes da integração via programação com JMF.

As classes `MediaDataSource` e `MediaSourceStream` foram desenvolvidas para buscar dados de mídia em um buffer alimentado pelo algoritmo de ajuste elástico e fornecer a interface de uma fonte de dados definida pela especificação

[12] Disponível no próprio *site* do Winamp, ver .

JMF. A classe `MediaDataSource` herda das classes `DataSource`, `PullDataSource`, enquanto a classe `MediaSourceStream` herda de `SourceStream` e `PullSourceStream`. As classes herdadas são todas definidas pela biblioteca JMF. Originalmente, o exibidor de conteúdo JMF reproduz apenas fluxos de formato MP2 (Windows e Linux) e MPEG-1 vídeo (apenas no Windows). Entretanto, a adição da biblioteca Jffmpeg tornou possível fornecer suporte aos formatos MP3 e MPEG-1 vídeo (também no Linux). Vale destacar que o suporte de JMF a fluxos de bytes é diferente do suporte a arquivos.

Para realizar a comunicação via programação com o VLC é necessário utilizar JNI uma vez que esse exibidor está escrito em linguagem C++. O desenvolvimento dessa comunicação foi delegado para trabalho futuro.

Na comunicação com exibidores via RTP, os dados de mídia são enviados utilizando o protocolo RTP (Perkins, 2003). O transmissor escreve o fluxo de mídia numa porta e solicita que o exibidor de conteúdo escute essa porta. Os exibidores de conteúdo integrados foram o JMF e o VLC e a biblioteca para realizar RTP utilizada foi a JMF.

A biblioteca RTP do JMF permite apenas transmitir dados do MPEG-1 áudio formato MP2 e MPEG-1 vídeo e sistemas. Assim, embora o exibidor VLC suporte todos os formatos de arquivos manipulados por esse trabalho, só é possível exibir, utilizando o JMF para transmitir RTP, os formatos MP2, MP3 e MPEG-1. Entretanto, ainda é possível acrescentar novos *plug-ins* ao JMF para que este suporte novos formatos, tanto para transmissão via RTP quanto para reprodução. Por exemplo, quando integrado ao Jffmpeg, essa solução permite manipular também dados de MPEG-2 MP2 e MPEG-1/2 MP3.

A classe `TransmissionThirdPartPlayer` contém características comuns das classes `JMFRTPTransmissionThirdPartPlayer` e `VLCRTPPlayer`. Dentre outras atribuições, a classe `TransmissionThirdPartPlayer` é responsável por interagir com um servidor RTP para enviar os dados. A classe `VLCRTPPlayer` escuta a transmissão enviando um comando via um processo externa e a classe `JMFRTPTransmissionThirdPartPlayer` interage com uma classe cliente RTP para buscar os dados programaticamente. O transmissor e receptor RTP são implementados pelas classes `RTPServer` e `RTPCliente`, respectivamente.

5.3. Considerações Adicionais

A comunicação com exibidores de conteúdo pode ser também classificada como realizada de forma integrada ou não. Uma comunicação integrada possui duas vantagens principais: o fato do exibidor poder fornecer informações sobre o estado atual da apresentação da mídia (como, por exemplo, o instante atual da apresentação da mídia) e a possibilidade de posicionar espacialmente os exibidores de conteúdo dentro da interface gráfica da aplicação cliente.

As classes que realizam comunicação de modo integrado com exibidor de conteúdo implementam a interface `IIntegratedThirdPartPlayer`. O único exibidor de conteúdo cuja interação pode ser realizada de modo integrado é o JMF, via arquivo (tempo de compilação) e programação (tempo de execução). A classe `JMFIntegratedThirdPartPlayer` implementa os serviços definidos por essa interface e também agrupa um conjunto de características comuns entre `JMFFilePlayer` e `JMFDataSourcePlayer`.

Por fim, vale novamente ressaltar que atualmente estão disponíveis codificadores da SUN para poucos formatos de mídias. Para exibir formatos não suportados pelo JMF, as ferramentas VLC e Winamp podem ser utilizadas. Contudo, essas ferramentas executam de forma não-integrada com a aplicação cliente. O desenvolvimento de uma comunicação via programação com o VLC, já mencionado como trabalho futuro, pode permitir uma comunicação integrada com suporte a vários formatos.

6
Utilização da Ferramenta de Ajuste em Sistemas Hipermídia

Este capítulo primeiramente descreve as necessidades de adaptabilidade de sistemas hipermídia, para depois apresentar como a ferramenta de ajuste pode ser utilizada nesses sistemas e, enfim, como foi realizada a integração da ferramenta com sistema HyperProp.

6.1.
Necessidades de Adaptabilidade de Sistemas Hipermídia

Segundo Rodrigues (Rodrigues, 2003), mecanismos de adaptação em sistemas hipermídia são transformações necessárias para tornar os hiperdocumentos consistentes (espacialmente e temporalmente) ou adequá-los a um contexto de exibição (plataformas, usuários etc.). Adaptações podem ser necessárias por causa da flexibilidade na descrição dos documentos; pela própria navegação do usuário, gerando eventos que não puderam ser previstos antes de iniciar a exibição do documento; por atrasos, também impossíveis de serem previstos no sistema operacional e na rede; ou por quaisquer outras informações provenientes do contexto de exibição (recursos de hardware disponíveis na máquina do cliente, preferências e perfil do usuário etc.).

Um sistema hipermídia pode ser subdividido em diferentes ambientes e todos eles podem ou não suportar mecanismos de adaptação. O ambiente de autoria é onde os autores criam os hiperdocumentos. A autoria é realizada utilizando algum modo de especificação, como, por exemplo, uma linguagem declarativa, para descrever os objetos de mídia do documento, suas propriedades e relacionamentos. Além disso, o autor deve especificar características de exibição para cada objeto de mídia, entre as quais, pode-se destacar, sua duração esperada de apresentação. Tais características devem poder ser especificadas de modo

flexível, utilizando, por exemplo, diferentes alternativas de exibição, como explicado mais adiante.

Adaptações, como ajuste elástico, podem ser realizadas nesse ambiente, normalmente em tempo de compilação, com objetivo de manter a consistência temporal de um documento, desde sua especificação, e evitar que seja feito em tempo de exibição no cliente. Por exemplo, em um sistema de TV pode-se desejar adaptar o documento no transmissor para evitar processá-lo em todos os clientes.

O ambiente de armazenamento oferece mecanismos para gravar, recuperar e adaptar os documentos. É possível separar o armazenamento em duas classes, que apresentam requisitos distintos e podem operar de maneira independente: o armazenamento da descrição dos documentos hipermídia e o armazenamento do conteúdo dos objetos de mídia (arquivos de texto, imagem, áudio, vídeo etc.). Dentre as funções do ambiente de armazenamento, vale detalhar aqui a possibilidade de adaptação. O principal objetivo da adaptação localizada no ambiente de armazenamento é que o mesmo forneça conteúdos com configurações que melhor atendam ao contexto em que eles venham a ser exibidos, buscando ao mesmo tempo otimizar a utilização dos recursos que estejam situados no servidor (ambiente de armazenamento), no caminho até o cliente hipermídia (provedor de serviços de comunicação) e no próprio cliente (ambiente de execução).

O último ambiente é o de execução, também conhecido como ambiente de apresentação. Esse ambiente normalmente encontra-se situado junto ao cliente hipermídia e deve conter os mecanismos para interagir com os usuários e controlar a exibição das mídias de tal forma que as intenções especificadas pelos autores sejam respeitadas. Entre os elementos que compõem o ambiente de execução podem ser destacados: as ferramentas de exibição e o formatador hipermídia.

As ferramentas de exibição correspondem aos módulos, de software ou hardware, responsáveis por lidar com os dispositivos de entrada e saída e controlar a exibição do conteúdo dos objetos, cuidando, por exemplo, de questões de descompressão, tratamento de interações do usuário, controle da mudança de velocidade de exibição para o caso de objetos de mídia contínua etc.

Formatador hipermídia é o nome dado ao elemento que reúne as funções para o controle da apresentação previamente especificada pelo autor. A Figura 68

ilustra a comunicação de um formatador hipermídia com uma ferramenta de exibição através de uma interface padronizada de comunicação entre eles (API FF), conforme proposto em Rodrigues (Rodrigues, 2003). A ferramenta de exibição recebe eventos de apresentação do formatador (como *play*, *stop* e *resume*) e a mídia a ser exibida, que pode vir como um arquivo ou fluxo de dados. A ferramenta de exibição é responsável por exibir a apresentação hipermídia para o usuário e repassar para o formatador eventos resultantes da interação do usuário com a apresentação e eventos resultantes da apresentação de um determinado trecho do conteúdo. Os exibidores de conteúdo que não obedecem à API FF podem interagir com o formatador através de um adaptador, que mapeia as funções da API FF para as funções do exibidor de conteúdo.

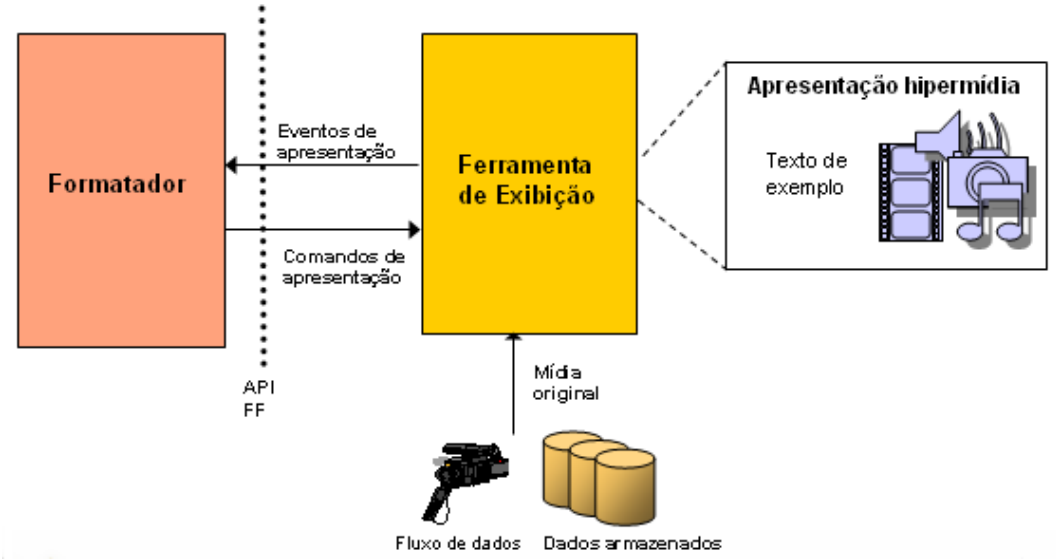


Figura 68 - Integração de um formatador hipermídia com uma ferramenta de exibição.

Para orquestrar a apresentação, o formatador precisa construir um *plano de execução* que contém as informações de sincronismo entre todas as entidades do documento, incluindo aquelas relações que não podem ser previstas, como interação com usuário (Bachelet et al., 2004). A fim de manter consistente uma apresentação, muitas vezes o formatador precisará ajustar o hiperdocumento (estrutura ou a forma de apresentação), ou requisitar que as ferramentas de exibição façam o ajuste do conteúdo. A forma mais comum de adaptar objetos de mídia é possibilitar a escolha de partes do documento baseando-se em diferentes alternativas de conteúdo. Com isso, os objetos do documento passam a poder ter não apenas os seus conteúdos, mas também os seus aspectos de exibição adaptáveis.

Uma adaptação pode ser realizada em tempo de compilação, durante a construção do plano de execução. Nessa situação, as técnicas de adaptação podem considerar as preferências do usuário e as características da plataforma de exibição. A vantagem dessas técnicas é a diminuição do número de adaptações uma vez iniciada uma apresentação.

Infelizmente, as adaptações em tempo de compilação não são suficientes para garantir uma boa apresentação. Existem situações em que é necessário utilizar adaptações em tempo de exibição, que modificam o já criado plano de execução, em função de algum evento imprevisível. As técnicas de adaptação em tempo de exibição podem considerar as preferências do usuário e as características da plataforma de exibição (como as técnicas de tempo de compilação) e outras características mais dinâmicas, como interação do usuário.

