INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Perfil de Sistemas de Informação

Aplicação de gravação e edição de áudio multi-faixa colaborativa

(Relatório Final)

João Filipe Baltar Antunes
(Licenciado)

Trabalho de projecto para obtenção do grau de Mestre
em Engenharia Informática e de Computadores

Júri:
Presidente do Júri: Fernando Manuel Gomes de Sousa
Vogal - Arguente: Luís Manuel Costa Assunção
Vogal - Orientador: Artur Jorge Ferreira
Vogal - Orientador: Paulo Alexandre Leal Barros Pereira

Dezembro de 2010

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Perfil de Sistemas de Informação

Aplicação de gravação e edição de áudio multi-faixa colaborativa

(Relatório Final)

João Filipe Baltar Antunes
(Licenciado)

Trabalho de projecto para obtenção do grau de Mestre em Engenharia Informática e de Computadores

Aluno: João Filipe Baltar Antunes

Júri:
Presidente do Júri: Fernando Manuel Gomes de Sousa

Vogal - Arguente: Luís Manuel Costa Assunção

Vogal - Orientador: Artur Jorge Ferreira

Vogal - Orientador: Paulo Alexandre Leal Barros Pereira

Dezembro de 2010

Resumo

Os programas de gravação e edição de áudio em ambientes multi-faixa são populares entre os músicos, para desenvolverem o seu trabalho. Estes programas apresentam funcionalidades de gravação e edição, mas não promovem o trabalho colaborativo entre músicos. De forma a colaborar, os vários elementos de uma banda musical têm de se reunir no mesmo local físico.

Com este trabalho pretende-se criar uma solução para a colaboração no contexto da gravação e edição de áudio. Tem-se como objectivo o desenvolvimento de uma aplicação distribuída que facilite a gravação e edição de áudio, estando os elementos de cada banda musical em localizações físicas distintas.

A aplicação desenvolvida tem funcionalidades de manipulação de áudio, bem como mecanismos para a sincronização do trabalho entre os vários elementos da banda.

A manipulação de áudio consiste em reprodução, gravação, codificação e edição de áudio. O áudio é manipulado no formato Microsoft WAV, resultante da digitalização do áudio em *Pulse Code Modulation (PCM*) e posteriormente codificado em FLAC (Free Lossless Audio Codec) ou MP3 (Mpeg-1 Layer 3) de forma a minimizar a dimensão do ficheiro, diminuindo assim o espaço que ocupa em disco e a largura de banda necessária à sua transmissão pela internet. A edição consiste na aplicação de operações como amplificação, ecos, entre outros.

Os elementos da banda instalam no seu computador a aplicação cliente, com interface gráfica onde desenvolvem o seu trabalho. Esta aplicação cliente mantém a lógica de sincronização do trabalho colaborativo, inserindo-se como um dos *peers* da arquitectura *peer-to-peer* híbrida da aplicação distribuída. Estes *peers* comunicam entre si, enviando informação acerca das operações aplicadas e áudio gravado pelos membros da banda.

Palavras-chave:
Gravação, edição, áudio, colaboração, aplicação distribuída.

Abstract

Multi-track audio recording and editing programs are popular among musicians, in order to develop their work. These applications include several functionalities for recording and editing audio, but do not promote collaborative work. Existing applications require band members to physically meet in order to produce their work.

This project aims to create a solution for collaborative work in the audio recording and edition context. The goal of this project is to develop a distributed application that eases audio recording and editing, whilst band members are at different locations.

The application includes audio manipulation functionalities, as well as mechanisms to synchronize the work developed by each band member.

Audio manipulation consists of playback, recording, codification and edition. The audio is manipulated in Microsoft WAV format, result of the audio digitalization in Pulse Code Modulation (PCM) and later codified in FLAC (Free Lossless Audio Codec) or MP3 (Mpeg-1 Layer 3) to minimize the size of the file, requiring less disk space and less bandwidth for the audio transmission over the network. For audio edition there are some operations like amplification, echoes, among others.

Each band member installs in his/her computer the client application, with a graphical user interface in which they develop their work. This client application has the collaborative work synchronization logic, being one of the peers of the distributed application hybrid peer-to-peer architecture. These peers communicate among themselves, sending information about operations applied and audio recorded by the band members.

Keywords:

Recording, editing, audio, collaboration, distributed application.

Índice

1 – Introdução

O desenvolvimento de músicas em ambientes multi-faixa é bastante comum entre músicos. Existem vários programas para gravação e edição de música em ambiente multi-faixa, tais como o *Goldwave* [1], o *Multiquence* [2] e o *Adobe Audition* [3].

Entre as principais características destes programas estão a reprodução, gravação e edição de áudio. Relativamente à edição destacam-se as operações de cópia, truncamento, remoção e aplicação de efeitos através de operações aritméticas, amplificação/atenuação e filtragem. Estes programas não incluem nas suas funcionalidades uma forma de os seus utilizadores trabalharem de forma colaborativa, a partir de localizações diferentes.

Com este projecto pretende-se introduzir uma solução para o trabalho colaborativo no contexto da edição musical, possibilitando o desfasamento espacial e temporal dos músicos que colaboram na edição.

1.1 - Descrição geral dos objectivos

O objectivo do projecto é o desenvolvimento de uma aplicação distribuída para edição colaborativa de músicas. A ideia consiste em que uma banda musical possa, através da internet, desenvolver as suas músicas, por via da gravação e edição de faixas.

A aplicação apresenta ambiente de edição de áudio em multi-faixa, na linha dos programas de edição existentes. Suporta a gravação de faixas a partir de diferentes dispositivos de aquisição de áudio (microfone ou outros).

A aplicação inclui funcionalidades de suporte ao trabalho colaborativo, interligando os diferentes utilizadores e sincronizando o respectivo trabalho. A sincronização tem em conta o possível desfasamento temporal entre as operações efectuadas pelos utilizadores; assim, é possível que um utilizador edite determinado áudio, sem que os restantes estejam a usar a aplicação nessa altura.

1.2 – Descrição global da solução

Na Figura 1 pode observa-se a descrição global da solução implementada.

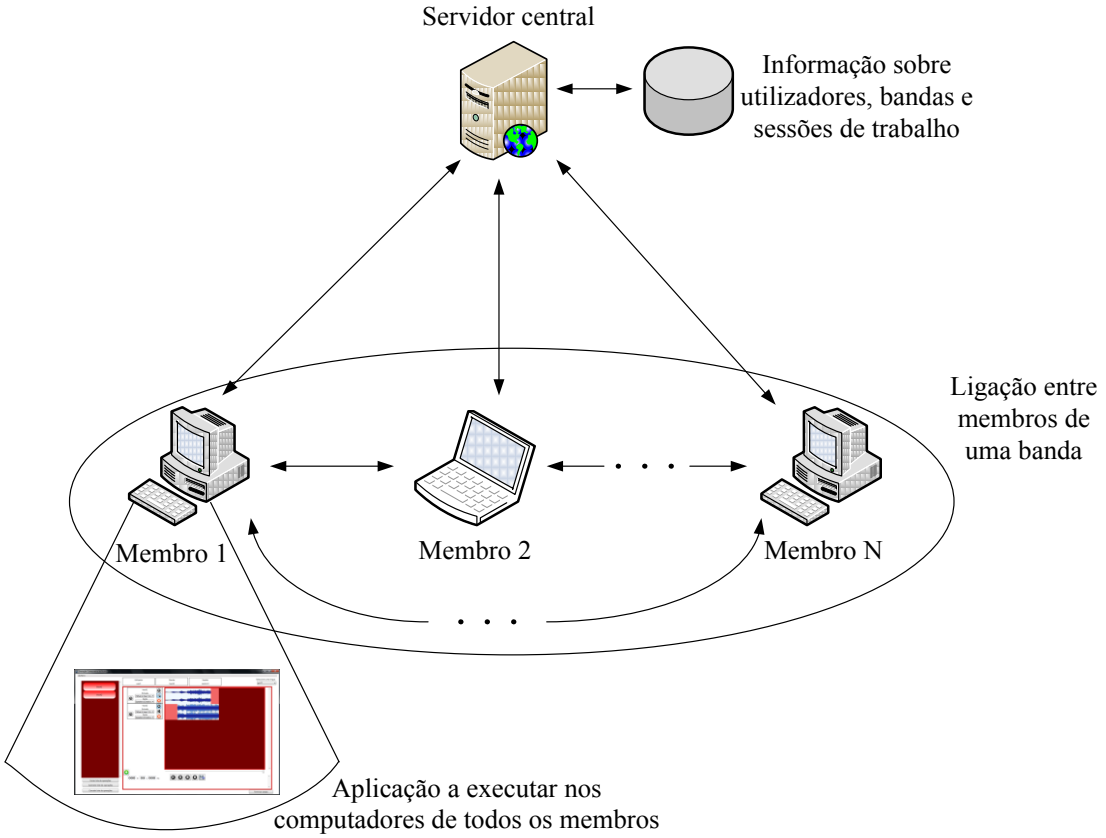

Figura 1 - Descrição global da solução

Como se pode ver na figura, a solução consiste numa aplicação distribuída com uma arquitectura *peer-to-peer* híbrida. O servidor central mantém informação acerca de utilizadores, bandas e sessões de trabalho (cada sessão corresponde a uma música). Este servidor funciona também como serviço de descoberta de *peers*. A arquitectura *peer-to-peer* híbrida minimiza o fluxo de pedidos ao servidor, sendo a maioria do tráfego realizado entre *peers*.

Cada membro da banda terá de instalar no seu computador a aplicação cliente, que terá uma interface gráfica para que este possa desenvolver o seu trabalho. Esta aplicação cliente será um *peer* no sistema distribuído e comunicará com os restantes enviando informação acerca das operações aplicadas e dos ficheiros áudio gravados. A aplicação cliente terá a lógica necessária para tratar da sincronização do trabalho realizado pelos vários membros de uma banda. No computador de cada membro é armazenada informação acerca das sessões de trabalho (uma por música) em que está envolvido. Esta informação é replicada em vários computadores, sendo necessária a sincronização dessas réplicas.

A aplicação cliente apresenta uma interface semelhante a aplicações de gravação e edição de áudio existentes, com o objectivo de ser familiar para utilizadores deste tipo de aplicações. A interface apresenta ainda alguns elementos que não se encontram noutras aplicações de gravação e edição de áudio, relativos à componente colaborativa deste trabalho.

1.3 - Organização do documento

O restante documento está organizado da seguinte forma:

- O Capítulo 2 formula o problema da edição colaborativa de áudio. É introduzida terminologia necessária para a compreensão do documento e são apresentados os diferentes elementos da solução desenvolvida tais como suporte a manipulação de áudio, suporte a trabalho colaborativo e interface gráfica.
- O Capítulo 3 apresenta a camada de manipulação de áudio. São descritas as funcionalidades desta camada e a forma como foram implementadas. Estas funcionalidades passam por gravação, reprodução, codificação e edição de áudio.
- O Capítulo 4 aborda o suporte ao trabalho colaborativo. É feita uma análise à forma de colaboração esperada na aplicação e, em sequência dessa análise, é descrita a solução adoptada para suportar a colaboração.
- O Capítulo 5 apresenta a aplicação cliente. Esta tem a interface gráfica a ser utilizada pelos membros da banda para desenvolver o seu trabalho. A implementação recorre aos elementos de suporte à manipulação de áudio e suporte ao trabalho colaborativo desenvolvidos.
- O Capitulo 6 apresenta as conclusões e trabalho futuro.

