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.

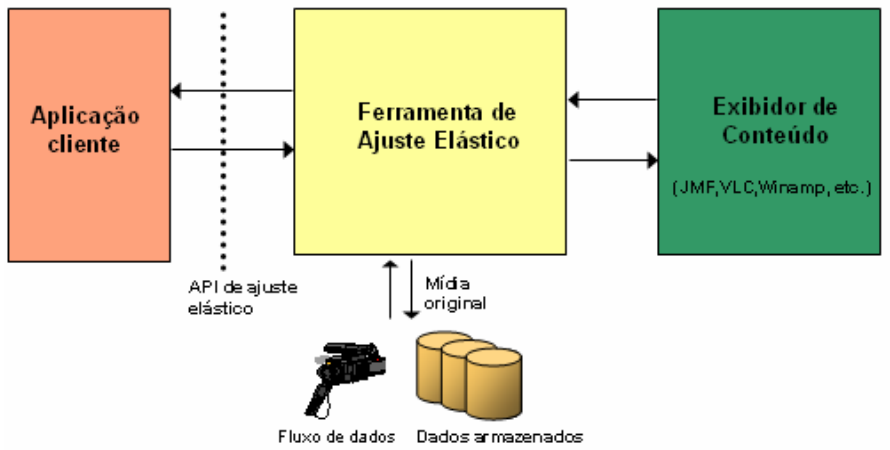


Figura 44 - Contexto de utilização da ferramenta de ajuste elástico.

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.

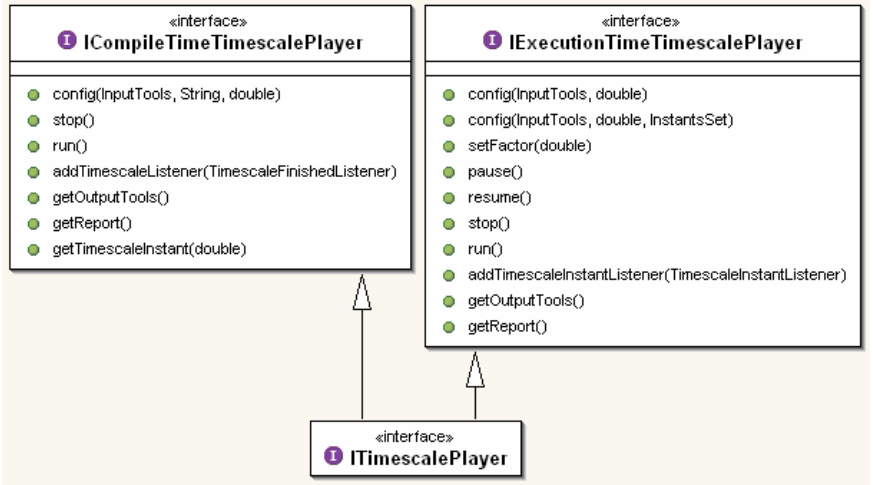


Figura 45 - Implementação das APIs de ajuste elástico.

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.

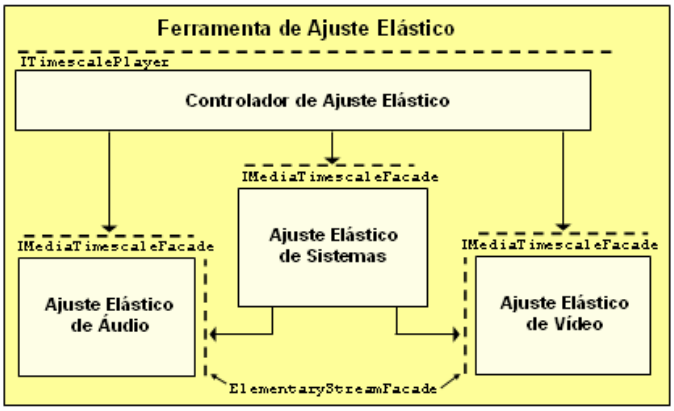


Figura 46 - Detalhes da ferramenta de ajuste elástico.

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.