6.2. Utilização da Ferramenta de Ajuste em Sistema Hipermídia

O ajuste elástico atua na apresentação dos objetos, aumentado ou diminuindo suas durações em relação aos valores originais (ou ideais), com o objetivo de garantir a consistência temporal do documento, ou seja, respeitar as restrições temporais entre objetos de mídia impostas pelo autor[13]. Quando o formato da mídia a ser apresentado é textual, ou corresponde a imagens estáticas, o ajuste da duração de exibição é trivial. No entanto, se um objeto de mídia contínua precisa ter sua duração corrigida, o ideal é que esse ajuste ocorra sem que seja prejudicada a qualidade da apresentação.

A Figura 69 ilustra a inclusão da ferramenta de ajuste, definida no Capítulo 4, dentro de uma ferramenta de exibição de um sistema hipermídia. Nesse cenário, a ferramenta de ajuste é formada por três aplicações diferentes: o adaptador, a ferramenta de ajuste elástico e o exibidor de conteúdo. O formatador interage com o adaptador de modo análogo ao previamente ilustrado na Figura 68. O adaptador é uma aplicação que faz uso da ferramenta de ajuste e do exibidor de conteúdo para fornecer uma interface de uma ferramenta de exibição com capacidade de realizar ajuste elástico. A ferramenta de ajuste não foi modificada e

[13] É importante observar que os conceitos em que se baseia o ajuste elástico temporal podem ser transpostos para o arranjo espacial dos objetos, resultando em um ajuste elástico das dimensões dos objetos cujos conteúdos são mídias visíveis.

é acessada através das APIs definidas na Seção 4.1 e o exibidor de conteúdo não depende da ferramenta de ajuste elástico nem do adaptador.

O adaptador recebe do formatador comandos de apresentação, que incluem a mídia a ser exibida, executa algumas operações (como cálculo do fator de ajuste necessário) e solicita serviços à ferramenta de ajuste elástico (como também pode se comunicar diretamente com o exibidor de conteúdo). A ferramenta de ajuste, por sua vez, passa para o exibidor de conteúdo comandos de apresentação (como *start* e *stop*) e o conteúdo processado da mídia a ser exibida. Enfim, o exibidor de conteúdo apresenta a mídia e retorna eventos de apresentação para a ferramenta de ajuste (como, por exemplo, a informação que a apresentação terminou) e para o adaptador, que os repassa para o formatador. Por fim, a ferramenta de ajuste pode levantar aqueles eventos de apresentação definidos nas APIs de ajuste (Seção 4.1).

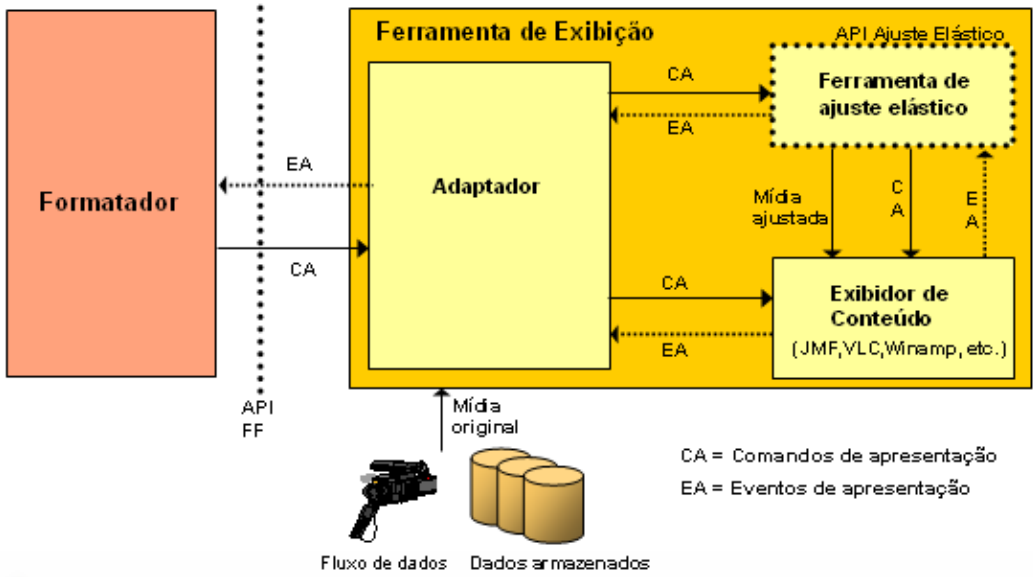


Figura 69 - Integração de uma ferramenta de exibição com ajuste elástico com um formatador hipermídia.

Três tipos de adaptadores de exibidor de conteúdo foram definidos. O *FlexMediaFileAdapter* utiliza a ferramenta de ajuste para gerar um arquivo temporário com a mídia processada e o envia para um exibidor de conteúdo. Como conseqüência, esse adaptador é mais adequado para executar ajustes em tempo de compilação.

Para permitir realizar ajuste em tempo de exibição, dois outros adaptadores foram implementados: *FlexMediaRTPAdapter* e *FlexMediaDataSourceAdapter*. Os dois adaptadores lêem os dados de mídia de um fluxo de dados, realizam o

ajuste elástico e o enviam como um novo fluxo ao exibidor de conteúdo. A diferença entre esses adaptadores é que o *FlexMediaRTPAdapter* transmite e recebe os dados da mídia utilizando RTP enquanto o *FlexMediaDataSourceAdapter* os envia utilizando um objeto Java. O Apêndice B ilustra uma aplicação desenvolvida para enviar dados de mídia, recebê-los e solicitar ajuste em tempo de exibição. Essa ferramenta pode ser utilizada por aplicações clientes da ferramenta de ajuste.

6.3. Integração da Ferramenta de Ajuste com o Sistema HyperProp

O formatador HyperProp inclui várias funcionalidades para orquestrar apresentações hipermídia, dentre elas destacam-se a adaptação de documentos em tempo de compilação e de execução, estratégias de antecipação das ações de exibição (mecanismos de pré-busca), mecanismos de integração com exibidores de mídia e de integração com a infra-estrutura de exibição (em especial, sistema operacional e rede).

O adaptador de exibidor de conteúdo é responsável por traduzir os comandos recebidos do formatador pela API da ferramenta de exibição do HyperProp (Rodrigues, 2003) (API FF, ver Figura 69) para as APIs da ferramenta de ajuste elástico (ver Seção 4.1). A Tabela 6, abaixo, ilustra o mapeamento dos métodos entre a API FF e a API de ajuste elástico em tempo de compilação desenvolvida nesta dissertação.

API FF	**API de compilação de ajuste elástico**
`prepare`	`config,addTimescaleListener, getTimescaleInstant,start`
`start`	(mecanismo de comunicação com exibidor de conteúdo)
`stop`	(mecanismo de comunicação com exibidor de conteúdo)
`getAttributeValue`	`getOutputTools,getReport`

Tabela 6 - Mapeamento dos métodos da API FF com os da API de ajuste elástico em tempo de compilação.

O ajuste em tempo de compilação deve ocorrer durante a preparação do adaptador. Por isso, o método `prepare` da API FF deverá chamar os métodos `config` e `start` da ferramenta de ajuste. Na preparação, o adaptador é informado da duração implícita da mídia a ser exibida e da sua duração esperada. A duração implícita do objeto pode já estar especificada ou então ser calculada (por exemplo, através da chamada `getDuration` oferecida pelo *player* JMF). A duração

esperada foi calculada pelo algoritmo de otimização do formatador HyperProp. O adaptador calcula o fator de ajuste a ser aplicado através da divisão da duração esperada pela implícita. No caso de não ser possível determinar a duração implícita da mídia *a priori*, por exemplo, para fluxos de dados ao vivo, o fator de ajuste deve ser explicitamente informado ao adaptador. Para tal, pode-se acrescentar um parâmetro ao conjunto de parâmetros de exibição da mídia passados pelo formatador e acrescentar a lógica de tratar esse novo parâmetro no adaptador[14].

Caso seja desejado que o ajuste aconteça de modo assíncrono (isto é, a ferramenta de ajuste deva executar em uma nova *thread*), o adaptador deverá se registrar como observador do evento de finalização do processamento de ajuste, através do método `addTimescaleListener`, pois a sua preparação só pode terminar depois que o processamento de ajuste já estiver terminado.

Ainda durante a preparação do adaptador, ao final do processamento de ajuste, o adaptador pode desejar descobrir novos valores para determinados instantes de tempo, o que é possível através do método `getTimescaleInstant`. Esse método é particularmente importante para manter a coerência da apresentação hipermídia. A justificativa para isso é que o modelo conceitual do formatador HyperProp prevê que um autor crie âncoras temporais internas especificadas através de intervalos cujas extremidades são tempos relativos ao início da apresentação do objeto inteiro. Logo, uma vez ajustada a duração de uma mídia, os intervalos das âncoras também precisam ser corrigidos.

Os métodos `start` e `stop` não são mapeados para métodos da API FF, mas sim para a classe `IThirdPartPlayer` de integração com exibidor de conteúdo apresentada no Capítulo 5. O método `getAttributeValue` da API FF pode ser usado para recuperar qualquer atributo do adaptador e, por isso, pode ser mapeado para todos os métodos `get` das APIs de ajuste de seu interesse. São eles: `getOutputTools` e `getReport`.

De maneira similar à tabela anterior, a Tabela 7 ilustra o mapeamento dos métodos entre a API FF e a API de ajuste elástico em tempo de execução.

---

[14] Vale ressaltar que, no estágio atual de desenvolvimento, o algoritmo de otimização do formatador HyperProp que calcula as durações esperadas para todas as mídia do documento não

API FF	API ajuste elástico
`Prepare`	`config,addTimescaleInstantListener`
`Start`	`start`
`Stop`	`stop`
`setAttributeValue`	`setFactor`
`getAttributeValue`	`getOutputTools,getReport`

Tabela 7 - Mapeamento dos métodos da API FF com os da API de ajuste elástico em tempo de execução.

Nesse caso, o ajuste é configurado durante a preparação do adaptador e, conseqüentemente, o método `prepare` da API FF deve chamar o método `config` da ferramenta. Ainda durante a preparação, o adaptador pode chamar o método `addTimescaleInstantListener` para cadastrar-se como observador do evento de descoberta dos novos valores de determinados instantes de tempo (indicados pelo método `config`). Ademais, a execução do ajuste é realizada paralelamente a apresentação da mídia, o que implica que os métodos `start` e `stop` podem ser diretamente mapeados (além de também precisar se mapeado para métodos da classe `IThirdPartPlayer` de integração com exibidor de conteúdo).

O método `setAttributeValue` da API FF pode ser usado para modificar qualquer atributo do adaptador, em particular, o fator de ajuste utilizado, efetuando-se, assim, o mapeamento com o método `setFactor`. Essa característica é particularmente importante caso o formatador perceba que eventos imprevisíveis modificaram os cálculos previamente efetuados na elaboração do plano de execução. Por fim, o mapeamento do método `getAttributeValue` é análogo ao descrito para a API de compilação.

Somente o exibidor de conteúdo JMF foi testado na comunicação com o formatador. As classes utilizadas foram `JMFFilePlayer` e `JMFDataSourcePlayer` gerando respectivamente os adaptadores *FlexMediaFileAdapter* e *FlexMediaDataSourceAdapter*. Considerando que esses adaptadores desenvolvidos podem ser executados de modo integrado (ver Capítulo 5), é possível monitorar os eventos de apresentação e notificá-los ao formatador. O adaptador é responsável por traduzir o modelo de eventos do JMF no modelo de eventos do formatador e vice-versa. Desse modo, o formatador pode sincronizar a apresentação de outras mídias (por exemplo, um navegador HTML, um

---

está preparado para receber objetos com durações indeterminadas.

visualizador de JPEG etc) a partir dos eventos de apresentação recebidos pelo adaptador do JMF. Além disso, também é possível integrar espacialmente a exibição do conteúdo ajustado com outras mídias. Por exemplo, uma janela do sistema operacional pode conter diversos objetos diferentes (texto de legenda, vídeo, imagens etc), escondendo do usuário a existência de vários exibidores diferentes.