2 – Edição colaborativa de áudio

Este capítulo formula o problema da edição colaborativa de áudio. É introduzida terminologia necessária para a compreensão do documento e são apresentados os diferentes elementos da solução desenvolvida e suas funcionalidades.

A aplicação colaborativa para a edição de áudio tem três componentes, designadamente:

- Suporte à reprodução, gravação e edição de áudio.
- Suporte ao trabalho colaborativo.
- Interface com o utilizador.

A secção que se segue apresenta a terminologia adoptada no restante documento.

2.1 – Terminologia adoptada

No contexto deste trabalho são definidas 3 entidades, as quais são representadas por tipos de dados adequados:

- Bloco áudio – Consiste na peça de áudio, de duração variável, normalmente associada a um ficheiro de áudio. Por exemplo, fazendo a gravação do som que se obtém do microfone, o ficheiro obtido pode ser utilizado como um bloco áudio. Outro exemplo pode ser uma música lida a partir de Compact Disk (CD); esta será, a menos que subdividida pelo utilizador, um bloco áudio. Sobre os blocos áudio podem ser feitas edições, tais como por exemplo efeitos de eco ou amplificação do som. O bloco áudio é a unidade elementar usada na construção da música.
- Faixa áudio – É constituída por um ou vários blocos áudio (não sendo estes exclusivos a uma só faixa). Os blocos áudio estão normalmente relacionados entre si, do ponto de vista do utilizador; por exemplo, uma determinada faixa é utilizada para colocar vários blocos áudio gravados pelo mesmo instrumento.
- Sessão de trabalho – Representa a edição da música. A sessão é composta por várias faixas áudio, que ao serem reproduzidas em conjunto resultam na música. Na sessão existe ainda informação dos blocos de áudio que foram gravados e que podem ser usados nas faixas.

Em resumo, a sessão de trabalho é composta por várias faixas e blocos áudio. Os blocos áudio podem ser utilizados em várias faixas, tendo por isso as faixas referências para os blocos áudio que a compõem. Aos blocos áudio podem ser aplicadas edições, tais como por exemplo amplificação ou ecos. Na Figura 2 apresenta-se o esboço do modelo de dados que suporta os três tipos de dados descritos.

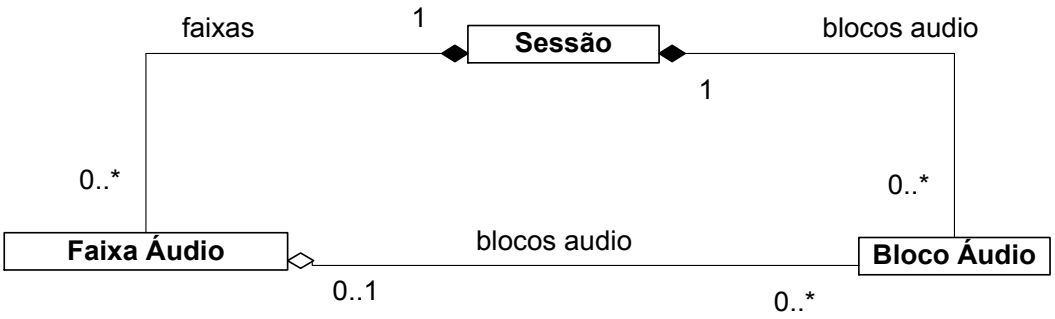

Figura 2 - Esboço do modelo de dados que relaciona as entidades sessão, faixa áudio e bloco áudio

Designa-se de colaborador cada utilizador da aplicação, inserido numa banda, que colabora no desenvolvimento de uma música. Por exemplo, uma banda pode utilizar a aplicação, cada membro no respectivo computador e/ou local, colaborando no desenvolvimento da mesma música.

2.2 – Elementos constituintes da aplicação

Na Figura 3 apresentam-se os elementos constituintes da aplicação: a camada de suporte à manipulação de áudio, a camada de suporte ao trabalho colaborativo e a aplicação cliente, que disponibiliza a interface gráfica para que os utilizadores desenvolvam o seu trabalho.

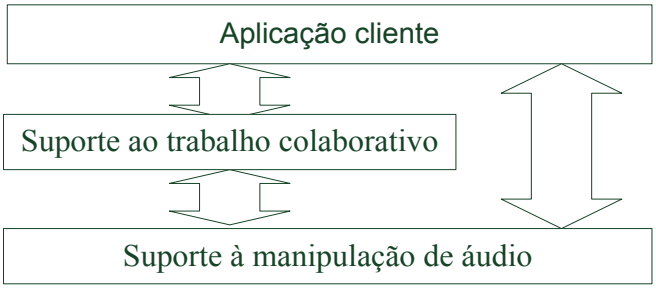

Figura 3 - Elementos constituintes da aplicação

2.2.1 - Suporte à reprodução, gravação e edição de áudio

A aplicação tem capacidade de reproduzir faixas separadamente ou em simultâneo, na forma como se apresentam na interface gráfica. Oferece ainda a possibilidade de escolher o dispositivo de saída (por exemplo se o computador estiver equipado com duas placas de som).

A gravação contempla a escolha de diferentes dispositivos de entrada, permitindo a utilização de diferentes instrumentos. O áudio gravado é guardado em ficheiro de formato Microsoft WAVE [4], resultante da digitalização do áudio em *Pulse Code Modulation (PCM*), normalmente utilizado neste tipo de *software* por ser um formato sem perda, resultante da digitalização do sinal na placa de som ou noutro dispositivo.

Quanto à edição, consideram-se algumas funcionalidades como:

- Analisar o sinal de áudio.
- Cortar e seleccionar partes de um ficheiro áudio.
- Amplificar ou diminuir volume a todo ou parte do ficheiro áudio.
- Aplicar alguns efeitos sobre o áudio tais como a possibilidade de fazer *fade in* ou *fade out*, *delay*, e*cho, reverb,* entre outros.

As operações sobre sinais necessárias neste trabalho consistem na composição de somas e multiplicações: somar blocos para reprodução simultânea; realização de amplificações e filtros do tipo *Finite Impulse Response (FIR*) e *Infinite Impulse Response (IIR*) [5] [6] para operações como *echo* e *reverb*. Para implementação destas funcionalidades, e em especial dos efeitos áudio, será utilizado como referência o livro *Introduction to Signal Processing*, de S. Orfanidis [5].

Tendo em conta a componente colaborativa do trabalho, surge a questão do tamanho dos ficheiros de áudio a processar. A edição colaborativa obriga à manipulação, processamento e armazenamento de diversos ficheiros áudio. Mesmo no cenário em que se considera apenas o armazenamento dos ficheiros, sem as implicações inerentes ao cenário de colaboração, a dimensão dos ficheiros é um factor a considerar.

Os ficheiros WAVE sem perda têm tipicamente dimensão elevada; como referência, um ficheiro WAVE de um minuto, com áudio *stereo*, 16 bits por amostra e frequência de amostragem de 44100 Hz, terá cerca de 10 MB, sendo que no desenvolvimento de uma música se utilizarão vários ficheiros destes. A dimensão destes ficheiros terá influência

em termos de capacidade de armazenamento (nas máquinas dos utilizadores ou num possível servidor) e no ritmo de transmissão necessário para assegurar a transmissão rápida dos ficheiros entre colaboradores. Por esta razão, é importante ter compressão dos ficheiros áudio, através de codificadores específicos para áudio, tais como por exemplo o Mpeg-1 Layer 3 (MP3) [7] (formato com perda), ou Free Lossless Audio Codec (FLAC) [8], (formato sem perda) ou codificadores universais tais como o GNU Zip (GZip) [9] ou o Zip [10].

Para desenvolver esta componente do trabalho utiliza-se a Plataforma .NET e a biblioteca de processamento de áudio BASS da empresa *Un4seen Developments* [11], disponível gratuitamente para utilizadores individuais. Outras bibliotecas consideradas foram DirectX (utilizando o DirectSound) [12] e FFmpeg [13] mas a biblioteca BASS aparenta ser de mais simples utilização e tem funcionalidades que satisfazem as necessidades do trabalho.

A biblioteca BASS pode ser usada em Windows (x86, x64 e CE), Mac OSX, Linux e iPhone. Suporta reprodução e edição de diversos formatos áudio (MP3, MP2, MP1, OGG, WAV, AIFF, XM, IT, S3M, MOD, MTM, UMX) e tem capacidades de gravação. Sobre o sistema operativo Windows, a biblioteca recorre a DirectX 3 ou superior e tira partido de *drivers* DirectSound e DirectSound3D se existirem. Em Mac OSX utiliza CoreAudio. Em conjunto com a biblioteca, existem APIs para C/C++, Delphi, Visual Basic, MASM, .NET e Java. Outra razão importante que justifica a escolha desta biblioteca é o facto de ter documentação completa sobre as operações que disponibiliza, bem como a existência de um fórum bastante participado para a resolução de dúvidas.

2.2.2 - Suporte ao trabalho colaborativo

Num programa para gravação e edição de áudio de forma colaborativa, a concorrência no acesso a dados e a necessidade de simultaneidade de colaboradores conectados (para propagação das alterações) são dois aspectos a ter em conta. Antes de detalhar acerca destes aspectos, passa-se à análise das operações disponíveis sobre cada tipo de dados, nomeadamente o bloco áudio, a faixa áudio e a sessão de trabalho apresentados na Secção 2.1.

2.2.2.1 Operações sobre os tipos de dados e concorrência

A uma sessão de trabalho pode-se adicionar/remover faixas e adicionar/remover blocos áudio (Figura 4).

Figura 4 - Operações sobre uma sessão

Numa faixa áudio, pode-se adicionar/remover blocos áudio, bem como alterar a sua disposição ao longo da faixa. Quando se pretende fazer nova gravação, esta será feita imediatamente no contexto de uma faixa, isto é, quando terminada a gravação, o bloco áudio correspondente será imediatamente colocado na faixa. A faixa pode também ser colocada sem som (modo *mute*). A Figura 5 ilustra as operações disponíveis sobre uma faixa áudio.

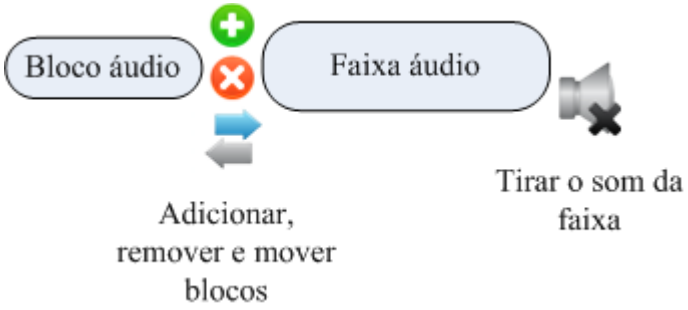

Figura 5 - Operações sobre uma faixa áudio

Um bloco áudio pode ser alvo de edições, como por exemplo a aplicação de efeitos, amplificações e cortes entre outras (Figura 6).