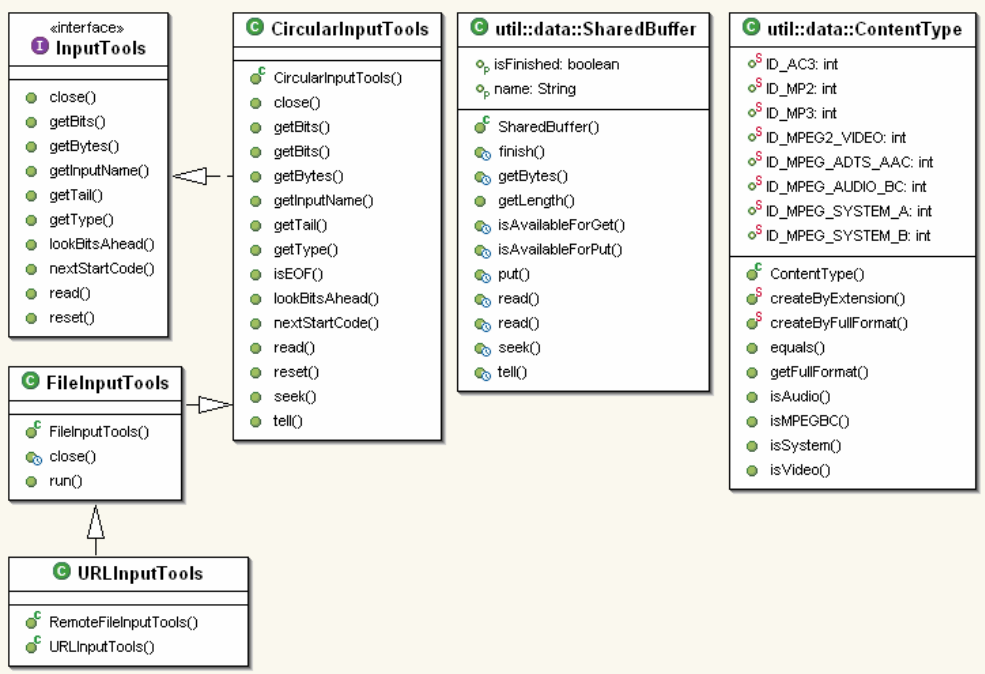


Figura 48 - Classes do subsistema de entrada.

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.

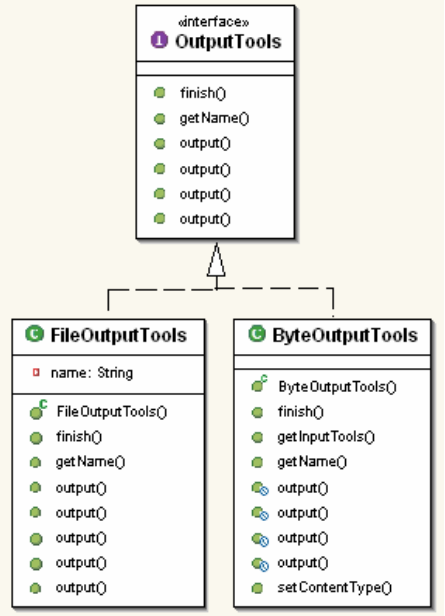


Figura 49 - Classes do subsistema de entrada.

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.

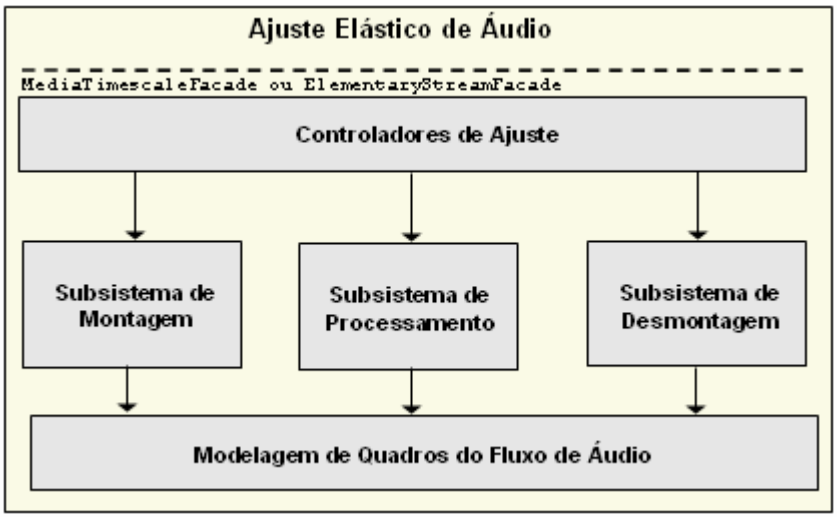


Figura 50 - Visão geral da implementação do algoritmo de ajuste de áudio.

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.