Para exibir formatos não suportados pelo JMF, as ferramentas VLC e Winamp podem ser facilmente utilizadas numa comunicação não-integrada. Atualmente, o VLC pode ser utilizado em um *FlexMediaFileAdapter* e *FlexMediaRTPAdapter* e o Winamp em um *FlexMediaFileAdapter*. Conforme já discutido no Capítulo 5, o desenvolvimento de uma comunicação via programação com o VLC pode permitir uma comunicação integrada com suporte a vários formatos em um adaptador *FlexMediaDataSourceAdapter*.

O Apêndice C ilustra um exemplo de hiperdocumento que precisa de ajuste elástico para ser corretamente apresentado pelo formatador HyperProp.

7
Resultados Obtidos

Medidas de qualidade objetivas e subjetivas foram extraídas do algoritmo de ajuste de áudio e comparadas com os resultados obtidos com o algoritmo *Music 01* do *Sound Forge 8.0*. O *Sound Forge 8.0* foi escolhido como ferramenta de comparação dentre as enumeradas nos trabalhos relacionados pelos seguintes motivos: a aplicação dos seus algoritmos de ajuste gera um arquivo com o áudio ajustado e comprimido, trata-se de uma ferramenta bastante utilizada atualmente, inclusive em aplicações comerciais de processamento de áudio, e foi acessível durante a realização da pesquisa. Dentre os algoritmos do *Sound Forge*, alguns são específicos para manter determinadas características do áudio. O algoritmo *Music 01* foi escolhido por ter propósito mais geral (preservar a qualidade do áudio como um todo) e ser opção de ajuste *default* da ferramenta.

Todos os áudios utilizados nos testes deste capítulo foram codificados no formato MPEG-1 MP3 *stereo*, *44.1*kHz e *128*kbps. Esses parâmetros foram escolhidos por serem atualmente tipicamente utilizados durante codificação de áudio. Foram utilizados *5* tipos de áudio de curta duração (com duração de aproximadamente *8* segundos) e *2* de longa duração (cerca de *8* minutos), listados na Tabela 8.

Nome	**Conteúdo**	**Duração (em seg)**
Castanholas	Som de instrumento de percussão.	7.05
Flauta	Som de três notas de instrumento melódico tocadas em seqüência. Cada nota dura cerca de um terço da duração total do arquivo.	7.37
Voz	Voz feminina em Português.	8.70
Música	Música de nome “Águas de Março” de Tom Jobim, contendo voz feminina e masculina.	7.37
Filme	Trecho do áudio do filme “Matrix” com som de fundo e duas vozes.	8.93
Voz	Voz masculina em Português.	484

Música	Música "Gabriela" de Tom Jobim, contendo voz masculina e feminina.	482

Tabela 8 - Tipos de áudio utilizados nos teste de qualidade[15].

O Apêndice D ilustra uma ferramenta com interface gráfica que pode ser utilizada para solicitar processamento da ferramenta de ajuste. Essa aplicação foi utilizada para processar os áudios listados na Tabela 8 utilizando o algoritmo de ajuste proposto para áudio neste trabalho, que chamaremos de HyperAudioScale.

As seções a seguir detalham as medidas subjetivas e objetivas do algoritmo proposto. Por fim, também é apresentada uma análise da implementação da ferramenta de ajuste.

7.1. Medidas de Qualidade Subjetiva

As subseções a seguir descrevem primeiramente como o experimento foi realizado e depois apresenta os resultados obtidos.

7.1.1. Descrição do experimento

A elaboração e realização do teste para medição da qualidade subjetiva foram baseadas no padrão ITU-T P.911 (ITU, 1998) e no trabalho de Winkler e Faller (Winkler & Faller, 2005). Foram utilizados os *5* áudios de curta duração da Tabela 8. A escolha dos áudios foi baseada nas categorias recomendadas pela ITU-T, ilustradas na Tabela 9.

Categoria	Descrição
I	Voz de um locutor
II	Voz de mais de um locutor
III	Voz + música de fundo
IV	Som de um único instrumento musical
V	Som de vários instrumentos musicais

Tabela 9 - Categorias de tipos de áudio recomendadas para testes subjetivos, definidas pela ITU-T - fonte:(ITU, 1998).

Durante o teste, cada áudio foi reproduzido *16* vezes: *8* vezes o original, *4* fluxos processados utilizando o algoritmo proposto, com fatores de ajuste de *0.90, 0.95, 1.05* e *1.10*, e *4* fluxos processados com o algoritmo *Music 01* do *Sound Forge 8.0* com os mesmos fatores. Portanto, cada participante ouviu *80* áudios. Os

[15] Alguns áudios da tabela foram extraídos de uma seleção da *Audio Engineering Society* .

áudios foram apresentados sem interrupção, em pares (o original seguido de um áudio processado a partir deste), em ordem aleatória, sendo determinístico apenas o fato de que o mesmo tipo de áudio não seria reproduzido novamente antes que dois outros tipos o fossem. Entre dois áudios diferentes foi introduzido um tempo de *5* segundos de silêncio para o ouvinte avaliar o áudio que acabou de ouvir. O tempo total do teste para cada ouvinte foi, então, de pouco mais de *17* minutos (*80*8 + 80*5*).

Segundo a ITU-T (ITU, 1998), os testes subjetivos devem envolver entre *6* e *40* pessoas, dependendo da complexidade do problema. Neste trabalho, *10* pessoas participaram do teste. Todas elas afirmaram possuir audição normal. Os testes foram realizados com fone de ouvido *Philips SBC HP250* e software de reprodução *Windows Media Player versão 9* (Microsoft, 2005).

A qualidade de cada áudio foi atribuída relativamente, considerando que a qualidade do áudio original era excelente. A escala de avaliação utilizada possui 9 níveis (ITU, 1998), sendo cinco principais (Péssimo, Ruim, Razoável, Bom e Excelente) e quatro intermediários. O ouvinte deveria marcar um 'X' na qualidade julgada em um dos *9* níveis. A Figura 70 ilustra a escala adotada e a Tabela 10 resume a interpretação dos níveis principais de qualidade. Os níveis intermediários deveriam ser utilizados quando o ouvinte estivesse em dúvida entre dois níveis principais.

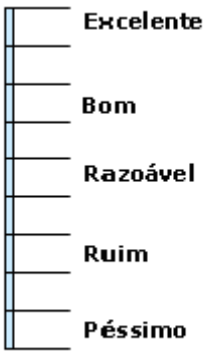


Figura 70 - Escala de votação utilizada para avaliar cada áudio no teste subjetivo.

Nível de qualidade	**Nível de degradação**
Excelente (muito boa)	Imperceptível
Bom	Pouco perceptível, mas não incômoda
Razoável	Perceptível e levemente incômoda
Ruim	Incômoda, mas não intolerável
Péssimo (muito ruim)	Muito incômoda (intolerável)

Tabela 10 - Interpretação de cada nível de qualidade.

Os ouvintes foram instruídos que alguns áudios poderiam ter a duração acelerada ou retardada, mas que a nota de qualidade por causa desse fator só deveria ser diminuída caso a velocidade de apresentação fosse incômoda. Além disso, os ouvintes foram treinados antes da realização dos testes, com objetivo de: tomar conhecimento de alguns defeitos comuns a áudios ajustados; serem apresentados aos áudios utilizados no teste; e serem expostos a uma simulação do teste para esclarecerem quaisquer dúvidas. Durante o treinamento, cada pessoa ouviu um conjunto de *5* pares de áudios. Sendo um de cada tipo da Tabela 8, apresentados em ordem aleatória.

7.1.2. Resultados obtidos

A média das notas obtidas para cada tipo de áudio, fator e algoritmo de ajuste são apresentadas na Tabela 11. As duas primeiras linhas da tabela apresentam os títulos dos dados apresentados. As próximas quatro linhas da tabela apresentam as médias das notas atribuídas, chamada de MOS (*Mean Opinion Score*), para diferentes fatores de ajuste processados utilizando o algoritmo proposto. As linhas seguintes são interpretadas de maneira similar só que o algoritmo de ajuste utilizado é o do *Sound Forge*.

Algoritmo de ajuste	**Fator de ajuste**	**Tipo de áudio**					**Média por fator e algoritmo**
		Castanholas	**Flauta**	**Voz**	**Música**	**Filme**	
Proposto	0.90	7.1	7.3	7.9	8.3	8.1	7.7
	0.95	6.8	8.0	8.7	8.4	8.4	8.1
	1.05	8.1	8.6	6.2	8.1	7.3	7.7
	1.10	7.4	7.6	5.8	7.1	7.0	7.0
Sound Forge	0.90	8.3	8.8	8.9	8.1	8.6	8.5
	0.95	8.6	8.9	8.9	8.6	8.2	8.6
	1.05	7.8	8.7	8.5	7.8	8.2	8.2
	1.10	7.4	8.5	8.0	8.0	8.4	8.1

Tabela 11 - MOS obtido para cada tipo de áudio e fator de ajuste.

A Figura 71 e a Figura 72 ilustram graficamente as notas da Tabela 11, , juntamente com os intervalos de confiança das notas com coeficiente de *95%* (Jain, 1991). Nas figuras, o segmento perpendicular a cada intervalo de confiança é o MOS. A Figura 71 ilustra os valores para os áudios processados com fatores *0.90* e *0.95* e a Figura 72 com os fatores *1.05* e *1.10*. Os áudios processados pelo HyperAudioScale e pelo *Sound Forge* são legendados,

respectivamente, pelas iniciais “h” e “s”, seguidos do fator de ajuste (em %) utilizado.

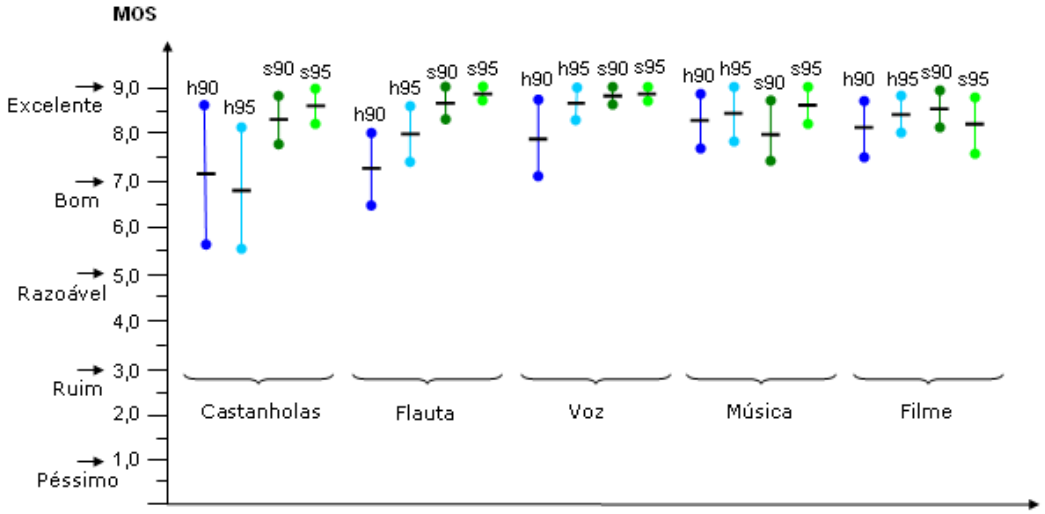


Figura 71 - MOS obtido para cada tipo de áudio com fator de ajuste 0.90 e 0.95 com coeficiente de confiança 95%.