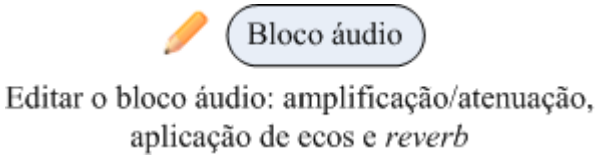

Figura 6 - Operação de edição de um bloco áudio - aplicação de amplificação/atenuação e efeitos

Tendo em conta as operações que se realizam sobre as várias unidades na criação da música, passa-se de seguida à análise dos problemas de concorrência inerentes.

Na sessão de trabalho, a adição de nova faixa não constitui dificuldade pois, sendo nova, não foi ainda partilhada com os restantes colaboradores. A remoção de uma faixa coloca problemas de concorrência, nos casos em que a faixa removida seja alvo de alterações por parte de outros colaboradores. A Figura 7 ilustra o conflito ocorrido entre a remoção da faixa áudio e a sua edição.

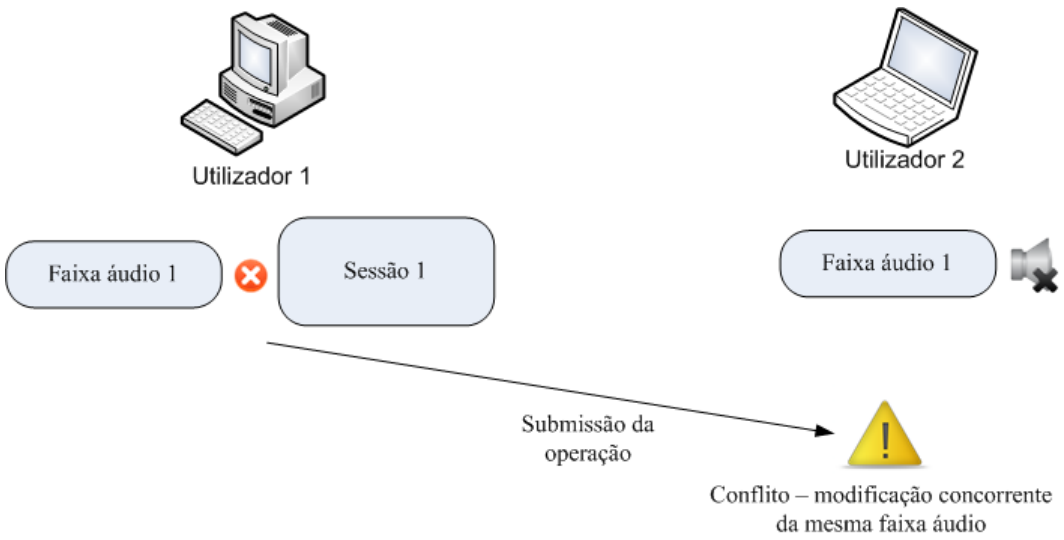

Figura 7 - Conflito na modificação de uma faixa áudio - removida por um utilizador e editada por outro

Quanto às faixas e blocos áudio, a concorrência pode acontecer em todas as operações, visto que qualquer delas estará a alterar algo que também pode ser alterado por outro colaborador. Além da concorrência no acesso a cada instância destes tipos de dados, alterações ao bloco áudio vão afectar a faixa (ou faixas) onde este está inserido. O mesmo acontece no caso das faixas inseridas na sessão. Na Figura 8 exemplifica-se um conflito na modificação de uma faixa áudio, em que um utilizador adiciona um bloco áudio e outro move um segundo bloco áudio já colocado na faixa. Na Figura 9 exemplifica-se um conflito na modificação de um bloco áudio, sendo este editado por 2 utilizadores.

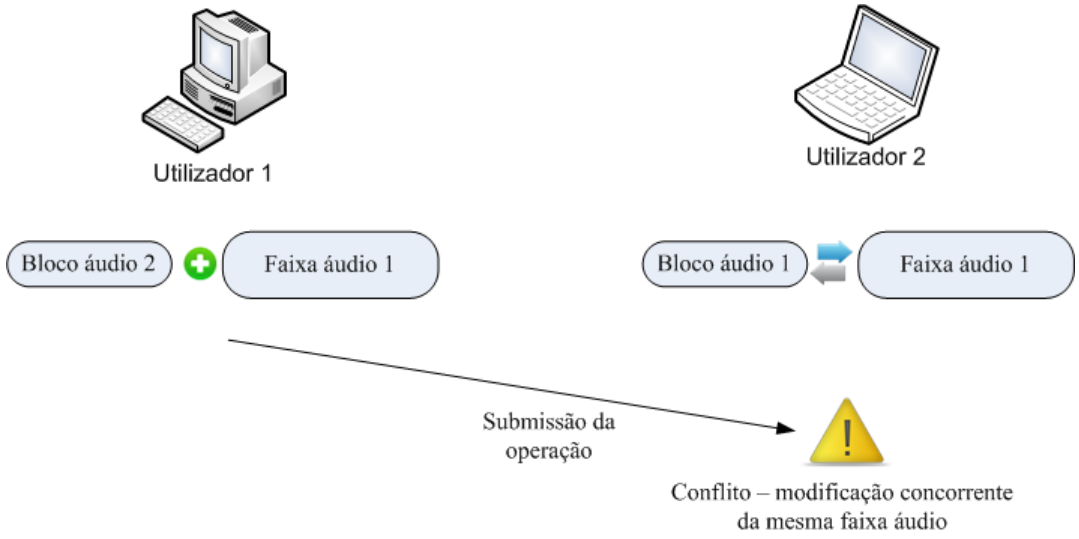

Figura 8 - Conlfito na modificação de uma faixa áudio - bloco adicionado por um utilizador e outro bloco movido por um segundo utilizador

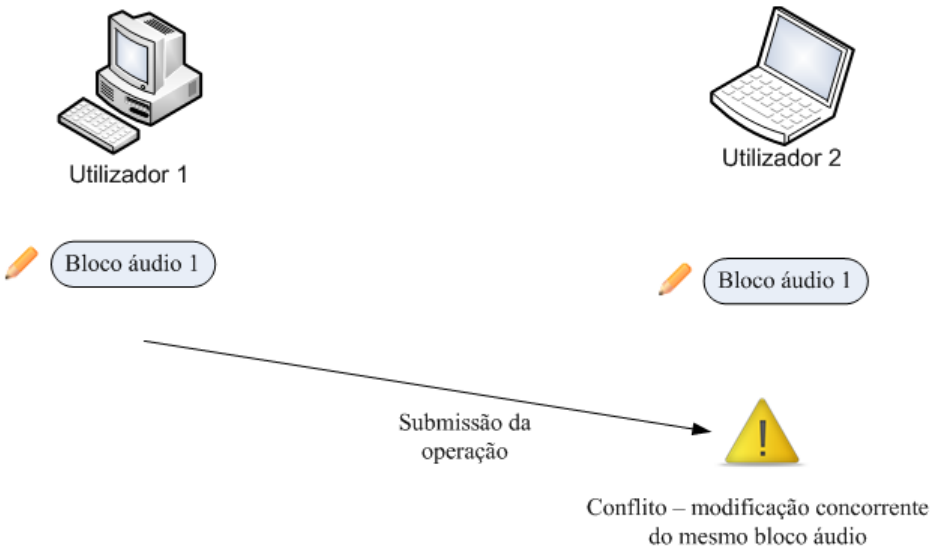

Figura 9 - Conflito na modificação de um bloco áudio - bloco editado por 2 utilizadores

Nos blocos áudio, as alterações vão afectar a forma como estes se encaixam nas faixas (por exemplo se forem removidos alguns segundos de som dos blocos, ou se for utilizada amplificação, que faça com que o bloco não produza o efeito esperado quando reproduzido com os restantes). Na Figura 10 exemplifica-se um conflito em que a edição de um bloco áudio por parte de um utilizador interfere com a utilização desse mesmo bloco áudio por parte de outro utilizador.

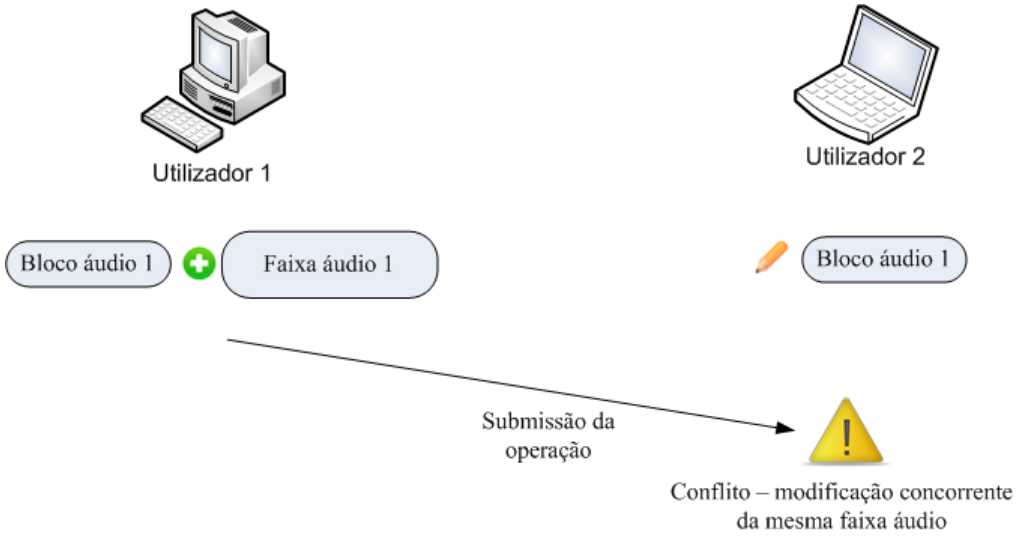

Figura 10 - Conflito na modificação de uma faixa e um bloco áudio – adicionado o bloco a uma faixa por um utilizador e editado por outro

2.2.2.2 Comunicação entre colaboradores

Outro aspecto a considerar é a comunicação do trabalho realizado aos restantes colaboradores. Sempre que um colaborador aplica operações à música, estas devem ser propagadas aos restantes colaboradores. Para lidar com esta questão existem duas soluções principais (não estando aqui descritas possíveis estratégias a empregar no desenvolvimento de cada uma delas):

- Utilização de repositório central.
- Manter a informação no computador de cada colaborador e enviá-la aos restantes quando estes se encontrarem conectados em simultâneo.

A utilização de repositório central é a solução ideal para acomodar o desfasamento temporal no trabalho dos colaboradores, isto é, nunca necessitariam de estar ligados em simultâneo.

A solução de manter a informação no computador de cada colaborador coloca o problema que o repositório central resolvia, isto é, a necessidade de os colaboradores estarem ligados em simultâneo. Assim, os colaboradores necessitam de estar ligados em simultâneo para que as operações realizadas sejam comunicadas entre eles. Esta solução deixa ainda prever a necessidade de existência de um servidor para funcionar como registo de ocorrências, para que os colaboradores saibam que certas informações se encontram desactualizadas.