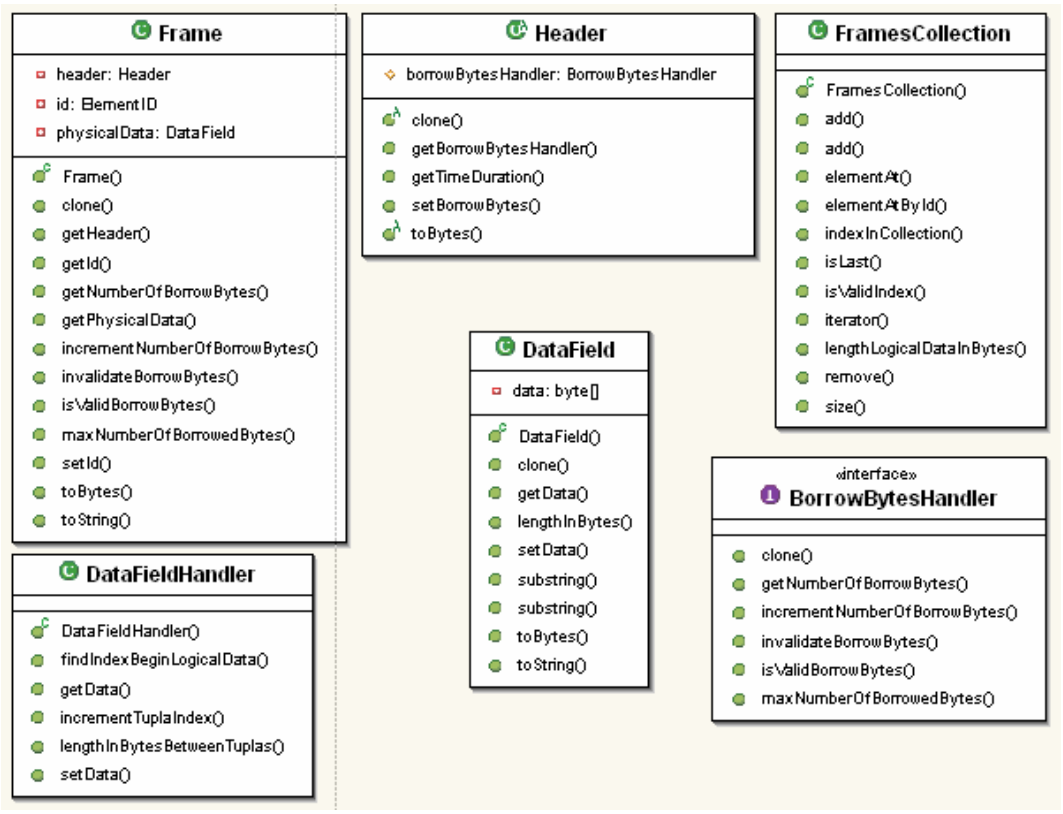


Figura 51 - Principais classes do formato genérico de áudio.

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.

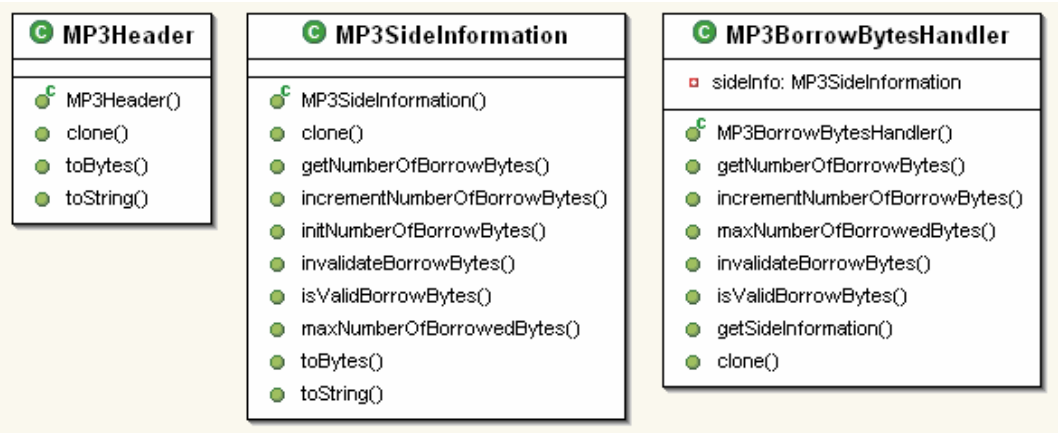


Figura 52 - Principais classes do formato MP3.

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.

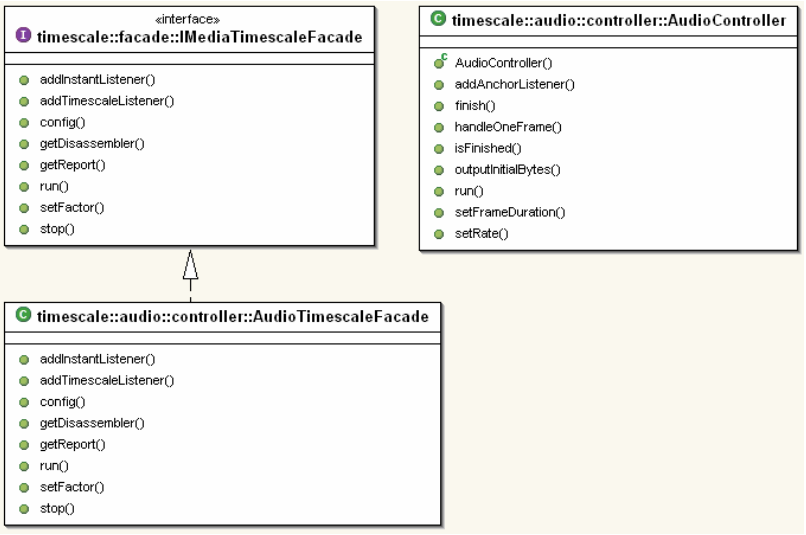


Figura 53 - Controladores do ajuste elástico de áudio.