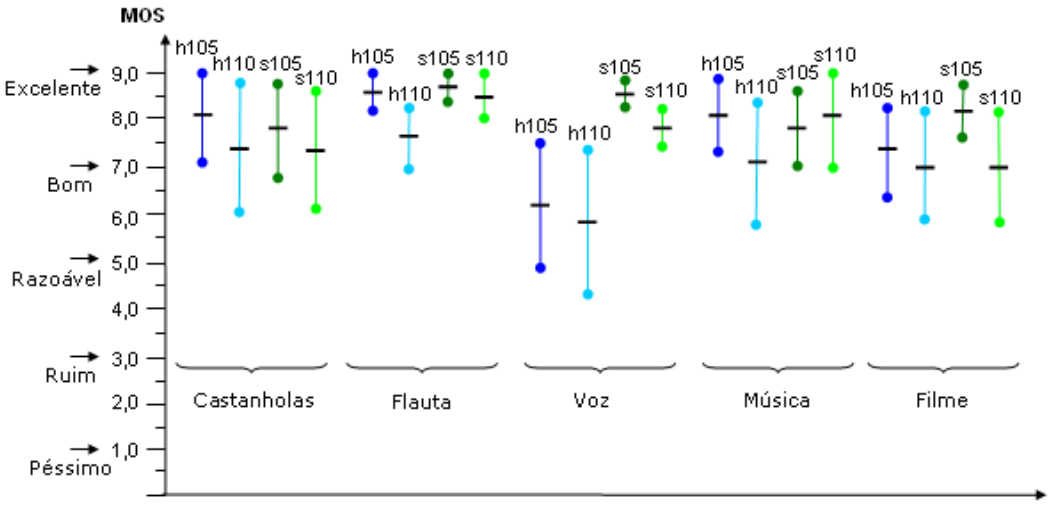


Figura 72 - MOS obtido para cada tipo de áudio com fator de ajuste 1.05 e 1.10 com coeficiente de confiança 95%.

Analisando os dados, pode-se notar que o algoritmo proposto gerou resultados bastante similares aos resultados do *Sound Forge* para a voz, música e filme utilizando fatores de *0.90* e *0.95*, e para castanholas utilizando fatores de *1.05* e *1.10*, sendo que seu pior resultado foi o áudio de voz com fator *1.05* e *1.1*. Mesmo para esses casos, o nível de aceitação está na faixa do razoável. De modo geral, pelo menos para o resultado deste teste, o algoritmo proposto se aproximou

do resultado do *Sound Forge*, tendo algumas vezes até obtido resultados melhores.

Levando-se em consideração as diferenças entre os algoritmos comparados (conforme discutido no Capítulo 2), como, por exemplo, o fato do algoritmo proposto funcionar em tempo de execução, e também as comparações de eficiência e precisão do algoritmo proposto, conforme será apresentado na próxima seção, o resultado do algoritmo proposto é bastante promissor, especialmente para aplicações em que o ajuste é aplicado em partes pequenas de áudio, como é o caso daquelas que usam o ajuste visando apenas a manutenção das relações de sincronismo.

Uma segunda análise dos dados visa comparar a qualidade atingida por diferentes tipos de áudio. Nesse momento, é importante considerar quais áudios utilizados no teste são mais bem-comportados, que significa conter um pequeno intervalo de freqüências num dado instante de tempo e possuir uma variação suave nas mudanças de freqüências. Os áudios de voz humana e flauta são mais bem-comportados do que os demais. O áudio das castanholas, embora tenha apenas o som de um instrumento musical, possui muito tempo com ruído branco, o que o torna mal comportado. Os áudios da música e filmes também possuem várias freqüências num dado momento.

A Tabela 11 ilustra as médias obtidas dos diferentes fatores de ajuste para cada tipo de arquivo. É esperado que a qualidade do algoritmo do *Sound Forge* seja melhor para áudios bem-comportados, pois ele pode tirar proveito das características do sinal para aplicar o ajuste. Desse modo, as maiores diferenças entre o algoritmo proposto e o *Sound Forge* ocorrem com o áudio da voz humana e flauta. Por outro lado, a diferença entre os resultados dos demais áudios foram menores.

Uma última análise dos dados mostra que a qualidade atribuída aos áudios mais acelerados foram ligeiramente maiores do que a dos menos acelerados utilizando a mesma variação do fator de ajuste. Ou seja, as médias para o fator *0.95* são maiores do que as do fator *1.05* e as médias para o fator *0.90* são maiores do que as do fator *1.10*. A primeira possibilidade para explicar essa afirmativa é que os algoritmos de ajuste funcionam melhor para acelerar do que retardar. No caso do algoritmo proposto, cada remoção de quadro acarreta uma descontinuidade e supressão de parte do sinal enquanto uma duplicação acarreta

duas descontinuidades e duplicação de parte do sinal. Outra possibilidade é que o ser humano prefere assistir uma apresentação com exibição mais rápida do que mais devagar, resultado esse também demonstrado por alguns estudos (Omoigui et al., 1999; Amir et al., 2000).

7.2. Medidas de Qualidade Objetivas

Algumas medidas objetivas de qualidade também foram extraídas do algoritmo proposto. Os testes realizados utilizaram os áudios da Tabela 8. Esta seção analisa a eficiência de processamento do algoritmo, a precisão da duração atingida no fluxo de áudio processado e a linearidade na escolha dos quadros a processar em fluxos de áudio com reservatório de bits. As medidas de eficiência e precisão são comparadas com o algoritmo *Music 01* do *Sound Forge*.

7.2.1. Eficiência

A ferramenta de ajuste proposta demora cerca de *10* segundos para processar qualquer um dos áudios longos (de aproximadamente *8* minutos) apresentados na Tabela 8 com fator *1.10*, utilizando uma máquina *Pentium 4 2.4GHz 1GB-RAM*, sem nenhum outro processo computacionalmente custoso sendo executado paralelamente. O *Sound Forge*, por sua vez, demora cerca de *54* segundos para as mesmas condições, sendo cerca de *27* segundos para abrir o arquivo, de *12* segundos para processá-lo e de *15* segundos para salvar o fluxo modificado em formato MP3. Vale destacar que os tempos medidos do *Sound Forge* não consideram o tempo que o usuário da ferramenta precisa para acessar as funcionalidades da ferramenta, somente o tempo utilizado pela ferramenta.

Sabe-se que a duração dos quadros de áudio pode ser calculada pela Figura 73. Desse modo, no caso deste teste, tem-se que a duração é de *26.12*ms *[=(1152/44.1)*10^{-3}]*.

$$Duração\_quadro = \frac{Total\ amostras\ do\ quadro}{taxa\_de\_amostragem\ do\ quadro}$$

Figura 73 - Cálculo da duração de um quadro do fluxo elementar.

Efetuando cálculos com base nos dados acima, o algoritmo proposto conseguiu atingir a taxa de *1849* quadros/segundo enquanto que para exibir o áudio é utilizada uma taxa de apenas *38.3* quadros/segundo.

Um último teste de eficiência foi realizado acrescentando metadados de sistemas MPEG-2 nos áudios MP3 anteriormente mencionados utilizando o algoritmo proposto. Com isso, o tempo total de processamento para os arquivos de *8* minutos de duração subiu de cerca de *10* para cerca de *20* segundos. Esse aumento do tempo de processamento se deve a necessidade de processar os dados do fluxo de sistema e instanciar *threads* que processam o áudio do fluxo elementar de trecho em trecho.

7.2.2. Precisão

A precisão dos algoritmos de ajuste elástico foi obtida através do cálculo da esperança da diferença entre a duração ótima e a atingida do fluxo, ajustado com fator *0.9* e *1.1*, utilizando os arquivos da Tabela 8. Para analisar esse parâmetro, é válido ressaltar que a duração dos fluxos de áudios divididos em quadros é obtida através da multiplicação do número de quadros no fluxo pela duração dos quadros. Desse modo, nenhum algoritmo de ajuste é capaz de sempre gerar a duração ótima, uma vez que a duração dos fluxos divididos em quadros é sempre múltiplo da duração dos quadros. No caso desse teste, os quadros possuem duração igual a *26.12*ms.

A Tabela 12 ilustra os resultados obtidos. Nos testes realizados, a precisão obtida em arquivos curtos com o HyperAudioScale foi menor do que um quadro, que é de *26.12*ms (com exceção de um caso por menos de 1ms), e de cerca de *2* quadros utilizando o *Sound Forge*. Para arquivos longos, os algoritmos obtiveram precisões iguais, menor do que um quadro.

Fator de ajuste aplicado	**Arquivo**	**E (Duração (ótima - atingida))**	
		Algoritmo proposto (ms)	**Algoritmo *Sound Forge* (ms)**
0.90	Castanholas	-5	-55
1.10	Castanholas	-5	-55
0.90	Flauta	23	-57
1.10	Flauta	-13	-43
0.90	Voz curta	20	-60
1.10	Voz curta	-20	-40
0.90	Música curta	23	-57
1.10	Música curta	7	-43
0.90	Filme	17	-63
1.10	Filme	-27	-47
0.90	**(MÉDIA CURTOS)**	**15.6**	**-58.4**
1.10	**(MÉDIA CURTOS)**	**-11.6**	**-45.6**
0.90	Voz longa	-14	-14
1.10	Voz longa	-16	-16

0.90	Música longa	-14	-14
1.10	Música longa	-16	-16
0.90	**(MÉDIA LONGOS)**	**-14**	**-14**
1.10	**(MÉDIA LONGOS)**	**-16**	**-16**

Tabela 12 - Precisão do processamento de arquivos MP3 (em milisegundos).

7.2.3. Linearidade

Conforme comentado na Seção 3.3, nem todos os quadros do fluxo de áudio podem ser escolhidos para processamento quando existe reservatório de bits. Isso significa que o algoritmo de ajuste elástico proposto pode ser não-linear em relação à escolha dos quadros a processar. Para medir a não-linearidade do algoritmo proposto para áudios com reservatório de bits, foi definida uma variável aleatória *X* que conta o número de quadros entre dois quadros processados.

A Tabela 13 ilustra a esperança e o desvio padrão dessa variável aleatória estimados com base nos arquivos da Tabela 8. É esperado que essa medida tenha piores resultados quando o número de quadros a serem processados cresce. Sendo assim, os fatores escolhidos para esse teste foram *0.9* e *1.1*. Com esses fatores, espera-se obter *X=10*.

Fator de ajuste aplicado	**Arquivo**	**X= Número de quadros entre dois quadros processados**	
		Esperança	**Desvio padrão**
0.90	Castanholas	9.96	5.34
1.10	Castanholas	10.00	0.54
0.90	Flauta	9.96	0.32
1.10	Flauta	10.0	0.21
0.90	Voz curta	9.96	5.34
1.10	Voz curta	10.00	0.53
0.90	Música curta	10.00	0.50
1.10	Música curta	10.00	0.52
0.90	Filme	9.70	5.56
1.10	Filme	10.0	0.24
0.90	Voz longa	10.0	2.21
1.10	Voz longa	10.0	0.44
0.90	Música longa	10.0	0.30
1.10	Música longa	10.0	0.25

Tabela 13 - Medida de não-linearidade do processamento de arquivos MP3.

Analisando a Tabela 13, verifica-se que a esperança medida de *X* de fato encontra-se bem próxima (ou igual) a *10*. Alguns arquivos de curta duração não conseguiram obter a média de *10.00* com precisão de duas casas decimais, o que se refletiu na precisão medida na subseção anterior, entretanto o valor foi atingido por arquivos maiores.

Os valores de desvio padrão são pequenos. O maior valor atingido foi *5.56*, no caso do áudio do filme e ainda esse valor não é ruim, pois é aproximadamente metade do valor da média. Isso significa que não existem muitos casos em que dois quadros simultâneos ou muito próximos são processados. Conclui-se, assim, que a não-linearidade do algoritmo proposto não é muito significativa, pelo menos nos áudios testados. Isso é interessante porque implica que não ocorrem mudanças significativas do fator de ajuste aplicado em diferentes partes do fluxo de áudio.

Vale observar que o desvio padrão é, em geral, maior para fatores de *0.9* do que de *1.1*. Esse resultado é esperado uma vez que, em fluxos de áudio com reservatório de bits, devem existir mais quadros com unidades lógicas menores do que o campo físico do que o contrário.