Os principais problemas da primeira solução em relação à segunda é a necessidade de ter um repositório central com capacidade de armazenamento superior e existir um maior fluxo de pedidos ao servidor. Tem de guardar toda a informação das músicas dos vários colaboradores, incluindo os próprios ficheiros de som, que têm dimensão não desprezável, mesmo admitindo a utilização de compressão. Como exemplo considere-se um ficheiro WAVE de 1 minuto, com áudio *stereo*, 16 bit por amostra e frequência de amostragem 44100 Hz; este ficheiro ocupa cerca de 10 MB. Aplicando sobre este ficheiro uma compressão FLAC (sem perda) que no caso típico obteria compressão de cerca de 50%, tem-se um ficheiro de 5 MB. Admitindo uma sessão de trabalho com 10 blocos áudio, para manter 10 sessões de trabalho no servidor seriam precisos 500 MB. É esperado que o servidor tenha mais do que 10 sessões de trabalho, já que cada música corresponde a uma sessão de trabalho, a existência de vários utilizadores e bandas deixa prever uma maior quantidade de sessões de trabalho. O maior fluxo de pedidos ao servidor pode acontecer já que sempre que são feitas operações nas aplicações, estas são enviadas para o repositório central. Desta forma, o uso de compressão é adequado para obter redução do espaço de armazenamento, bem como do tempo de transmissão, aumentando a escalibilidade do sistema.

Do ponto de vista dos utilizadores, a primeira solução é melhor, pois além de não obrigar à simultaneidade, permite que um utilizador se conecte a partir de diferentes computadores, podendo descarregar imediatamente todo o trabalho do servidor.

2.2.3 – Interface com o utilizador

A interface com o utilizador deve ser feita num ambiente que apresenta a música como um conjunto de faixas, que contém blocos colocados em determinados períodos temporais. Este tipo de interface é o característico (e necessário) de programas de edição e gravação de música.

Como exemplo, mostra-se na Figura 11 o aspecto gráfico da aplicação *Adobe Audition 3* [3].

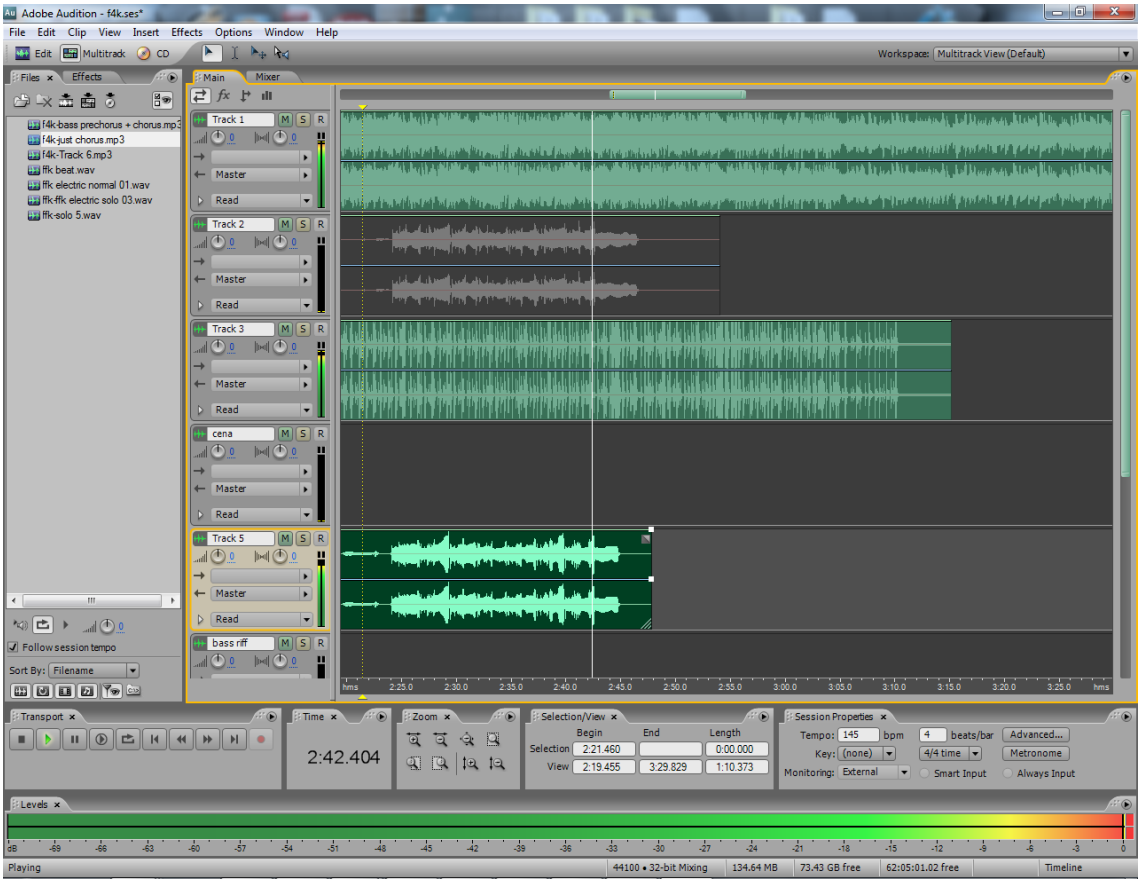

Figura 11 - *Adobe Audition 3*: exemplo de uma janela da aplicação.

Estabelecendo a correspondência entre os tipos de dados apresentados (sessão de trabalho, faixa áudio e bloco áudio) com a interface gráfica tem-se, implicitamente ao iniciar o desenvolvimento de uma nova música, uma sessão de trabalho. Na interface gráfica da aplicação existirá uma zona com os blocos áudio e outra zona com as faixas, nas quais serão colocados os blocos disponíveis na sessão.

Na Figura 12 observa-se como se inserem na interface os níveis de dados referidos. Estando implicitamente inserido numa sessão de trabalho, não existe nenhuma zona específica para a sessão. As faixas áudio apresentam-se na zona central da figura, com um menu rectangular à esquerda e os blocos áudio nele inseridos à direita. Os blocos áudio estão inseridos nas faixas, representados pelos rectângulos brancos com a representação em imagem do áudio a azul. Os blocos áudio aparecem ainda no menu à esquerda na aplicação, representados por rectângulos vermelhos e brancos, de forma a serem utilizados nas faixas.

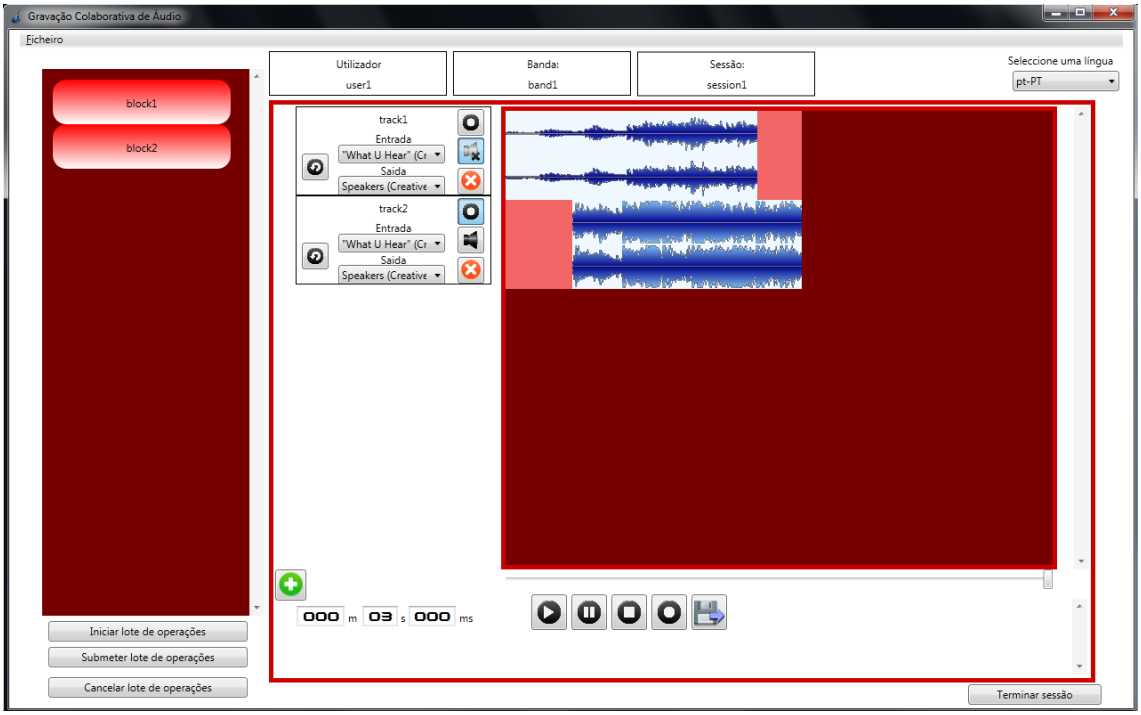

Figura 12 - Interface gráfica da aplicação desenvolvida

Outro aspecto interessante na interface é a possibilidade de escolher a língua, recorrendo para isso a *resource files* para conter as traduções, utilizando o ficheiro correcto tendo em conta a língua escolhida.

A interface gráfica é implementada na tecnologia Windows Presentation Foundation (WPF) sobre a Plataforma .NET. A escolha deve-se à familiaridade com o desenvolvimento sobre esta plataforma, adquirida ao longo do curso no ISEL. Neste contexto, WPF é a tecnologia actualmente divulgada pela Microsoft para o desenvolvimento de interfaces gráficas.

3 – Manipulação de áudio

Este capítulo apresenta a camada de manipulação de áudio. São descritas as funcionalidades desta camada e a forma como foram implementadas. A Figura 13 apresenta o elemento da aplicação implementada que é abordado neste Capítulo.

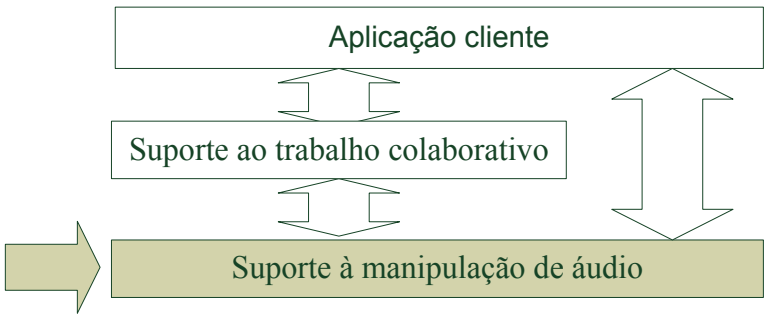

Figura 13 - Elementos constituintes da aplicação - camada de suporte à manipulação de áudio

As secções deste capítulo são:

- 3.1 – Testes de compressão – aborda o estudo realizado com o intuito de escolher os formatos áudio a utilizar na aplicação.
- 3.2 – Camada de manipulação de áudio – aborda a biblioteca desenvolvida para manipulação de áudio. É apresentada a forma como esta camada é utilizada, as suas funcionalidades e explica a sua implementação.

A camada de suporte à manipulação de áudio requer capacidades de:

- Reprodução e gravação de áudio utilizando diferentes dispositivos de entrada e saída.
- Edição de áudio, recorrendo à aplicação de efeitos. Estes efeitos podem ser obtidos com a aplicação de operações aritméticas (como a multiplicação por uma constante no caso da amplificação) e aplicação de filtros FIR e IIR para obtenção de efeitos como *echo* ou *reverb*.
- Compressão e descompressão de ficheiros WAVE [4] para outros formatos, nomeadamente FLAC [8] e MP3 [7].