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.

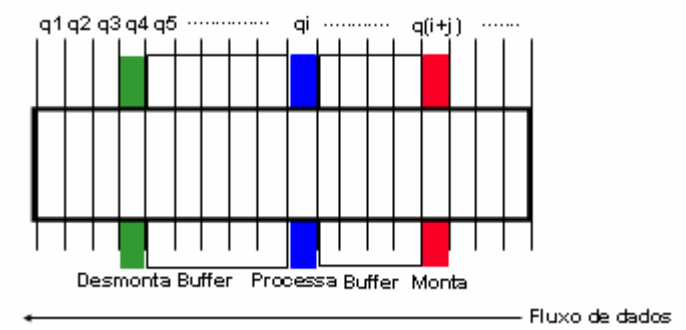


Figura 54 - Montagem (incluindo leitura), processamento e desmontagem (incluindo escrita) de dados de áudio.

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.

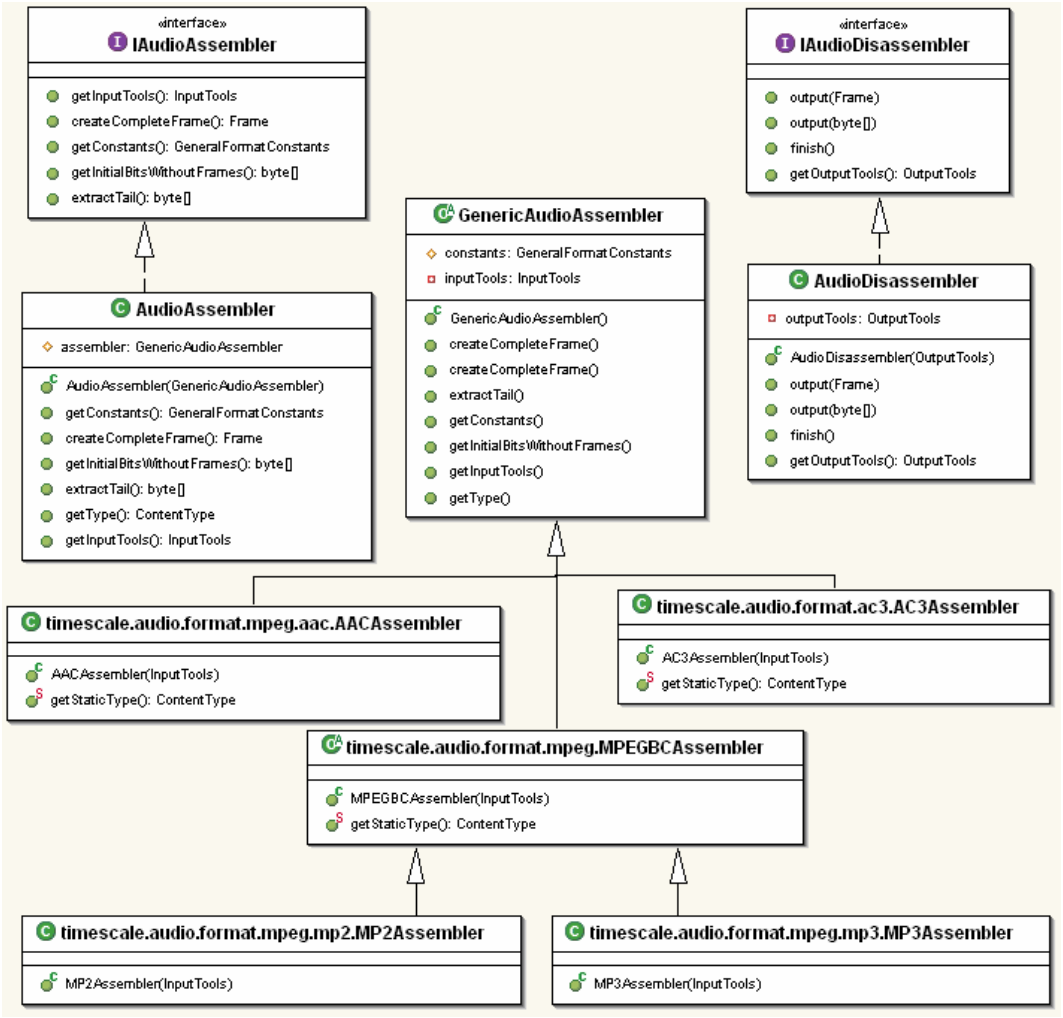


Figura 55 - *Assemblers* e *disassembler* de objetos de formatos de áudio.

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.