7.3. Análise da Implementação

A implementação atual da ferramenta de ajuste está quase finalizada, sendo sua principal pendência o fato de não estar plenamente integrada com o algoritmo de ajuste de vídeo, como mencionado na Seção 4.5. Esta subseção analisa a extensibilidade do código fonte para novos formatos de mídia e o retardo introduzido pelo algoritmo de ajuste da ferramenta.

7.3.1. Extensibilidade

O subsistema de ajuste de áudio atualmente oferece suporte a vários formatos. Considerando que o processamento é realizado nas classes de formato genérico de áudio, a inclusão de um novo formato de áudio pode ser realizada simplesmente definindo um conjunto de classes que contêm as particularidades do formato, mais um *assembler* para construir tais classes a partir de dados binários, adicionado a um meio de reconhecer o novo formato para instanciar as classes corretas.

Por outro lado, os subsistemas de vídeo e sistemas foram desenvolvidos apenas para formatos MPEG-2, embora algum esforço tenha sido feito para abstrair as particularidades do formato. Para efetivamente generalizar tais subsistemas, é necessário estudar outros formatos de modo a coletar o que

realmente pode ser independente e o que deve ser instanciado para cada formato de mídia específico.

Ao total, a ferramenta de ajuste contém *172* classes, sendo que *49* são do algoritmo de áudio, *41* do de vídeo e *20* do de sistemas. *37* classes são genéricas para serem utilizadas por algoritmos de ajuste de qualquer mídia e *25* realizam a comunicação com exibidores de conteúdo.

7.3.2. Retardo Introduzido

O retardo inicial introduzido pela ferramenta de ajuste é definido como o tempo mínimo que é necessário acrescentar antes de iniciar a reprodução de um fluxo de mídia. Esse tempo deve ser suficiente para que a exibição da mídia possa ocorrer sem interrupção, precedida pela realização do ajuste elástico, que pode continuar ocorrendo em paralelo à exibição.

Sabe-se que o tempo total de processamento para realizar ajuste elástico em fluxos de sistemas é expresso pela fórmula abaixo (ver também Figura 59).

$$TTotal = TIniDemux + \sum_{trechos} TControlador + \sum_{trechos} Max\{AjusteFE_i\} + TFinalMux$$

Figura 74 - Cálculo do tempo total para ajustar fluxo de sistemas.

O *TIniDemux* é o tempo necessário para o demultiplexador encontrar e montar o primeiro PACKET do fluxo de sistemas. A partir daí, o demultiplexador sinaliza que o controlador pode efetivamente começar sua execução.

Os tempos *TControlador* e o $Max\{AjusteFE_i\}$ devem ser contabilizados para cada trecho da mídia original. *TControlador* é o tempo que o controlador de sistemas precisa para percorrer todos os fluxos elementares e configurar parâmetros para execução de seus processadores. Em cada trecho, os tempos necessários para realizar ajuste elástico em cada fluxo elementar podem ser diferentes. Uma vez que o controlador deverá esperar a finalização do processamento de todos os fluxos, o tempo a ser contabilizado é o do processamento do fluxo elementar mais demorado, dado por $Max\{AjusteFE_i\}$.

O multiplexador começa a executar logo que o controlador processa o primeiro PACKET do fluxo de mídia e continua até terminarem todos os PACKETs do fluxo de sistemas. O *TFinalMux* indica o tempo decorrente quando o controlador já terminou sua execução, juntamente com os processadores de

fluxos elementares, e o multiplexador ainda precisa dar saída aos últimos PACKETs do fluxo de sistemas.

Considerando, entretanto, que o processamento pode ser realizado durante a exibição da mídia e as medidas apresentadas da Subseção 7.2.1, pode-se afirmar que os principais fatores de retardo inicial a serem considerados são a montagem de um PACKET inicial e retardo introduzido por dependências de bits nos fluxos elementares. Na realidade, o retardo inicial só é realmente um problema se o ajuste estiver sendo realizado em tempo de exibição e não for possível acessar os dados da mídia com velocidade superior à de exibição.

A montagem do PACKET inicial é imprescindível, uma vez que não é possível que a ferramenta de ajuste dê saída a um PACKET do fluxo de sistemas enquanto todos os seus bits não tenham sido processados. Isso acontece porque as estruturas PACKET do fluxo MPEG-2 possuem campos que devem ser atualizados somente depois do processamento de todos os quadros de fluxos elementares nele contidos.

Para dar uma idéia do retardo que pode ser introduzido por este fator, pode-se considerar o pior caso, que é o PACKET de áudio. Sabendo que o tamanho máximo de um PACKET é de 2^{16} bytes e que o tamanho típico de um pacote de áudio é de *417*bytes, é possível codificar até *157* quadros de áudio em um único PACKET. No entanto, dificilmente o codificador irá deixar tantos quadros de áudio juntos, sendo seu comportamento padrão intercalá-los entre quadros de vídeos. Nos exemplos testados, tem-se um valor típico de *20* quadros de áudio em um único PACKET. Ainda com número reduzido de quadros, é necessário acrescentar um retardo cerca de $20*26.12*10^{-3}\approx 0.5$ segundos para receber todos os dados e começar a realizar o ajuste.

Além disso, deve-se considerar o retardo introduzido por dependências de bits nos fluxos elementares. O algoritmo de ajuste de áudio introduz retardo em formatos que possuem dependência entre quadros, pois é necessário armazenar em memória todos os quadros dos quais o quadro processado depende. Por exemplo, como mencionado na Subseção 3.4.1, em fluxos MP3 é possível que um quadro possa depender de até *10* quadros anteriores. Assim, é possível ter que introduzir um retardo de até $10*36*10^{-3}=0.36$ segundos para receber os dados de mídia em fluxos MP3. No caso de fluxos de sistemas a situação é ainda pior, pois um quadro de áudio que está em um PACKET pode depender de um quadro que está

em um PACKET anterior. Como esses dois PACKETs ainda podem estar intercalados por PACKETs de fluxos de outras mídias, o retardo a ser introduzido pode ser muito grande.

O algoritmo de MPEG-2 vídeo também introduz um retardo devido à ordem de codificação dos quadros de vídeo ser diferente da ordem de apresentação (Cavendish, 2005). Esse problema introduz retardo de um único quadro de vídeo e, portanto, é menos sério do que o retardo do áudio.

Outro retardo a se observar é o tempo que a ferramenta de ajuste precisa para modificar algum parâmetro de processamento, por exemplo, o fator de ajuste a ser aplicado. Nesse caso, é importante observar o tamanho dos *buffers* existentes para a realização do ajuste (ver Figura 58) e para comunicação com o exibidor de conteúdo. Grandes *buffers* podem introduzir retardo se alguma mudança for solicitada. Por outro lado, pequenos *buffers* podem diminuir desempenho da ferramenta e até ocasionar falhas na exibição da mídia.

O retardo introduzido pela ferramenta de ajuste pode impactar em cenários em que existe interatividade do usuário. Soares (Soares, 2005) fornece alguns dados interessantes para ilustrar como o retardo prejudica a interatividade de uma ligação telefônica. A tabela abaixo ilustra como o aumento do valor de retardo pode confundir os participantes da conversa, aumentando tanto o tempo de silêncio de uma conversa quanto o tempo em que os dois participantes falam.

Estado	**Valor do Retardo**		
	40ms[16]	**600ms**	**1200ms**
Um lado fala	40.5%	40.0%	38.4%
Dois lados falam	4.2%	5.9%	6,2%
Mútuo silêncio	23.2%	25.9%	29.4%

Tabela 14 - Efeitos do aumento do retardo em conversa telefônica - fonte: (Soares, 2005).

Uma opção para resolver ou diminuir o problema do retardo é permitir que a ferramenta de ajuste tenha acesso aos dados audiovisuais com velocidade superior à de exibição. Por exemplo, quando a ferramenta está integrada a um formatador hipermídia, mecanismos de *prefetch* dos objetos no ambiente de execução podem ajudar a solucionar o problema.

[16] O valor de *40*ms é o tempo máximo permitido em uma ligação fixa percorrendo distâncias continentais (menos de *5000*Km).

8
Considerações Finais

Este trabalho apresenta o problema de aplicações que precisam solicitar ajuste elástico com requisitos bem definidos, como o processamento em tempo de exibição, manipulando fluxos de mídia compridos, sem se tornar dependente do decodificador e gerando mídias de alta fidelidade utilizando um fator de ajuste variável. Como foi anteriormente discutido, nenhuma das soluções de ajuste existentes na literatura atendem satisfatoriamente a tais requisitos e isso motiva a definição de novos algoritmos de ajuste voltados para essas aplicações.

Com essa motivação, este trabalho propõe algoritmos para ajuste elástico, em tempo de exibição e compilação, para fluxos de áudio comprimidos. Os algoritmos atuam diretamente sobre fluxos comprimidos sem utilizar mecanismos de decodificação e recodificação e com independência do exibidor de conteúdo. O ajuste elástico é proposto para um modelo genérico de fluxos de áudio e então instanciado para diversos padrões de áudio de alta qualidade atualmente utilizados.

Além desses, também são propostos algoritmos para manter o sincronismo intermídia quando o ajuste é aplicado em fluxos de sistemas. Esse algoritmo permite que as operações de ajuste elástico em cada mídia possam ser realizadas de forma independente, evitando-se o dilema da escolha de um instante específico em que o ajuste deva ocorrer simultaneamente em todos os fluxos individuais.

A ferramenta de ajuste permite que outras aplicações possam facilmente solicitar serviços interagindo apenas com suas APIs de ajuste. Além de implementar os algoritmos propostos, a ferramenta está integrada ao algoritmo desenvolvido por Cavendish (Cavendish, 2005), o que possibilita também oferecer o ajuste elástico em fluxos de vídeo MPEG-2. Vale ressaltar que as APIs de ajuste propostas são genéricas para serem implementadas de diferentes maneiras. A implementação realizada por este trabalho é apenas um exemplo.

Como estudo de caso, este trabalho discute como a ferramenta de ajuste pode ser integrada a sistemas hipermídia e realiza a integração com o sistema HyperProp através da construção de adaptadores de exibidores de conteúdo. A presença da ferramenta de ajuste permite que o formatador possa exibir corretamente hiperdocumentos que, de outro modo, não poderiam ter a consistência temporal preservada.

Algumas medidas de qualidade foram extraídas do algoritmo de ajuste. A elaboração de testes para comparar diferentes áudios com ajuste elástico processados por diferentes algoritmos é muito rara na literatura. Os resultados obtidos mostram que a qualidade do algoritmo proposto se aproxima à do *Sound Forge*, o que é um ótimo resultado, uma vez que o algoritmo proposto funciona em tempo de exibição e pode ser facilmente integrado a uma outra aplicação, ao contrário do *Sound Forge*, e que o *Sound Forge* é uma ferramenta bastante utilizada atualmente, inclusive em aplicações comerciais de processamento de áudio. Vale lembrar que em aplicações para manutenção das relações de sincronismo, o ajuste é aplicado apenas em pequenas partes do fluxo de áudio, o que reduz o efeito da queda de qualidade decorrente do processamento. Além disso, o tempo de processamento para executar o algoritmo proposto é muito menor do que o do *Sound Forge* e a duração atingida é mais próxima da ótima.

Amostras de áudios processados com o algoritmo proposto podem ser encontradas na seguinte URL:
http://www.telemidia.puc-rio.br/~smbm/ajusteaudio

A análise da implementação demonstra que a ferramenta de ajuste é extensível, principalmente para novos formatos de áudio. Além disso, aplicações clientes do ajuste elástico devem sempre considerar o retardo inicial que precisa ser introduzido quando é solicitado o processamento de ajuste em tempo de exibição.

Alguns trabalhos futuros precisam ainda ser desenvolvidos. O principal deles é terminar a integração do algoritmo de ajuste para fluxos de sistemas desenvolvido com o algoritmo de ajuste de vídeo de Cavendish (Cavendish, 2005). Isso implica não somente na criação de classes de integração como em algumas modificações no algoritmo de vídeo.