Para definir os formatos áudio a utilizar, realizou-se um estudo onde foram feitos testes de desempenho de formatos de compressão.

3.1 – Testes de compressão de áudio

O formato WAVE [4] utilizado em aplicações de edição de áudio origina ficheiros com elevada dimensão, sendo estes ficheiros resultantes da digitalização do áudio em *Pulse Code Modulation* (PCM). Dado o número de ficheiros envolvidos e o número de sessões de trabalho típico e expectável, é importante tentar minimizar o tamanho dos ficheiros áudio utilizados, para poupar espaço em disco e reduzir a largura de banda necessária para a sua transmissão (tornando a transmissão mais rápida). Assim, é importante analisar a compressão sobre ficheiros PCM adquiridos a partir de dispositivos de gravação.

Para testar alguns métodos de compressão foram utilizados excertos de três músicas de estilos diferentes. De cada música foram utilizados um, dois e três minutos. Desta forma são testados os métodos de compressão com diferentes estilos de música e tamanhos, visto que estes dois factores podem influenciar a taxa de compressão. Os ficheiros estão no formato WAVE, *stereo* com 16 bit por amostra e frequência de amostragem de 44100 Hz.

Para comprimir os ficheiros foram utilizados três métodos: Zip [10], FLAC [8] e MP3 [7]. A codificação Zip é uma técnica de codificação universal sem perda, tal como utilizado em aplicações como o WinZip e WinRar. No contexto de edição de áudio a possibilidade de comprimir os ficheiros sem perda de qualidade é importante, para que o áudio não se vá deteriorando durante as sucessivas edições no desenvolvimento da música.

O formato MP3 foi considerado nestes testes, mas a sua utilização durante a fase de desenvolvimento das músicas não é recomendada, visto que é um formato com perda de qualidade. Este formato é no entanto pertinente pois é o mais conhecido na partilha de música pela internet, podendo ser usado quando o desenvolvimento da música está concluído e se pretende partilhar e divulgar o resultado final.

Na compressão através de Zip, é utilizada uma biblioteca que funciona como *wrapper* para .NET da biblioteca ZLib [14]. Na compressão FLAC e MP3, é utilizada a biblioteca BASS [11] em conjunto com codificadores de linha de comandos de FLAC [8] e LAME (LAME Aint MP3 Encoder) [15], respectivamente. No formato Zip e em FLAC foram escolhidos os parâmetros para obter compressão máxima. Em MP3 foi utilizado *Variable Bit Rate* (*VBR*) com ritmo binário mínimo 128 kbit/s e máximo 192

kbit/s. O ritmo binário 128 kbit/s aproxima-se da qualidade CD áudio, do ponto de vista da percepção áudio por parte do ser humano.

Na Tabela 1 mostram-se os resultados dos testes efectuados, contendo o tamanho original dos ficheiros áudio utilizados, o tamanho dos ficheiros comprimidos e a razão de compressão (percentagem do tamanho do ficheiro comprimido em relação ao original) obtida de acordo com (1).

$$Razão = 100 * \frac{Tamanho\ Comprimido}{Tamanho\ Original} \quad (1)$$

		Espaço ocupado em disco (em MB)				Razão de compressão		
Música	Parte do áudio utilizada (aproximado)	Original	Zip	FLAC	MP3	Zip	FLAC	MP3
Banda da Armada Portuguesa - Lisboa a Noite	Do minuto 4 a 5	10,1	9,5	5,5	1,1	94,06%	54,46%	10,89%
	Do minuto 4 a 6	20,2	19,4	12,1	2,1	96,04%	59,90%	10,40%
	Do minuto 4 a 7	30,3	29,3	18,2	3,1	96,70%	60,07%	10,23%
Metallica - Seek & Destroy	Do minuto 0 a 1	10,1	9,4	6,4	1,2	93,07%	63,37%	11,88%
	Do minuto 0 a 2	20,2	19,1	13,5	2,3	94,55%	66,83%	11,39%
	Do minuto 0 a 3	30,3	28,7	20,5	3,4	94,72%	67,66%	11,22%
Stone Sour - Through Glass	Do minuto 0 a 1	10,1	9,5	5,8	1	94,06%	57,43%	9,90%
	Do minuto 0 a 2	20,2	19,2	12,3	2,1	95,05%	60,89%	10,40%
	Do minuto 0 a 3	30,3	29,2	19,9	3,1	96,37%	65,68%	10,23%
		Razão de compressão média obtida				94,96%	61,81%	10,73%

Tabela 1- Resultados dos testes de compressão

Como se pode observar na Tabela 1, MP3 é, como esperado, o método de compressão que obtém os ficheiros mais pequenos. Zip obtém resultados pouco interessantes (razão de compressão média de 95%), o que se justifica pelo facto de ser um codificador universal e como tal não explora as características específicas do ficheiro áudio; por outro lado, o FLAC, sendo especializado para áudio, obtém uma compressão razoável, em que os ficheiros comprimidos apresentam uma razão de compressão média de 62%.

Os resultados do codificador MP3 indicam que a dimensão final do ficheiro após processamento e distribuição (da versão final) é cerca de 10% da dimensão original sem compressão.

3.2 – Camada de manipulação de áudio

3.2.1 – Contrato entre aplicação e biblioteca de manipulação de áudio

O desenvolvimento da camada de manipulação de áudio está assente num conjunto de componentes de software nomeadamente, a biblioteca BASS [11], codificadores FLAC [8] e LAME [15]. Dado este compromisso feito na biblioteca com outros componentes, torna-se importante minimizar a dependência entre a biblioteca e a aplicação.

A importância de diminuir a dependência entre a aplicação e a biblioteca prende-se com a facilidade que isso traz para a substituição da biblioteca por outra que tenha as mesmas funcionalidades, mas que difira na implementação. Desta forma, a substituição da biblioteca ocorre com poucas alterações à aplicação.

De forma a minimizar a dependência, optou-se por desenvolver outra biblioteca (AudioLibraryContract.dll) com o contrato que qualquer biblioteca de manipulação de áudio passível de ser utilizada na aplicação tem que cumprir.

A biblioteca que implementa o contrato será a única comprometida com a biblioteca BASS e outros componentes específicos à implementação. A Figura 14 apresenta os componentes da aplicação dispostos em camadas. No topo temos qualquer aplicação que pretenda utilizar a camada de manipulação de áudio. A fazer a ligação entre a aplicação e a implementação da biblioteca de manipulação de áudio, tem-se uma biblioteca que estabelece o contrato. A implementação recorre aos elementos externos biblioteca BASS e codificadores FLAC e LAME.

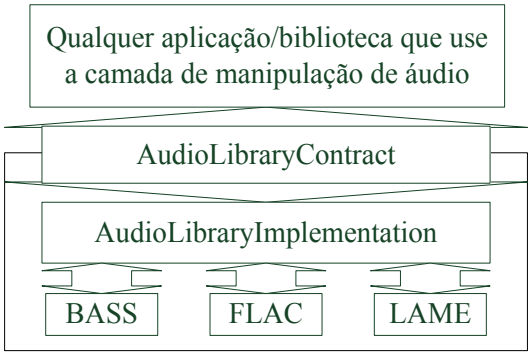

Figura 14 - Componentes da aplicação *desktop*

De seguida apresentam-se as classes e interfaces desenvolvidas para o contrato da biblioteca de manipulação de áudio.

A interface **ILibraryManagement**, na Figura 15, expõe métodos que devem ser chamados no início e no fim da utilização de um programa que use uma biblioteca que implemente o contrato definido. Estes métodos servem para realizar algum tipo de código de iniciação ou término, como obtenção e libertação de recursos necessários.

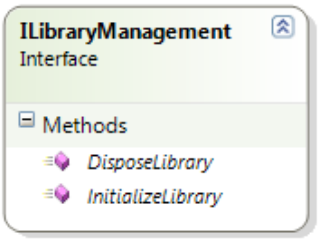

Figura 15 - Interface ILibraryManagement

A hierarquia **IDevice** na Figura 16, representa os dispositivos ligados ao computador, como por exemplo microfone e as colunas ligadas à placa de som, entre outros. A interface **IDevicesGetter**, também na Figura 16, expõe métodos que permitem obter os dispositivos existentes no computador onde o programa é executado. São obtidos os dispositivos de entrada e saída de áudio, sendo também possível obter o dispositivo utilizado por omissão, quer para entrada quer para saída.

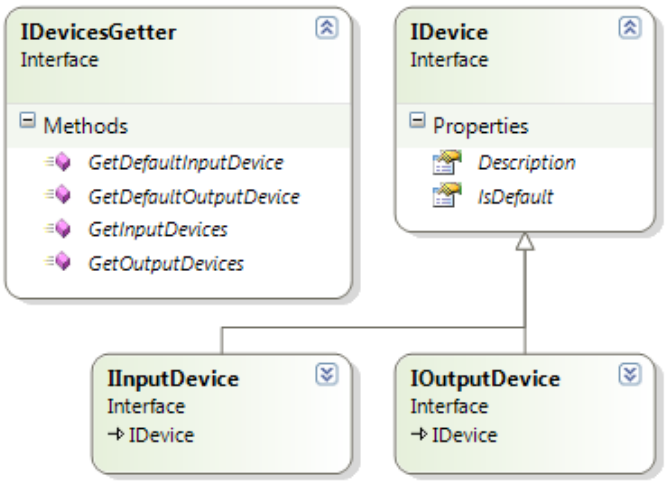

Figura 16 - IDevice e IDevicesGetter

A interface **IAudioBlock** representa um bloco áudio, isto é, um excerto áudio que pode ter sido gravado ou importado a partir de ficheiro existente. Um objecto deste tipo é utilizado para manter em memória um bloco áudio em uso na aplicação, de forma a ser usado durante gravações, reproduções e misturas. Também como parâmetro de codificação ou retorno de descodificação de áudio.

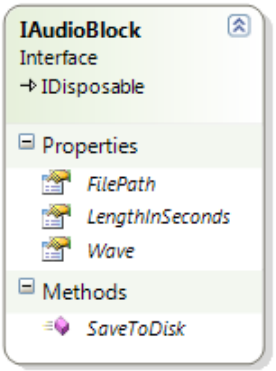

Figura 17 - Interface IAudioBlock

A hierarquia **IAudioEncoder**, representa um codificador, com duas interfaces que estendem os métodos de **IAudioEncoder**, uma interface com parametrização específica de FLAC e outra para MP3. A hierarquia **IAudioDecoder** é em tudo semelhante a **IAudioEncoder** mas para representar um descodificador. Apresentam-se estas hierarquias na Figura 18 e Figura 19 respectivamente.