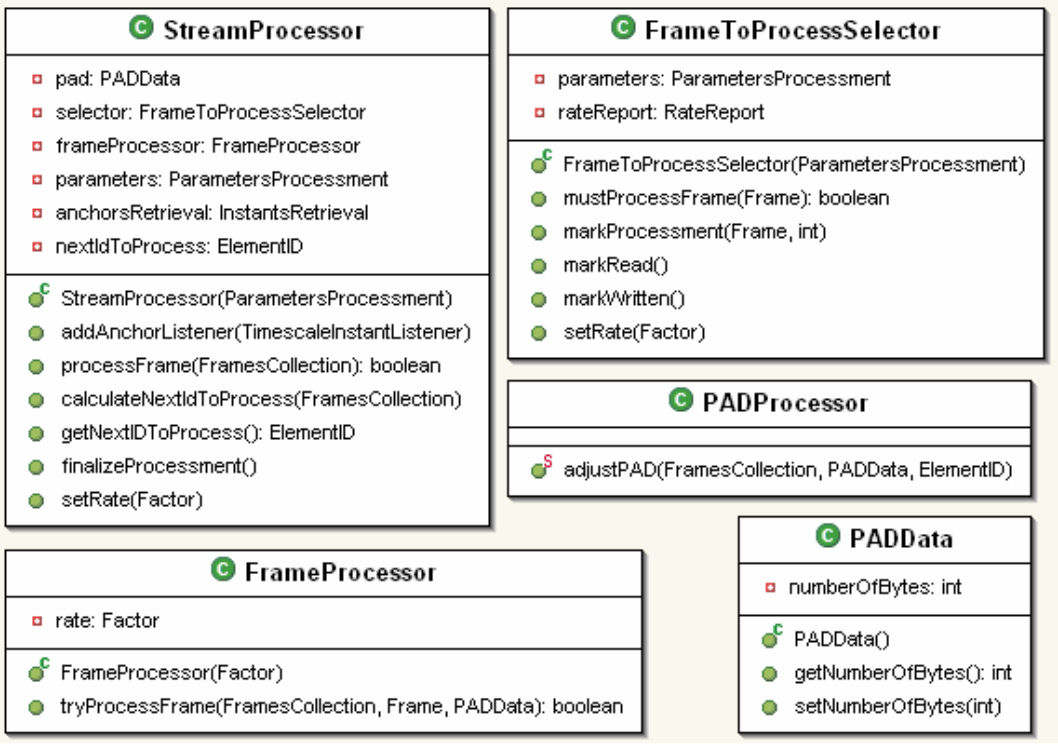


Figura 56 - Principais classes do subsistema de processamento de áudio.

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.

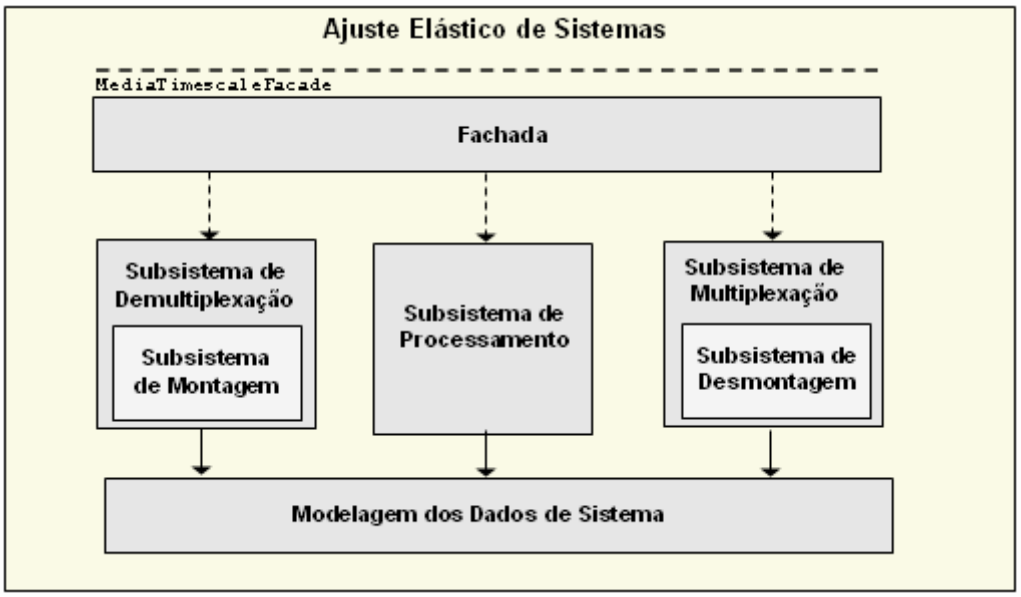


Figura 57 - Visão geral da implementação do algoritmo de ajuste em fluxos de sistemas.

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.