Outro trabalho importante é a criação de novos testes de qualidade subjetiva para medir a qualidade utilizando outros formatos de áudio ou mesmo MP3 com

diferentes taxa de bits e de amostragem. Também é importante definir o real limite de fator de ajuste que o algoritmo utilizar com qualidade aceitável e não simplesmente supor que o valor é 10%. Ademais, uma vez terminada a integração com o algoritmo de vídeo, é possível realizar testes com fluxos audiovisuais.

Além disso, algumas extensões aos algoritmos de ajuste importantes são generalizar o algoritmo de sistemas para manipular o formato do Fluxo de Transporte do MPEG-2 e o algoritmo de áudio para suportar o perfil principal do AAC. O algoritmo de ajuste de áudio também pode suportar novos formatos, como o OGG-Vorbis.

Em relação à implementação, alguns trabalhos futuros são:

- O algoritmo de verificação do sincronismo intermídia incluindo o tratamento de situações de erro, quando a ferramenta de ajuste não consegue atingir o fator desejado, sem perder o sincronismo intermídia;
- A criação ou remoção de estruturas de sistemas, se necessário, durante a realização de ajuste;
- O controle para que o intervalo máximo entre as amostras de relógio transmitidas em fluxos de sistemas não ultrapasse o tempo máximo definido;
- O suporte a fluxos AAC codificados em outros protocolos de transporte e alguns casos bem particulares de fluxos MPEG.

A comunicação com exibidores de conteúdo também precisa ser melhorada. Atualmente, ainda não existe uma comunicação que funcione de modo integrado à aplicação cliente com suporte a vários formatos. Para solucionar esse problema, é necessário desenvolver uma integração entre o VLC e um adaptador *FlexMediaDataSourceAdapter* utilizando Java JNI (*Java Native Interface*).

Por fim, embora a ferramenta de ajuste já esteja sendo utilizada pelo formatador HyperProp, é interessante criar novos exemplos de integração, utilizando as diversas funções da ferramenta de ajuste, principalmente explorando o processamento em tempo de exibição.

9
Referências

ALY, S.; YOUSSEF, A. **Synchronization-Sensitive Frame Estimation: Video Quality Enhancement**. Multimedia Tools and Applications, 2002.

AMIR, A.; PONCELEON, D.; BLANCHARD, B.; PETKOVIC, D.; SRINIVASAN, S.; Cohen, G. **Using Audio Time Scale Modification for Video Browsing**. Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000.

ARONS, B. **Techniques, Perception, and Applications of Time-Compressed Speech**. Proceedings of American Voice I/O Society, 1992.

ARONS, B. **Efficient Listening with Two Ears: Dichotic Time Compression and Spatialization**. Proceedings of the 1994 International Conference of Auditory Display, 1994.

ATSC. **Digital Audio Compression Standard (AC-3)**. 1995.

AUDIO ENGINEERING SOCIETY. **Perceptual Audio Coders: What to Listen For**. AES Technical Council, 2001.

BACHELET, B.; MAHEY, P.; RODRIGUES, R.F.; SOARES, L.F.G. **Elastic Time Computation in QoS-Driven Hypermedia Presentations**. Research Report RR-04-16, LIMOS, Blaise-Pascal University, Clermont-Ferrand, França, 2004.

BERNSEE, S.M. **Time Stretching and Pitch Shifting of Audio Signals**. DSP Dimension, 2003. Disponível em http://www.dspdimension.com. Acesso em set. 05.

BERNSEE, S.M. **C/C++ Library for High Quality Audio Time Stretching and Pitch Shifting**. DSP Dimension, 2005. Disponível em http://www.dspdimension.com/. Acesso em set. 05.

CAVENDISH, S.A. **Ferramenta de Adaptação de Ajuste Elástico em Fluxos MPEG2**. Dissertação de Mestrado, Departamento de Informática – PUC-Rio, Rio de Janeiro, Brasil, 2005.

COVELL, M.; SLANEY, M.; ROTHSTEIN, A. **FastMPEG: Time-Scale Modification of Bit-Compressed Audio Information**. Speech and Signal Processing (ICASSP), Proceedings of the 2001 International Conference on Acoustics (IEEE), Salt Lake City, UT, 2001.

COVELL, M.; SLANEY, M.; ROTHSTEIN, A.; **FastMPEG: Time-Scale Modification of Bit-Compressed Audio Information**. 2001. Disponível em http://www.slaney.org/covell/FastMPEG/. Acesso em out. 2005.

DOLBY. **Dolby Laboratories**. 2005. Disponível em http://www.dolby.com/. Acesso em set. 2005.

DOLBY. **Dolby Solutions for Signal Processing**. 2005. Disponível em http://www.dolby.com/professional/pro_audio_engineering/solutions_signalprocessing.html. Acesso em out. 2005.

ENOUNCE. **Enounce 2xAV**. 2003. Disponível em http://www.enounce.com/. Acesso em jan. 05.

FFMPEG. **ffmpeg**. Disponível em http://sourceforge.net/projects/ffmpeg/. Acesso em set. 05.

GABOR, D. **Theory of Communication**. Journal of Institution of Electrical Engineers, 1946.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison Wesley, Primeira edição, 1995.

GOLUBCHIK, L.; LUI, J.; MUNTZ, R. **Reducing I/O Demand in Video-On-Demand Storage Servers**. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, 1995.

HAMMER, F., **Time-scale Modification using the Phase Vocoder**, Dissertação de Mestrado, Institute for Electronic Music and Acoustics (IEM), Graz University of Music and Dramatic Arts, A-8010 Graz, Austria, 2001.

ISO/IEC. **Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s - Part 3: Audio**. 11172-3, 1993.

ISO/IEC. **Information technology - Generic coding of moving pictures and associated audio information - Part 7: Advanced Audio Coding (AAC)**. 13818-7, 1997.

ISO/IEC. **Information technology - Generic coding of moving pictures associated audio information - Part 3: Audio**. 13818-3, 1998.

ISO/IEC. **Information technology - Generic coding of moving pictures and associated audio information: Systems**. 13818-1, 2000.

ISO/IEC. **Information technology - Generic coding of moving pictures and associated audio information: Video**. 13818-2, 2000.

ISO/IEC. **Information technology - Coding of audio-visual objects - Part 3: Systems**. 14496-1, 2001.

ISO/IEC. **Information technology - Coding of audio-visual objects - Part 3: Audio**. 14496-3, 2001.

ISO/IEC. **Information technology -- Generic coding of moving pictures and associated audio information -- Part 7: Advanced Audio Coding (AAC)**. 13818-7, 2004.

ITU-T. **Subjective audiovisual quality assessment methods for multimedia applications**. Recomendação P.911, 1998.

JAIN, R. **The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling**. John Wiley & Sons, 1991.

JFFMPEG. **Codec pack for the Java Media Framework**, 2005. Disponível em http://jffmpeg.sourceforge.net/. Acesso em set. 2005.

LAROCHE, J.; DOLSON, M. **Improved Phase Vocoder Time-scale Modification of Audio**, IEEE Transactions on Speech and Audio Processing, 1999.

LEMAY, J.C. **Sample Calculators**, 1998. Disponível em http://mp3.deepsound.net/eng/samples_calculs.php. Acesso em set. 2005.

LEE, E.; NAKRA, T. M.; BORCHERS, J. **You're The Conductor: a Realistic Interactive Conducting System for Children**, Proceedings of the NIME 2004 International Conference on New Interfaces for Musical Expression, Japão, 1998.

LEVINE, S.; SMITH, J.O.III. **A Sines+Transients+Noise Audio Representation for Data Compression and Time/Pitch-Scale Modifications**. 105th Audio Engineering Society Convention, EUA, 1998.

LEVINE, S. **Audio Representations for Data Compression and Compressed Domain Processing**. Tese de Doutorado, Stanford University, 1998. Disponível em http://www-ccrma.stanford.edu/~scottl/thesis.html. Acesso em out. 2005.

LIN, C.; ZHOU, J.; YOUN, J.; SUN, M. **MPEG Video Streaming with VCR Functionality**. IEEE Transactions of Circuits and Systems for Video Technology, 2001.

MICROSOFT CORPORATION. **Windows Media Player**. 2005. Disponível em http://www.microsoft.com/windows/windowsmedia/. Acesso em dez. 05.

MOVING PICTURE EXPERTS GROUP. **Reference software ISO/IEC 14496-5**. 2001. Disponível em http://www.chiariglione.org/mpeg/standards.htm. Acesso em nov. 2005.

MUCHALUAT-SAADE, D.C.; SILVA, H.V.O. **NCL 2.0: Exploiting and Integrating New Concepts to Hypermedia Declarative Languages**. Relatório técnico, Laboratório Telemídia, 2004. Disponível em http://www.telemidia.puc-rio.br/. Acesso em out. 2005.

NILSSON, M. **ID3v2**. 2003. Disponível em http://www.id3.org/. Acesso em jun. 05.

NOLL, P. **Digital Audio for Multimedia**. Proceedings Signal Processing for Multimedia. NATO Advanced Audio Institute, 1999.

NOLL, P. **MPEG Digital Audio Coding Standards**. CRC Press LLC, 2000. Disponível em http://www.ece.ucdavis.edu/~mihaela/mpeg_audio_coding.pdf. Acesso em jan. 05.

NULLSOFT. **Winamp 5.11 Player**. 2005. Disponível em http://www.winamp.com/. Acesso em nov 05.

OMOIGUI, N.; HE, L.; GUPTA, A.; GRUDIN, J.; SANACKI, E. **Time-Compression: Systems Concerns, Usage, and Benefits**. Proceedings of the SIGCHI conference on Human factors in computing systems, EUA, 1999.

PERKINS, C. **RTP: Audio and Video for the Internet**. Addison Wesley, 2003.

PORTNOFF, M. R. **Time-scale Modification of Speech Based on Short-time Fourier Analysis**. IEEE Transactions on Acustics, Speech and Signal Processing, 1981.

PROSONIQ MPEX. **Minimum Perceived Loss Time Compression/Expansion**. 2004. Disponível em http://mpex.prosoniq.com/. Acesso em set. 2005.

PURNHAGEN, H. **MPEG Audio FAQ**. 2004. Disponível em http://www.tnt.uni-hannover.de/project/mpeg/audio/faq/mpeg2.html. Acesso em nov. 05.

RODRIGUES, R.F. **Formatação e Controle de Apresentações Hipermídia com Mecanismos de Adaptação Temporal**. Tese de Doutorado, Departamento de Informática – PUC-Rio, Rio de Janeiro, Brasil, 2003.

RONI MUSIC. **Amazing Slow Downer - an Easy Way to Slow Down the Music without Changing the Pitch**. 2005. Disponível em http://www.ronimusic.com/. Acesso em out. 05.

SOARES, L.F.G. **Notas de Aula do Curso Fundamentos de Multimídia**. Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil, 2005.

SONY. **The Sound Forge Product Family**. 2005. Disponível em http://www.soundforge.com. Acesso em ago. 2005.

SUN MICROSYSTEMS. **Java Media Framework API Specification**. 1999. Disponível em http://java.sun.com/jmf. Acesso em set. 05.

VIDEOLAN. **VLC media player**. 2005. Disponível em http://www.videolan.org/vlc/. Acesso em nov. 2005.

WINKLER, S.; FALLER, C. **Audiovisual Quality Evaluation of Low-Bitrate Video**. Proc. SPIE/IS&T Human Vision and Electronic Imaging, 2005.

WOLTERS, M.; KJÖRLING, M.; HOMM, D.; PURNHAGEN, H. **A Closer Look into MPEG-4 High Efficiency AAC**. 2003. Disponível em www.telos-systems.com/techtalk/hosted/HE-AAC.pdf. Acesso em nov. 2005.

10 Apêndice A:
Exemplos de Código para Solicitar Ajuste Elástico

A ferramenta de ajuste foi desenvolvida para que outras aplicações possam facilmente solicitar ajuste elástico utilizando a API especificada na Subseção 4.1.

A Tabela 15 ilustra um trecho de código em Java simplificado para uma aplicação cliente solicitar a geração em tempo de compilação de um arquivo contendo o fluxo de áudio e a apresentação do fluxo processado no exibidor de conteúdo VLC (Videolan, 2005).