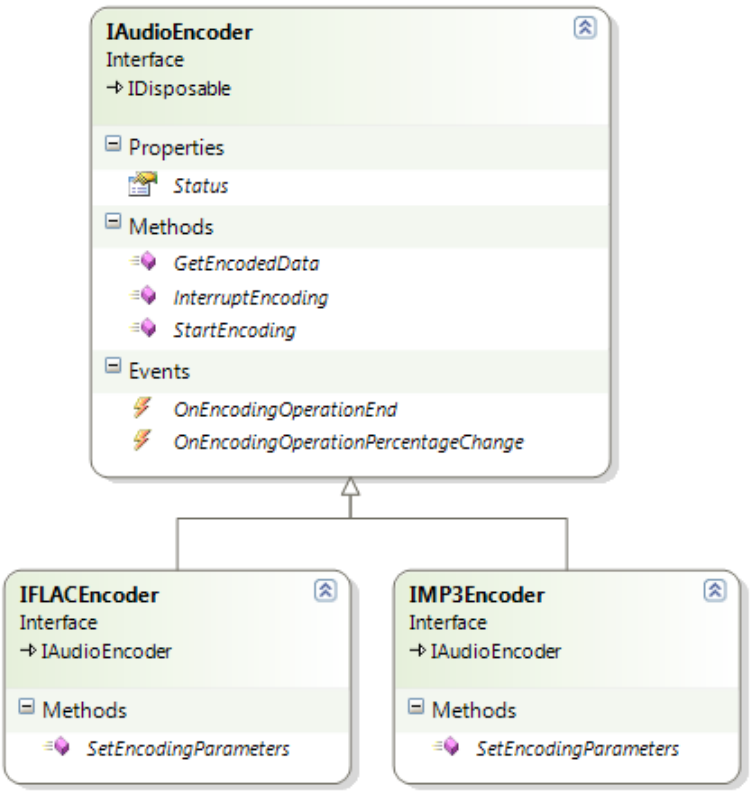

Figura 18 - Hierarquia IAudioEncoder

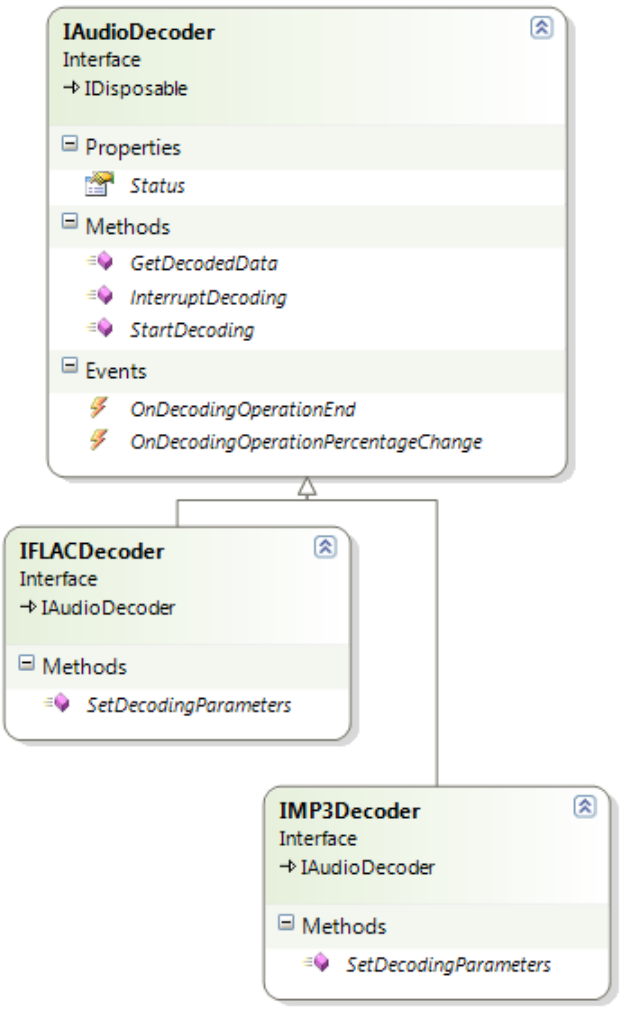

Figura 19 - Hierarquia IAudioDecoder

A interface **IMultipleDeviceAudioRecorder** representa um reprodutor/gravador. Uma implementação desta interface deve permitir a reprodução de múltiplos blocos áudio, através de diferentes dispositivos de saída. Deve ainda permitir a gravação de áudio a partir de múltiplos dispositivos de entrada. A interface **IMultipleDeviceAudioRecorder** pode ser observada na Figura 20 e um exemplo de utilização deste reprodutor/gravador pode ser observado na Figura 21.

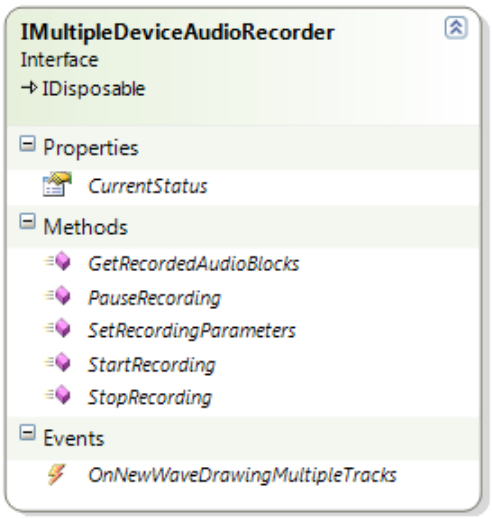

Figura 20 - Interface IMultipleDeviceAudioRecoder

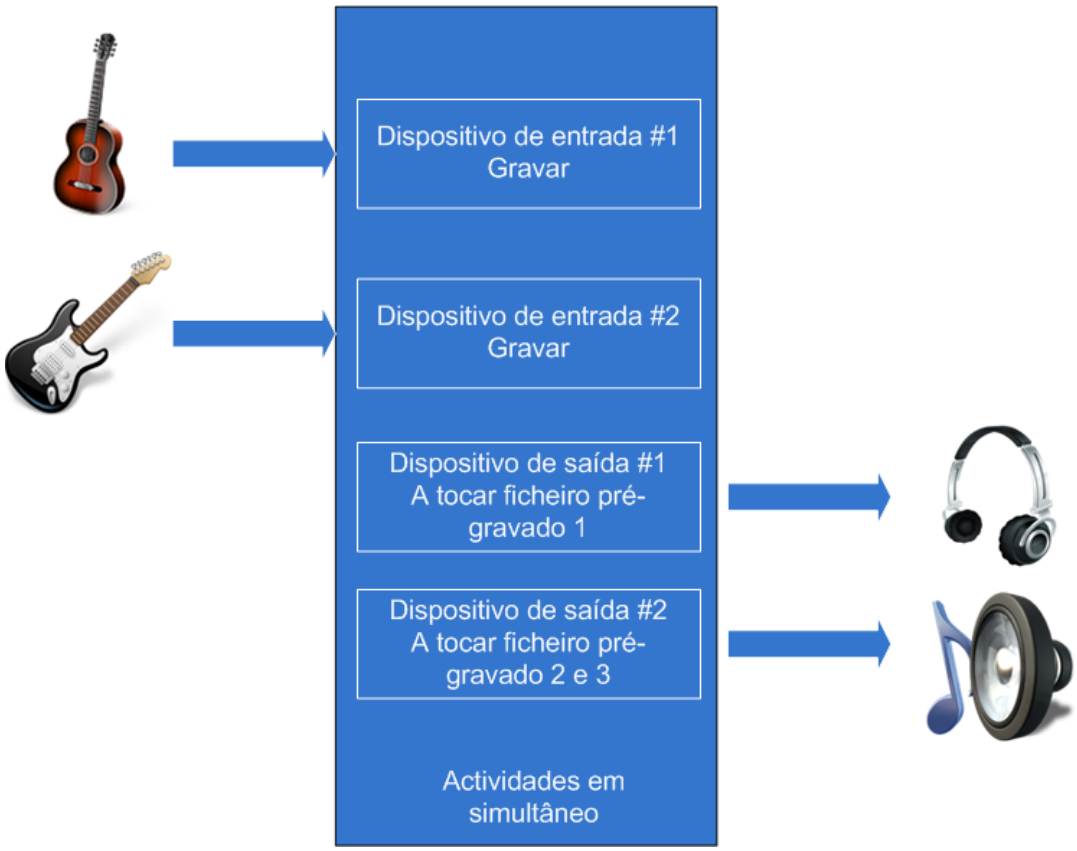

Figura 21 - Exemplo de funcionamento de reprodutor/gravador multi-faixa

A interface **IMultipleDeviceAudioPlayer**, na Figura 22, representa um reprodutor de áudio. Uma implementação desta interface deve permitir a reprodução de múltiplos blocos áudio, através de diferentes dispositivos de saída. A Figura 21 também aqui se aplica do ponto de vista da reprodução (na figura, os 2 últimos blocos respeitantes a

dispositivos de saída) ignorando a componente de gravação (dispositivos de entrada, os 2 primeiros blocos da figura).

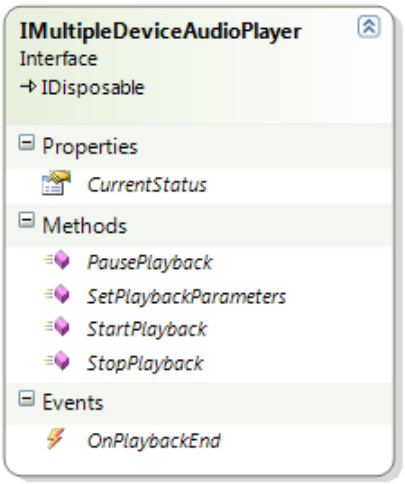

Figura 22 - Interface IMultipleDeviceAudioPlayer

A interface **IAudioMixer**, na Figura 23 representa um sequenciador de blocos áudio. Como parâmetros de entrada recebe vários blocos áudio e a sua colocação ao longo do tempo da música. Como saída ter-se-á um único bloco áudio resultante da mistura dos blocos áudio passados como parâmetro.

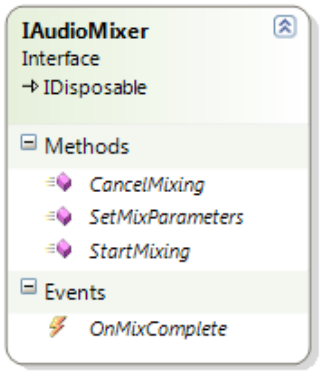

Figura 23 - Interface IAudioMixer

A interface **IWaveDrawingInformation**, na Figura 24, representa a informação necessária para obter uma representação em imagem de um bloco áudio. Expõe métodos que permitem obter essa imagem em formato Bitmap [16]. A interface **IWaveDrawer**, também na Figura 24, expõe um método que permite, dado um bloco áudio gerar a informação acerca da sua representação gráfica (retornando uma instância de uma classe que implementa **IWaveDrawingInformation**).

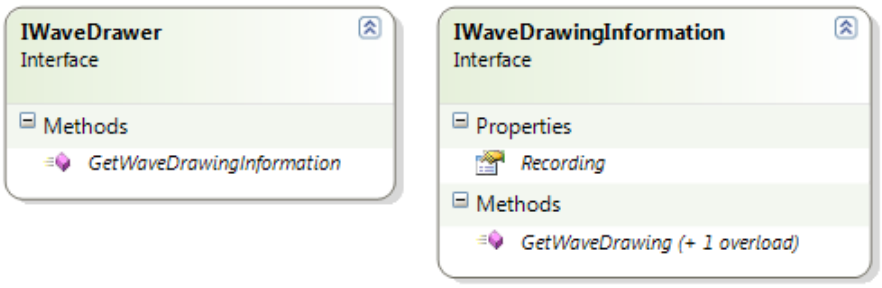

Figura 24 - Interfaces IWaveDrawer e IWaveDrawingInformation

Para gerar a imagem que representa o bloco áudio, são utilizadas definições da classe estática **WaveDrawingSettings**, na Figura 25, onde estão informações como as cores para a imagem, o tamanho em *pixels* de altura e largura de um segundo de áudio.