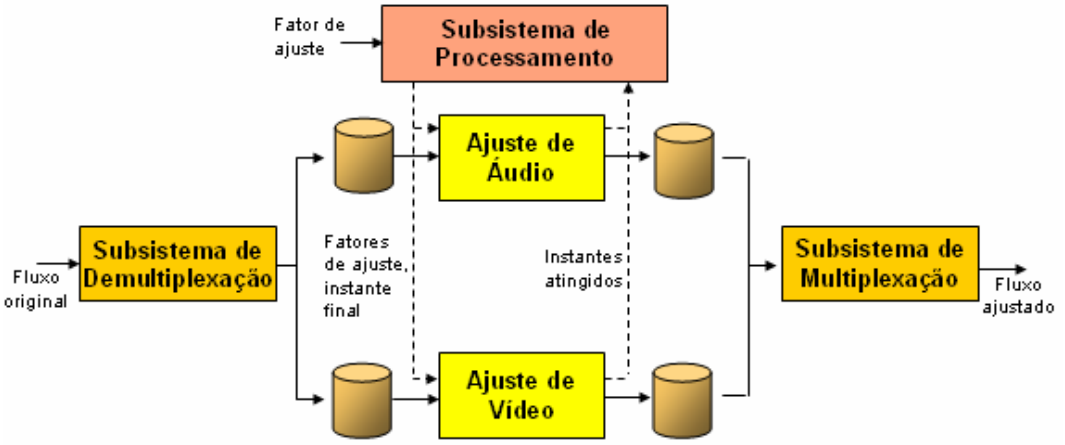


Figura 58 - Algoritmo de ajuste de dados audiovisuais.

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.

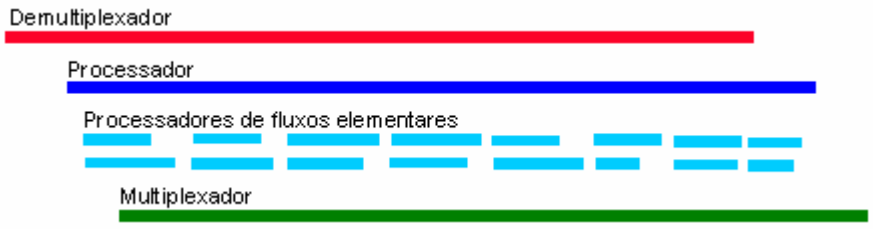


Figura 59 - Tempo de vida das *threads* responsáveis pelo ajuste elástico.

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.

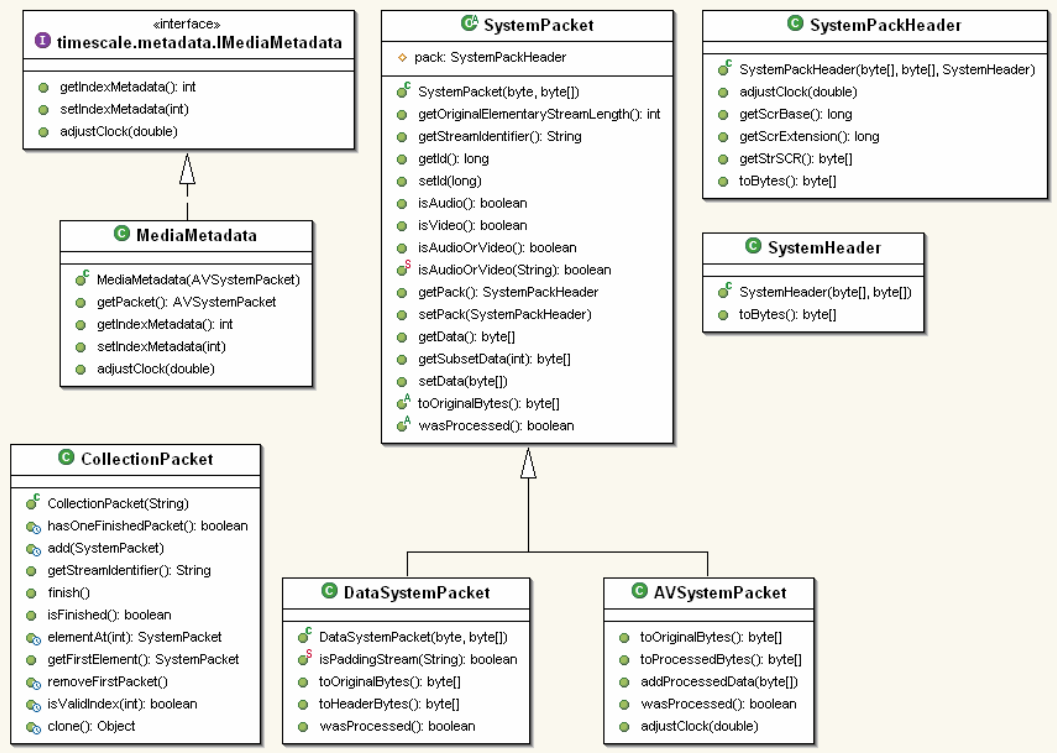


Figura 60 - Classes do formato de MPEG-2 sistemas.

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.