O código apresentado a seguir é divido em quatro partes. A primeira delas é a configuração da ferramenta de ajuste. Para tal, é necessário chamar o método `config` passando o arquivo de entrada, o nome do arquivo de saída e o fator de ajuste. Além disso, a aplicação cliente se registra como observadora do evento de finalização de ajuste. A seguir, é necessário iniciar o processamento. Nessa parte, a aplicação cliente optou por criar uma nova *thread*. Na terceira parte, a aplicação cliente descobre o novo valor de um determinado instante da mídia original e imprime o valor encontrado. A última parte é o código da aplicação que deve ser executado quando esta foi avisada que o arquivo contendo o áudio processado já foi gerado. A aplicação cliente agora pode solicitar a apresentação do arquivo processado utilizando, no exemplo, o exibidor de conteúdo VLC.

```
//Para configurar a ferramenta de ajuste elástico
ICompileTimeTimescalePlayer timescalePlayer = new
TimescalePlayer();
String inputFile = "C:/MediaFiles/FileAudio.mp3";
Double factor = 0.90;
String outputFile =
        Functions.getOutputFileName(inputFile,outputFile,factor);
int inputType = IOConstants.InputSource.LOCAL;
InputTools inputTools =
        InputToolsBuilder.getInputTools(inputFile, inputType);
timescalePlayer.config(inputTools, outputFile, factor);
timescalePlayer.addTimescaleListener(this);
```

```
//Para iniciar a geração do ajuste
Thread t = new Thread(timescalePlayer);
t.start();
```

```
//Para descobrir o valor de uma âncora
double originalInstant = 10;
double t = timescalePlayer.getTimescaleInstant(originalInstant);
String result = "Mudou instante " + originalInstant + " por " + t;
System.out.println(result);
```

```
//Quando é avisado do evento de finalização do ajuste
String vlcPath = IOConstants.VLC_PATH;
IThirdPartyPlayer thirdPartPlayer = new VLCFilePlayer(vlcPath);
String outputFile =  timescalePlayer.getOutputTools().getName();
thirdPartPlayer.setInput(outputFile);
thirdPartPlayer.prepare();
thirdPartPlayer.play();
```

Tabela 15 - Trechos de código para solicitar a geração e exibição de um arquivo ajustado.

Um segundo exemplo é ilustrado na Tabela 16. Nesse caso, uma aplicação cliente está solicitando ajuste elástico em tempo de exibição com a apresentação do fluxo processado no exibidor de conteúdo JMF (Sun, 1999).

O código apresentado a seguir é divido em quatro partes. A primeira delas é a configuração da ferramenta de ajuste. Para tal, é necessário chamar o método `config` passando o arquivo de entrada, o fator de ajuste e os instantes de tempo a serem monitorados. Além disso, a aplicação cliente se registra como observadora dos eventos de monitoramento de instantes de tempo. A seguir, é necessário iniciar a geração do ajuste. Nessa parte, a aplicação cliente optou por criar uma nova *thread*. A terceira parte é a configuração da ferramenta de exibição, que é executada na *thread* corrente. A última parte é o código da aplicação que deve ser executado quando esta foi avisada que o valor de um instante monitorado foi recalculado. O código de exemplo apenas exibe o novo valor encontrado.

```
//Para configurar a ferramenta de ajuste elástico
IExecutionTimeTimescalePlayer timescalePlayer = new
TimescalePlayer();
String inputFile = "C:/MediaFiles/FileAudio.mp3";
Double factor = 0.90;
int inputType = IOConstants.InputSource.LOCAL;
InputTools inputTools =
```

```
        InputToolsBuilder.getInputTools(inputFile, inputType);
IntantsSet instants = new InstantsSet(getInstants());
this.timescalePlayer.config(inputTools, factor, instants);
this.timescalePlayer.addIntantListener(this.guiUpdater);
```

```
//Para iniciar a geração do ajuste e exibição de conteúdo
Thread t = new Thread(timescalePlayer);
t.start();
```

```
IThirdPartyPlayer ap = new JMFDataSourcePlayer();
ByteOutputTools out = (ByteOutputTools)
timescalePlayer.getOutputTools();
InputTools in =  out.getInputTools();
ap.setInput(in);
ap.prepare();
ap.play();
```

```
//Quando é avisado de descoberta de valor de instante monitorado
String result = "Mudou instante " + event.getOriginalInstant() + "
por " + event.getUpdatedInstant()+ "\n";
System.out.println(result);
```

Tabela 16 - Trechos de código para solicitar a exibição de um fluxo sendo ajustado.

11 Apêndice B:
Aplicação de envio e recepção de fluxo de mídia

Uma aplicação foi desenvolvida com objetivo de transmitir dados de mídia, recebê-los e encaminhá-los à ferramenta de ajuste elástico para apresentação em tempo de exibição. Para tal, dois novos subsistemas, um transmissor e um receptor, foram desenvolvidos, como ilustrado na Figura 75.

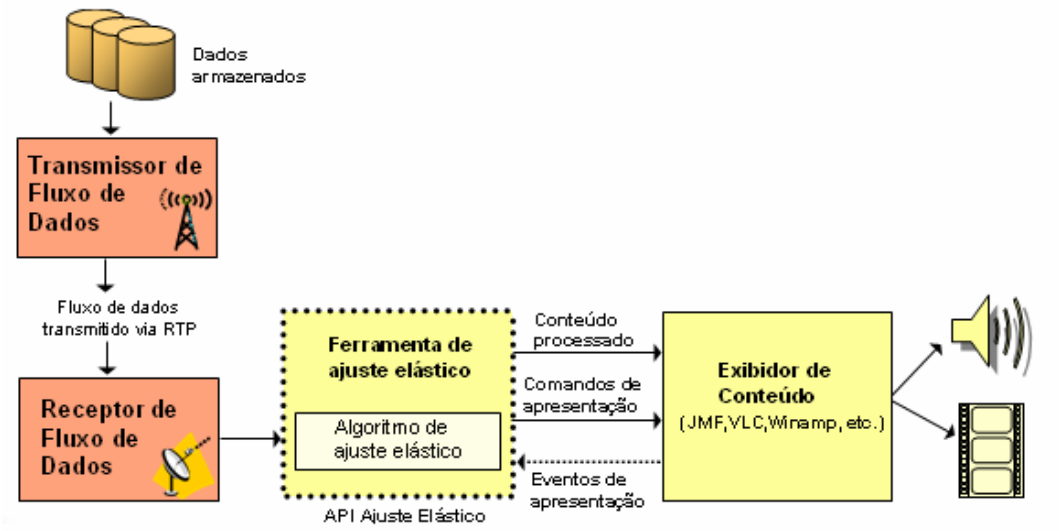


Figura 75 - Exibição de uma mídia com aplicação de ajuste elástico recebidas em tempo de exibição.

O subsistema **Transmissor** gera um fluxo de mídia a partir da leitura de um arquivo e o transmite utilizando o protocolo RTP. É esse subsistema quem inicia e controla a transmissão. O subsistema **Receptor** está sempre esperando receber fluxo em uma porta específica. Quando os dados chegam, o receptor os envia para a ferramenta de ajuste processá-lo e posteriormente exibi-lo utilizando a ferramenta de exibição JMF.

12 Apêndice C:
Exemplo da Integração com Formatador HyperProp

A Tabela 17 ilustra um documento NCL (Muchaluat & Silva, 2004) que precisa de funções de ajuste para ser corretamente apresentado pelo formatador HyperProp. Nas linhas *39* a *61* do documento, é especificado que a apresentação hipermídia é formada pelas seguintes mídias: um documento HTML contendo o título da apresentação, uma música em MP3, sua letra em HTML e *20* imagens. Os relacionamentos que devem ocorrer durante a apresentação, apresentados nas linhas *62* a *155*, são os seguintes:

- No início da apresentação, devem ser exibidas as mídias do título e da letra da música, a primeira imagem e começar a tocar o MP3 da música.
- As *20* imagens devem ser exibidas em seqüência, ou seja, inicialmente exibe a primeira durante um tempo, depois exibe a segunda também por um tempo, terceira até a vigésima. Desse modo, em nenhum momento duas imagens são exibidas simultaneamente.
- A apresentação do título, da letra da música, da vigésima imagem e do MP3 devem terminar simultaneamente.

As linhas *6* a *14* indicam a região do dispositivo de exibição onde cada mídia deve ser exibida. Todas as imagens devem ser exibidas na mesma região, só que em instantes diferentes. As linhas *26* a *35* especificam as características de exibição das mídias. Dentre elas, tem-se que o título da apresentação e a letra da música devem ser exibidos por *96* segundos, que a duração esperada de exibição de cada imagem é de *6* segundos, o nome da ferramenta de exibição do áudio e que a música e cada imagem possuem duração flexível especificadas por funções de custo. A duração esperada da música não é especificada no documento porque é calculada pela sua ferramenta de exibição.

As funções de custo são definidas nas linhas *14* a *25*. Cada uma delas especifica os valores permitidos para o fator de ajuste (`deltaShrink` e

`deltaStretch`), o custo máximo para aumentar ou diminuir a duração da mídia (`maxShrinkCost` e `maxStretchCost`) e o tipo da função (`type`).

Para respeitar as restrições do documento, a apresentação deve durar *96* segundos. Considerando que o tempo esperado para exibir a seqüência de imagens é de *20*6=120* segundos, é necessário diminuir o tempo esperado de exibição de cada imagem. Em média, cada imagem deverá durar *96/20=4.8* segundos, que implica na utilização de um fator de ajuste igual a *0.8*. O áudio MP3 pode ter duração entre *96*0.9 (=86.4)* e *96*1.1 (=105.6)* segundos. Supondo que o áudio MP3 dure *100* segundos, a ferramenta de ajuste deve ser utilizada com fator igual a *0.96*.

Por fim, caberá ao formatador escolher se o ajuste deve ser realizado em tempo de compilação ou de exibição instanciando o adaptador para exibidor de conteúdo adequado.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <ncl id="LaVieEnRose" xmlns="http://www.telemidia.puc-rio.br/specs/xml/NCL22/profiles"
03 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04 xsi:schemaLocation="http://www.telemidia.puc-rio.br/specs/xml/NCL22/profiles/NCL22.xsd">
05   <head>
06     <regionBase>
07       <region id="menuWindow" title="Seminaire du Limos (R. F. Rodrigues)"  width="1024"
08 height="740">
09         <region id="titleRegion" width="100%" height="100" />
10         <region id="lyricsRegion" top="100" width="40%" height="590" />
11         <region id="imageRegion" top="100" left="40%" width="60%" height="590" />
12       </region>
13     </regionBase>
14     <costFunctionBase>
15       <costFunction id="costFunction01" xsi:type="linear" deltaShrink="10%"
16  deltaStretch="10%">
17         <costFunctionParam name="maxShrinkCost" value="1000"/>
18         <costFunctionParam name="maxStretchCost" value="1000"/>
19       </costFunction>
20       <costFunction id="costFunction02" xsi:type="linear" deltaShrink="30%"
21  deltaStretch="30%">
22         <costFunctionParam name="maxShrinkCost" value="5000"/>
23         <costFunctionParam name="maxStretchCost" value="5000"/>
24       </costFunction>
25     </costFunctionBase>
26     <descriptorBase>
27       <descriptor id="titleDesc" region="titleRegion" explicitDur="96s"/>
28       <descriptor id="lyricsDesc" region="lyricsRegion" explicitDur="96s"/>
29       <descriptor id="audioDesc01" player="HYPF_FlexMP3AudioPlayer"
30  costFunction="costFunction01">
31         <descriptorParam name="soundLevel" value="0.3"/>
32       </descriptor>
33       <descriptor id="imageDesc" region="imageRegion" explicitDur="6s"
34  costFunction="costFunction02"/>
35     </descriptorBase>
36     <connectorBase><import alias="connBase" baseURI="limos.conn"/></connectorBase>
```