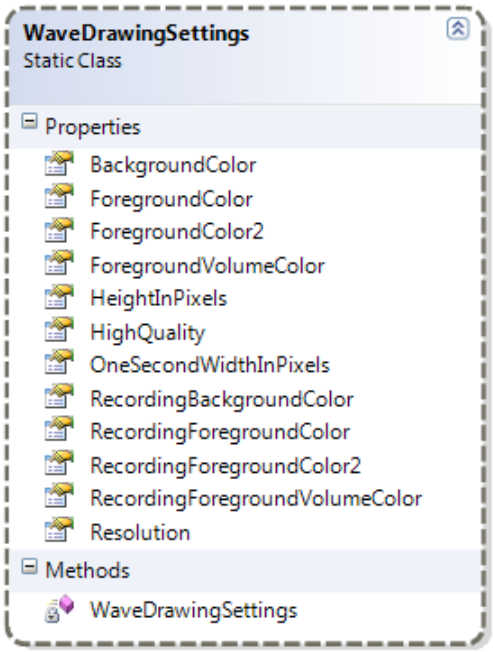

Figura 25 - Classe WaveDrawingSettings

A interface **IAudioEditor** expõe métodos para a edição de blocos áudio. Para efectuar estas edições foi adoptado um modelo em que se obtém um objecto do tipo **IAudioEditionComponent** que implementa determinada operação sobre o áudio. Obtido esse objecto, a invocação do método ApplyAudioEditionComponent aplicará a operação, recebendo como parâmetro o áudio a editar e o objecto de edição. A interface **IAudioEditionComponent** disponibiliza o método ApplyEditionToSample. Este método é invocado para cada amostra do bloco áudio, aplicando as operações necessárias e guardando num novo *array* de *bytes* de saída. Podem-se observar estas duas interfaces na Figura 26.

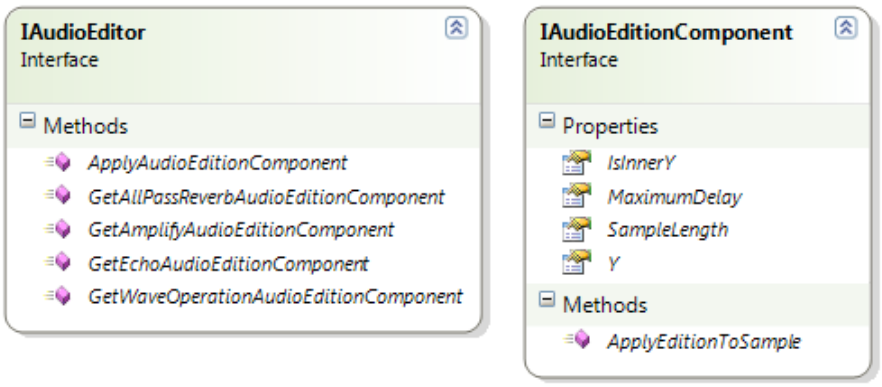

Figura 26 - Interfaces IAudioEditor e IAudioEditionComponent

Para obter instâncias das interfaces descritas, existe a classe **AudioLibraryFactory**, na Figura 27. Recebendo como parâmetro no construtor o caminho em disco para um *assembly* que implemente as interfaces expostas pela biblioteca, obtém por introspecção os construtores das classes que implementam as interfaces expostas. Quando é necessária uma instância de um desses tipos, passa-se como parâmetro ao método *GetObjectInstance* a interface cuja instância da implementação se pretende.

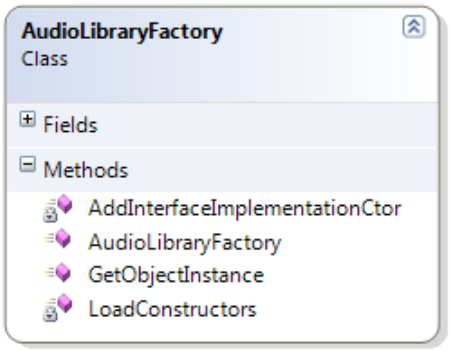

Figura 27 - Classe AudioLibraryFactory

Nem sempre as instâncias podem ser obtidas através desta classe. Por exemplo, uma instância de uma implementação de **IMultipleDeviceAudioPlayer** pode ser obtida desta forma. Já uma instância de **IWaveDrawingInformation** não o poderá ser, sendo obtida apenas através de retorno de métodos como **IWaveDrawer.** *GetWaveDrawingInformation*.

3.2.2 - Implementação

Dadas as interfaces apresentadas na secção 3.2.1, de seguida discutem-se alguns aspectos de implementação.

3.2.2.1 - Reprodutor/gravador (IMultipleDeviceAudioRecorder)

O reprodutor/gravador multi-faixa e multi-dispositivo (implementação de **IMultipleDeviceAudioRecorder**) permite reproduzir vários blocos áudio em diferentes dispositivos de saída e gravar a partir de diferentes dispositivos de entrada. Para realizar esta implementação utilizam-se as capacidades de gravação e reprodução da biblioteca BASS [11].

A parametrização do reprodutor/gravador recebe:

- O tempo em que deve começar a reprodução.
- A colocação dos blocos áudio a reproduzir, a sua posição na música e o dispositivo de saída a utilizar na sua reprodução.
- As gravações a efectuar (blocos áudio que vão ser criados) e os dispositivos de entrada a ser utilizados.

A chamada ao método Start inicia o processo de gravação. O processo de gravação é terminado com a chamada ao método Stop. No fim da gravação obtêm-se os blocos áudio gravados através do método GetRecordedAudioBlocks.

Ao longo da gravação é gerado o evento *OnNewWaveDrawingMultipleTracks*. Como parâmetros são passados uma representação em imagem do áudio gravado e a faixa a que esta gravação pertence. Isto permite no caso em que assim se pretenda numa interface gráfica, se possa apresentar a imagem do áudio durante a gravação.

A biblioteca BASS [11] possui o conceito de canal. Na implementação do gravador/reprodutor são utilizados 3 tipos de canal gravação, reprodução e mistura.

Um canal de gravação corresponde a uma gravação, resultando num bloco áudio. O número de canais de gravação utilizados depende da parametrização do gravador.

Um canal de reprodução corresponde a um bloco áudio a ser reproduzido. Por cada entrada de bloco áudio a reproduzir passada como parâmetro (por cada entrada, pois um bloco áudio pode ser passado mais que uma vez, para reproduzir em momentos diferentes da música) é iniciado um canal de reprodução.

Um canal de mistura funciona como agregador dos canais de reprodução, sendo aqui definida a altura em que cada um destes canais inicia efectivamente a reprodução. É iniciado um canal de mistura por cada dispositivo de saída que seja parametrizado. Os

canais de reprodução são adicionados ao canal de mistura associado ao dispositivo de saída correcto, tendo em conta a parametrização do gravador/reprodutor.

Nos casos em que se utilizam vários dispositivos de entrada e de saída em simultâneo, o início da reprodução/gravação dos canais utilizados não é atómico. Dada a velocidade a que as chamadas de iniciação dos canais, e que apenas a partir dos 20/30 milissegundos é que a diferença é perceptível, se forem usados poucos dispositivos (cerca de 5, já sendo uma situação pouco esperada) a diferença no tempo de iniciação não será perceptível para o ouvido humano. Para que esta diferença seja a mais baixa possível tornando-se imperceptível, a *thread* que faz iniciação dos canais não deve perder o processador. Por este motivo a prioridade desta *thread* é elevada para o máximo, tendo em conta que este máximo depende da prioridade base do processo.

3.2.2.2 – Reprodutor de áudio (implementação de IMultipleDeviceAudioPlayer)

A implementação do reprodutor de áudio multi-faixa e multi-dispositivo é semelhante à implementação do reprodutor/gravador, ignorando a componente de gravação.

No caso do reprodutor de áudio, ao contrário do gravador que apenas termina quando ordenado, pode terminar sem que seja chamado o método *Stop*. Quando isto acontece é gerado o evento *OnPlaybackEnd*.

3.2.2.3 – Sequenciador de áudio (implementação de IAudioMixer)

O misturador de áudio permite misturar vários blocos áudio obtendo apenas um como resultado final.

Este é implementado recorrendo à biblioteca BASS [11]; a implementação tem semelhanças com a implementação do reprodutor áudio, sendo que neste caso o resultado não é reproduzido mas sim guardado e retornado como um novo bloco áudio.

Para obter este resultado, são iniciados tantos canais de reprodução como necessários. Todos estes canais serão adicionados ao mesmo canal de mistura, que será adicionado a um dispositivo de saída “sem som”. Sobre este canal de mistura é possível obter os *bytes* que compõem o áudio, ou seja, os vários blocos áudio (através dos canais de reprodução) misturados.

3.2.2.4 - Codificadores e descodificadores de áudio (hierarquias IAudioEncoder e IAudioDecoder)

O desenvolvimento dos codificadores utiliza as aplicações flac.exe (codificador FLAC [8]) e lame.exe (codificador LAME [15]). Em ambos os casos a parametrização incide sobre a qualidade do ficheiro comprimido a obter o que influenciará também o seu tamanho em disco. A codificação é iniciada (método *Start*), sendo o processamento feito numa *thread* diferente da *thread* responsável pela interface gráfica, para que esta não bloqueie. É utilizada uma *thread* do *thread pool* para executar esta operação, de forma a não bloquear a *thread* da interface gráfica durante o processamento. O registo no evento *OnEncodingOperationEnd* resultará na notificação de término da codificação. Caso contrário tem de ser feito *polling* sobre o estado (propriedade Status) do codificador. O registo no evento *OnEncodingOperationPercentageChange* fará com que o registado receba notificações de alteração da evolução da codificação.

Os descodificadores são implementados da mesma forma, recorrendo também às aplicações flac.exe e lame.exe, fazendo a descodificação numa thread do *thread pool*. Neste caso existem os eventos *OnDecodingOperationEnd* e *OnDecodingOperationPercentageChange*, análogos aos eventos dos codificadores.

Para realizar a codificação ou descodificação, é lançado um processo (flac.exe ou lame.exe). Este processo é de seguida alimentado com os *bytes* do ficheiro a codificar/descodificar.

A utilização de uma aplicação externa para a codificação/descodificação deve-se ao facto das bibliotecas disponíveis para este fim serem implementadas em código *unmanaged*. De forma a evitar a utilização de uma aplicação externa, é necessário utilizar as ferramentas de interoperabilidade disponíveis em .NET, o que não foi realizado neste trabalho

3.2.2.5 – Editor de áudio (implementação de IAudioEditor)

A implementação de **IAudioEditor** aplica sobre os blocos áudio as edições disponibilizadas. Neste trabalho foram desenvolvidos 4 tipos de edições: amplificação/atenuação, multiplicação por onda (sinusoidal, triangular ou quadrada) e aplicação de ecos (que com a parametrização correcta, permite obter alguns tipos de *reverb*) e *all-pass reverberator*.

Todos os efeitos são aplicados passando uma instância de uma classe que implementa a interface **IAudioEditionComponent** ao método ApplyAudioEditionComponent. Este método recebe também o bloco áudio a editar e os tempos de início e fim de aplicação da edição. É executado um ciclo para todas as amostras a editar, executando o método ApplyEditionToSample do objecto do tipo **IAudioEditionComponent**.