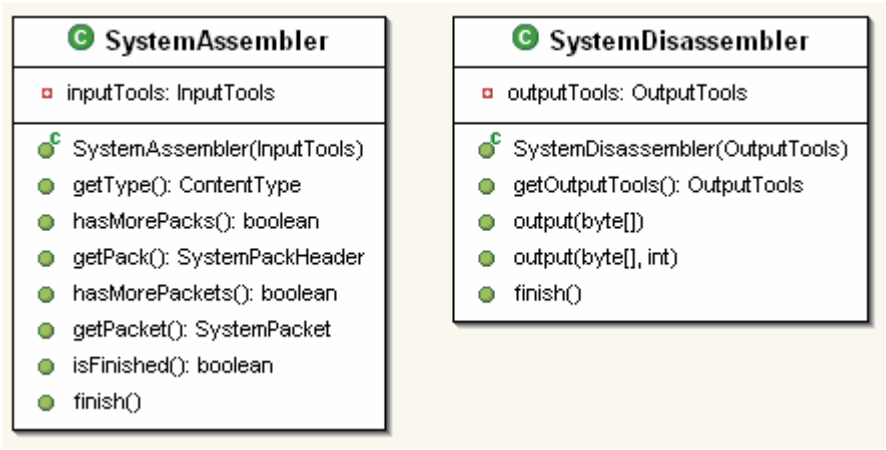


Figura 61 - Classes dos subsistemas de montagem e desmontagem.

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.

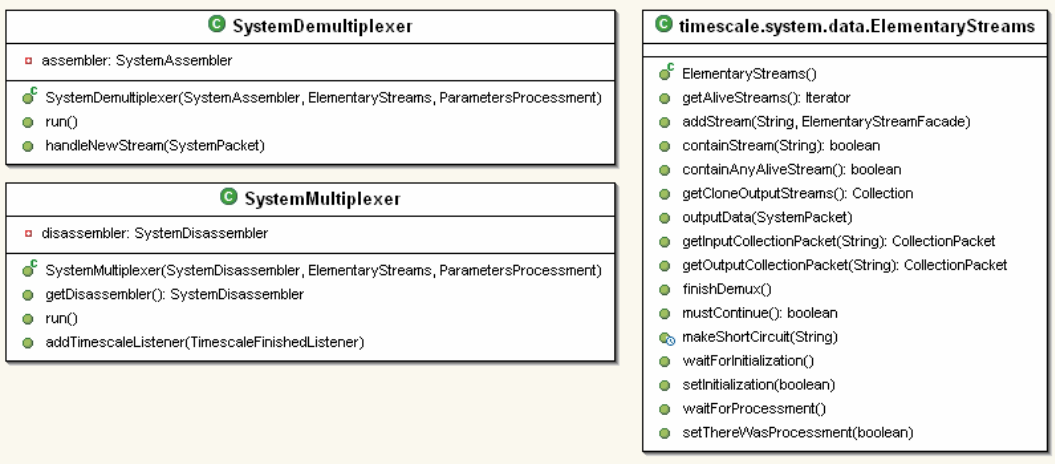


Figura 62 - Classes dos subsistemas de multiplexação e demultiplexação.

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.

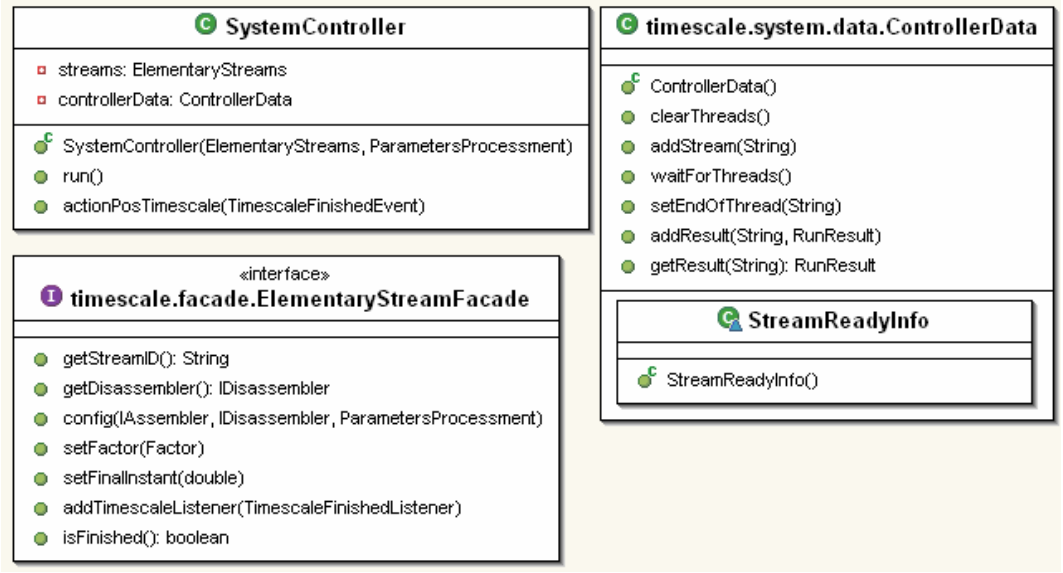


Figura 63 - Controladores do algoritmo de ajuste em fluxos de sistemas.