```
37   </head>
38   <body><port id="entryPort" component="title-lvr01"/>
39     <media type="audio" descriptor="audioDesc01" id="song-lvr01" src="laVieenRose.mp3"/>
40     <media type="text" descriptor="titleDesc" id="title-lvr01" src="lyrics/title.html"/>
41     <media type="text" descriptor="lyricsDesc" id="lvr01-lyrics" src="lyrics/lyr.html"/>
42     <media type="text" descriptor="imageDesc" id="lvr01-img1" src="images/image01.html"/>
43     <media type="text" descriptor="imageDesc" id="lvr01-img2" src="images/image02.html"/>
44     <media type="text" descriptor="imageDesc" id="lvr01-img3" src="images/image03.html"/>
45     <media type="text" descriptor="imageDesc" id="lvr01-img4" src="images/image04.html"/>
46     <media type="text" descriptor="imageDesc" id="lvr01-img5" src="images/image05.html"/>
47     <media type="text" descriptor="imageDesc" id="lvr01-img6" src="images/image06.html"/>
48     <media type="text" descriptor="imageDesc" id="lvr01-img7" src="images/image07.html"/>
49     <media type="text" descriptor="imageDesc" id="lvr01-img8" src="images/image08.html"/>
50     <media type="text" descriptor="imageDesc" id="lvr01-img9" src="images/image09.html"/>
51     <media type="text" descriptor="imageDesc" id="lvr01-img10" src="images/image10.html"/>
52     <media type="text" descriptor="imageDesc" id="lvr01-img11" src="images/image11.html"/>
53     <media type="text" descriptor="imageDesc" id="lvr01-img12" src="images/image12.html"/>
54     <media type="text" descriptor="imageDesc" id="lvr01-img13" src="images/image13.html"/>
55     <media type="text" descriptor="imageDesc" id="lvr01-img14" src="images/image14.html"/>
56     <media type="text" descriptor="imageDesc" id="lvr01-img15" src="images/image15.html"/>
57     <media type="text" descriptor="imageDesc" id="lvr01-img16" src="images/image16.html"/>
58     <media type="text" descriptor="imageDesc" id="lvr01-img17" src="images/image17.html"/>
59     <media type="text" descriptor="imageDesc" id="lvr01-img18" src="images/image18.html"/>
60     <media type="text" descriptor="imageDesc" id="lvr01-img19" src="images/image19.html"/>
61     <media type="text" descriptor="imageDesc" id="lvr01-img20" src="images/image20.html"/>
62     <link id="link1" xconnector="connBase#onBeginStart">
63       <bind component="title-lvr01" role="onBegin"/>
64       <bind component="song-lvr01" role="start"/>
65       <bind component="lvr01-img1" role="start"/>
66       <bind component="lvr01-lyrics" role="start"/>
67     </link>
68     <link id="link2" xconnector="connBase#endConstraint">
69       <bind component="song-lvr01" role="xEnd"/>
70       <bind component="title-lvr01" role="yEnd"/>
71     </link>
72     <link id="link3" xconnector="connBase#endConstraint">
73       <bind component="title-lvr01" role="xEnd"/>
74       <bind component="lvr01-img20" role="yEnd"/>
75     </link>
76     <link id="link4" xconnector="connBase#endConstraint">
77       <bind component="title-lvr01" role="xEnd"/>
78       <bind component="lvr01-lyrics" role="yEnd"/>
79     </link>
80     <link id="linkImage01" xconnector="connBase#onEndStart">
81       <bind component="lvr01-img1" role="onEnd"/>
82       <bind component="lvr01-img2" role="start"/>
83     </link>
84     <link id="linkImage02" xconnector="connBase#onEndStart">
85       <bind component="lvr01-img2" role="onEnd"/>
86       <bind component="lvr01-img3" role="start"/>
87     </link>
88     <link id="linkImage03" xconnector="connBase#onEndStart">
89       <bind component="lvr01-img3" role="onEnd"/>
90       <bind component="lvr01-img4" role="start"/>
91     </link>
92     <link id="linkImage04" xconnector="connBase#onEndStart">
93       <bind component="lvr01-img4" role="onEnd"/>
94       <bind component="lvr01-img5" role="start"/>
```

```
95      </link>
96      <link id="linkImage05" xconnector="connBase#onEndStart">
97        <bind component="lvr01-img5" role="onEnd"/>
98        <bind component="lvr01-img6" role="start"/>
99      </link>
100     <link id="linkImage06" xconnector="connBase#onEndStart">
101       <bind component="lvr01-img6" role="onEnd"/>
102       <bind component="lvr01-img7" role="start"/>
103     </link>
104     <link id="linkImage07" xconnector="connBase#onEndStart">
105       <bind component="lvr01-img7" role="onEnd"/>
106       <bind component="lvr01-img8" role="start"/>
107     </link>
108     <link id="linkImage08" xconnector="connBase#onEndStart">
109       <bind component="lvr01-img8" role="onEnd"/>
110       <bind component="lvr01-img9" role="start"/>
111     </link>
112     <link id="linkImage09" xconnector="connBase#onEndStart">
113       <bind component="lvr01-img9" role="onEnd"/>
114       <bind component="lvr01-img10" role="start"/>
115     </link>
116     <link id="linkImage10" xconnector="connBase#onEndStart">
117       <bind component="lvr01-img10" role="onEnd"/>
118       <bind component="lvr01-img11" role="start"/>
119     </link>
120     <link id="linkImage11" xconnector="connBase#onEndStart">
121       <bind component="lvr01-img11" role="onEnd"/>
122       <bind component="lvr01-img12" role="start"/>
123     </link>
124     <link id="linkImage12" xconnector="connBase#onEndStart">
125       <bind component="lvr01-img12" role="onEnd"/>
126       <bind component="lvr01-img13" role="start"/>
127     </link>
128     <link id="linkImage13" xconnector="connBase#onEndStart">
129       <bind component="lvr01-img13" role="onEnd"/>
130       <bind component="lvr01-img14" role="start"/>
131     </link>
132     <link id="linkImage14" xconnector="connBase#onEndStart">
133       <bind component="lvr01-img14" role="onEnd"/>
134       <bind component="lvr01-img15" role="start"/>
135     </link>
136     <link id="linkImage15" xconnector="connBase#onEndStart">
137       <bind component="lvr01-img15" role="onEnd"/>
138       <bind component="lvr01-img16" role="start"/>
139     </link>
140     <link id="linkImage16" xconnector="connBase#onEndStart">
141       <bind component="lvr01-img16" role="onEnd"/>
142       <bind component="lvr01-img17" role="start"/>
143     </link>
144     <link id="linkImage17" xconnector="connBase#onEndStart">
145       <bind component="lvr01-img17" role="onEnd"/>
146       <bind component="lvr01-img18" role="start"/>
147     </link>
148     <link id="linkImage18" xconnector="connBase#onEndStart">
149       <bind component="lvr01-img18" role="onEnd"/>
150       <bind component="lvr01-img19" role="start"/>
151     </link>
152     <link id="linkImage19" xconnector="connBase#onEndStart">
```

```
153      <bind component="lvr01-img19" role="onEnd"/>
154      <bind component="lvr01-img20" role="start"/>
155    </link>
156  </body>
157 </ncl>
```

Tabela 17 - Hiperdocumento escrito em NCL contendo especificação que precisa da aplicação de ajuste elástico para correta exibição.

13 Apêndice D:
Aplicação de Teste da Ferramenta de Ajuste

Com o objetivo de utilizar a ferramenta de ajuste em diferentes condições e testar suas funcionalidades, uma aplicação de teste foi desenvolvida. A Figura 76 ilustra a interface gráfica dessa aplicação.

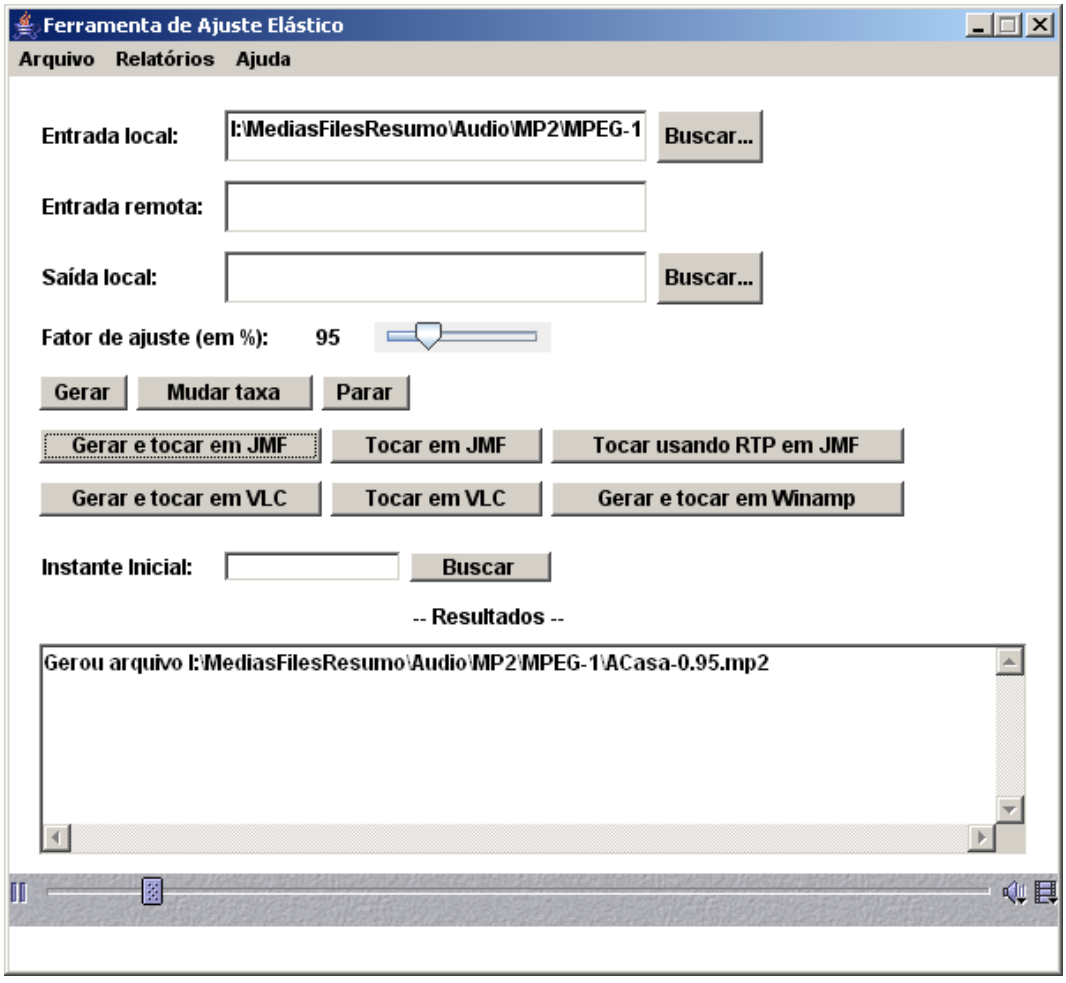


Figura 76 - Interface gráfica da aplicação de teste.

As funcionalidades da ferramenta de ajuste disponíveis através dessa interface desenvolvida são:

- Solicitar o ajuste de um arquivo de áudio, vídeo ou sistemas gerando como saída outro arquivo contendo a mídia processada (este arquivo pode ser posteriormente exibido ou não).

- Tocar um arquivo de mídia com a aplicação de ajuste elástico em tempo de exibição.
- Mudar o fator de ajuste utilizado no algoritmo de ajuste durante a geração de um novo arquivo ou a exibição de um fluxo que está sendo processado em tempo de exibição.
- Cancelar a aplicação de ajuste elástico e/ou a exibição de um fluxo de mídia.
- Visualizar o novo valor de um ou mais instantes de tempo durante a realização de um ajuste ou ao seu final.
- Visualizar um relatório contendo informações do último ajuste elástico realizado (quadros processados e tempo gasto para executar ajuste).

Os arquivos de mídia a serem processados podem estar locais na máquina ou remotos. Em qualquer dos casos, a aplicação de teste inicia e controla a transferência dos dados de mídia. Os exibidores de conteúdo utilizados foram: JMF, VLC e Winamp (ver Capítulo 5).