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.

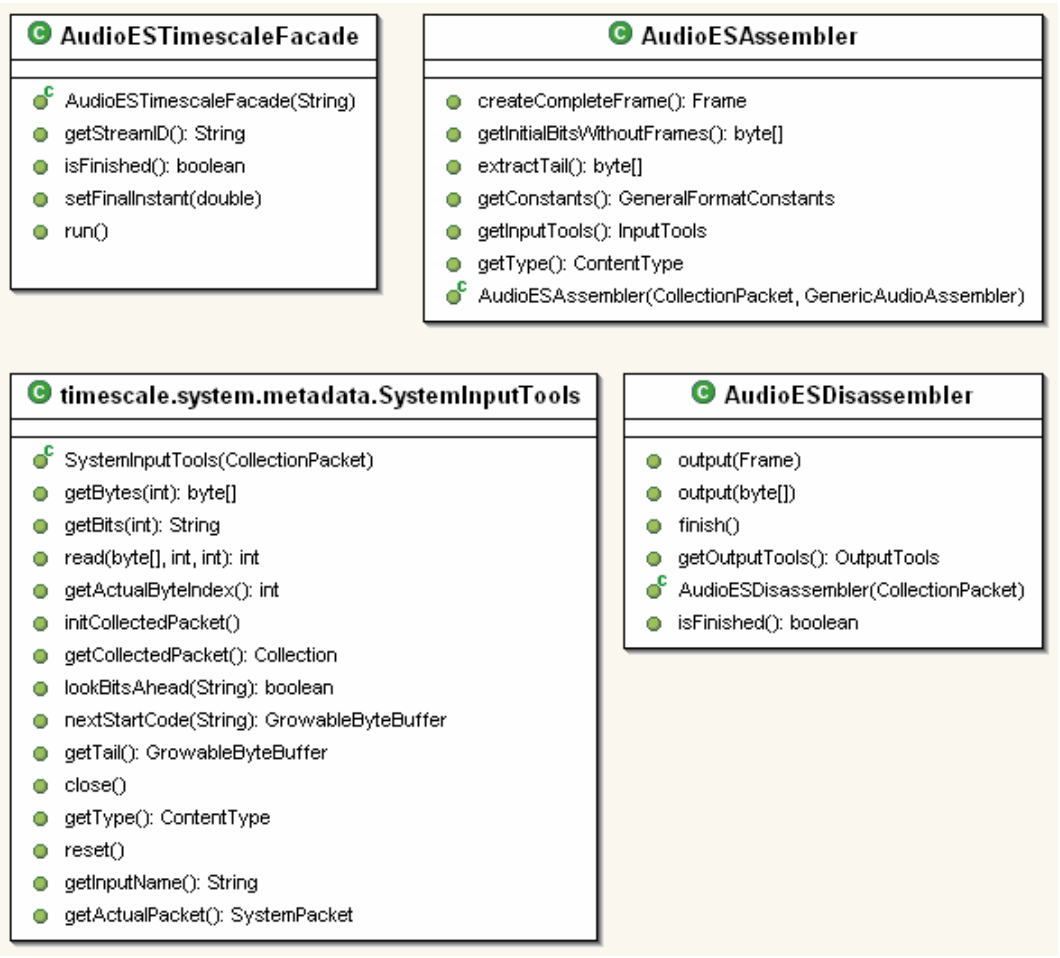


Figura 64 - Classes para realizar ajuste em fluxo elementar de áudio.

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).

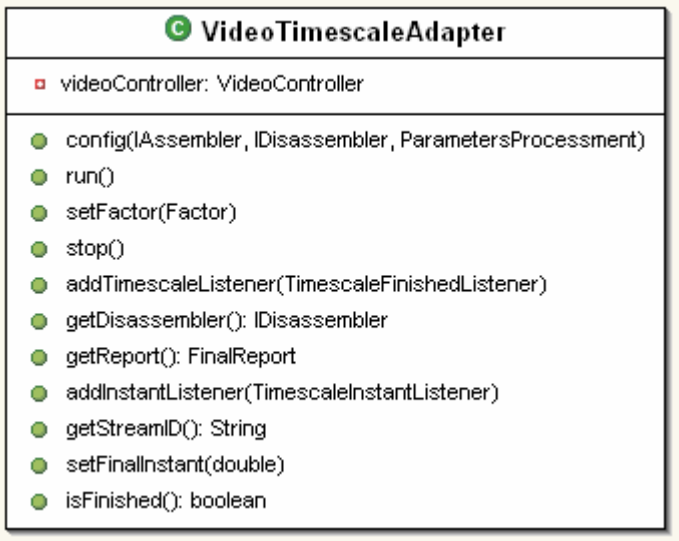
A classe `AudioESAssembler` implementa `IAudioAssembler` e, como tal, `é` responsável por criar objetos do modelo de dados de áudio. `AudioESAssembler`

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.