A amplificação/atenuação é implementada com a classe **AmplifiyEditionComponent**. Esta edição consiste na multiplicação da amostra recebida no método ApplyEditionToSample por um valor recebido por parâmetro aquando da criação do componente de edição (chamada ao método GetAmplifyAudioEditionComponent da classe **AudioEditor**).

A multiplicação por uma onda é implementada com a classe **WaveOperationEditionComponent**. Esta edição consiste na multiplicação da amostra recebida no método ApplyEditionToSample por um valor obtido de uma onda, indexada pelo índice da amostra subtraído do índice da primeira amostra a editar. As ondas são representadas pelas classes **SinusWave**, **SquareWave** e **TriangleWave**, que implementam a interface **ICalculatedWave**. Esta interface define um *indexer*, através do qual se obtém o valor a multiplicar pela amostra.

A aplicação de ecos recorre a um filtro IIR, implementado pela classe **IIREditionComponent**. O filtro IIR divide-se em duas componentes: a componente directa a qual incide sobre o sinal de entrada e a componente de realimentação sobre o sinal de saída, à medida que este vai sendo calculado. Em (2) apresenta-se a equação às diferenças de um filtro IIR definido pelos coeficientes b_k e a_k.

$$y[n] = \sum_{k=0}^{M} b_k . x[n-k] + \sum_{k=1}^{N} a_k . y[n-k] \quad (2)$$

Para cada componente do filtro (sobre o sinal de entrada e sobre a saída) é criado uma colecção de pares "chave-valor", indicando os valores de atraso e correspondentes factores de multiplicação pela amostra. Relativamente aos valores dos coeficientes do filtro IIR, é necessário que os coeficientes a_k sejam tais que o sistema resultante seja estável [5]. Caso o sistema não seja estável, produzem-se efeitos indesejados sobre o sinal tais como saturação do mesmo e "estalidos".

Variando a parametrização da aplicação de eco, é possível obter diferentes e interessantes resultados, com mais ou menos ecos, limitados (usando apenas a componente que incide sobre o sinal de entrada, que é equivalente a um filtro FIR) ou com ecos teoricamente infinitos (usando a segunda componente do filtro), que na prática acabam por deixar de se ouvir com a atenuação do sinal.

De acordo com [5], é possível obter alguns tipos de *reverb* apenas com a parametrização correcta do filtro IIR. Dois exemplos de *reverb* são o *all-pass reverberator* e o *plain reverberator*.

O *all-pass reverberator* consiste num filtro IIR com as duas componentes, em que a primeira tem duas entradas: sem qualquer atraso, com um factor de 'a' negativo (em que 'a' é um valor, em módulo, entre 0 e 1) e com um atraso 'D', com factor 1 (manter a amostra intacta). A segunda componente tem apenas uma entrada, com o mesmo atraso 'D', com um factor 'a'. Este tipo de *reverb* é implementado pela classe **AllPassReverberatorEditionComponent**. Em (3) apresenta-se a equação às diferenças do *all-pass reverberator*.

$$y[n] = a.y[n - D] - a.x[n - D] \quad (3)$$

O *plain reverberator* faz uso apenas da segunda componente do filtro (sendo o sinal de entrada a saída do primeiro componente), com uma entrada com um valor de atraso 'D' e factor 'a' em módulo entre 0 e 1. Este tipo de *reverb* é implementado pela classe **PlainReverberatorEditionComponent**. Em (4) apresenta-se a equação às diferenças do *plain reverberator*.

$$y[n] = a.y[n - D] + x[n] \quad (4)$$

4 - Suporte a trabalho colaborativo

Este capítulo aborda o suporte ao trabalho colaborativo. Começa por ser feita a análise à forma de colaboração esperada na aplicação (Secção 4.1). Em sequência desta análise, é descrita a solução adoptada para suportar a colaboração na Secção 4.2 e apresentada a respectiva implementação nas secções 4.3 e 4.4 Na Figura 28 está salientada (fundo caqui) a camada da aplicação abordada neste capítulo.

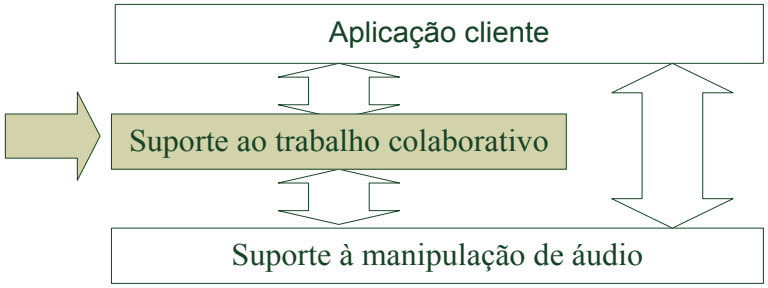

Figura 28 - Elementos constituintes da aplicação - camada de suporte ao trabalho colaborativo

4.1 – Análise da colaboração na aplicação

Para fazer a escolha da solução a empregar na definição da arquitectura da aplicação e no desenvolvimento dos seus elementos, é necessário fazer uma análise da forma como se prevê que a aplicação seja utilizada. A forma de colaboração por parte dos utilizadores terá impacto na escolha de soluções como:

- Arquitectura com servidor central ou *peer-to-peer*.
- Técnicas pessimistas ou optimistas de gestão de réplicas.

Uma banda será composta em média por 4/5 elementos que colaborarão, em locais diferentes, no desenvolvimento das músicas. Espera-se que estejam a trabalhar em simultâneo a maioria dos elementos da banda, assumindo que na banda todos fazem parte do processo de criação da música. Neste caso, é possível a adopção de uma arquitectura *peer-to-peer*, sendo feita a propagação dos dados entre colaboradores. Se por outro lado se adoptar um cenário que os colaboradores nunca trabalham em simultâneo, esta solução já não será viável, sendo necessário um repositório central para armazenar os dados de todos os colaboradores.

Utiliza-se como exemplo uma banda típica (apesar da composição das bandas poder diferir bastante), com 5 elementos em que existe um vocalista, dois guitarristas, um baixista e um baterista. Cada colaborador terá um papel diferente no desenvolvimento da música, por isso é pouco esperado que o colaborador efectue alterações sobre partes da música criadas por outro colaborador (acessos de escrita aos dados criados por outro colaborador). Por exemplo, um guitarrista não irá alterar nada do que o baterista fez, apenas criar algo com a guitarra tendo em conta o som de bateria gravado, ou seja, necessita apenas de um acesso de leitura aos dados do baterista. Este comportamento esperado permite que se adopte uma abordagem optimista na distribuição dos dados, isto é, os colaboradores podem não ter a cada momento, a versão mais actual dos dados.

Na colaboração é plausível assumir que os membros da banda conversam, por exemplo informando os outros que vão realizar alterações ou gravar um novo bloco áudio. Isto é importante para que o utilizador esteja ciente que os dados dos seus colegas têm alguma divergência em relação à réplica que este mantém localmente. Ainda assim, adoptando uma estratégia optimista, deve ser considerada a possibilidade de existirem escritas sobre os mesmos dados, sendo necessário recorrer ao uso de uma técnica de detecção e resolução de conflitos.

4.2 – Estratégia de colaboração

Tendo em conta a análise feita à forma como a colaboração acontecerá, como estratégia adopta-se uma arquitectura *peer-to-peer*, sendo os dados mantidos no computador de cada colaborador e as alterações sobre os dados propagadas entre eles. Os vários *peers* ligados encontram-se através de um servidor com informação acerca dos vários utilizadores, bandas e músicas, bem como os utilizadores ligados em cada momento.

Antes de detalhar acerca da estratégia de colaboração, introduzem-se alguns conceitos necessários neste contexto, tendo em conta a terminologia adoptada em [17].

- Objecto – algum tipo de dados que é partilhado.
- Site – nó (computador) do sistema distribuído que mantém réplicas dos objectos.
- *Master* site – nó do sistema distribuído que tem permissões de escrita sobre os objectos.
- Operação – uma acção feita sobre um objecto que provoca a sua alteração.

- *State transfer* – técnica de propagação de alterações, enviando a última versão alterada de um objecto.
- *Operation transfer* – técnica de propagação de alterações, enviando as operações realizadas sobre um objecto para que sejam aplicadas sobre as réplicas do objecto presentes nos restantes sites.
- Relação *happens-before* – relação de ordem entre dois acontecimentos, em que um acontece antes (*happens-before*) do outro.

No contexto deste trabalho, são considerados objectos (tendo em conta os níveis de dados definidos) a sessão de trabalho, a faixa áudio e o bloco áudio. Estes objectos são alvo de alterações, provocadas pelas operações que lhes são aplicadas.

Na arquitectura *peer-to-peer* a ser utilizada, os objectos são mantidos nos computadores dos colaboradores, estando uma réplica de cada objecto associado a um dado colaborador. Estas réplicas podem ter alguma divergência entre si, enquanto as alterações feitas por cada colaborador não são propagadas para os restantes. Cada computador de um colaborador é considerado um site, sendo que neste sistema em concreto, todos os sites são *master* sites, visto que todos os colaboradores podem alterar todos os objectos.

Para propagar as alterações feitas sobre um objecto para as restantes réplicas, será utilizada a técnica de *operation transfer*, de forma a fazer evoluir todas as réplicas do objecto para a versão mais actual. A diferença desta técnica em relação à técnica de *object transfer* é que esta envia apenas a informação acerca da operação aplicada, enquanto *object transfer* envia a réplica do objecto modificado. No contexto da edição colaborativa de música, isto significaria que cada vez que se fizesse uma operação sobre um bloco áudio, este seria reenviado. Mesmo utilizando codificação de áudio, a dimensão dos dados a transmitir não é desprezável. O envio de informação sobre as operações aplicadas minimiza o tráfego na rede e delega a evolução de uma réplica de um bloco áudio no processamento a ser realizado no site que recebe essa informação.

Sendo expectável que existam poucos acessos de escrita sobre o mesmo objecto por parte de colaboradores diferentes, opta-se por uma abordagem optimista da replicação dos dados. Isto significa que é possível que a cada momento, nem todos os colaboradores tenham os objectos na sua versão mais actual, mas podem prosseguir com o seu trabalho sem ter de esperar pelo acesso exclusivo ao objecto. Novamente tendo

em conta que os colaboradores conversam entre si, se existirem alterações de tal forma profundas que põe em causa o trabalho feito com base numa versão anterior de dado objecto, é provável que este facto seja comunicado entre colaboradores.

Utilizando uma abordagem optimista da replicação dos dados é necessário ter uma estratégia de detecção e resolução de conflitos.

4.2.1 - Detecção de conflitos

Para tratar da detecção de conflitos é utilizada uma estratégia baseada em relógios vectoriais [18], de forma a estabelecer relações de *happens-before* e de concorrência entre as operações realizadas sobre os objectos.

Designa-se de relógio vectorial a estrutura de dados que mantém para cada site um contador, incrementado cada vez que um evento acontece nesse site. Esta estrutura capta a relação de *happens-before* bem como a ocorrência de eventos de forma concorrente.

Um relógio vectorial, consiste num vector no formato {m,n,…,k } em que *m* é o valor do relógio para o primeiro site e *k* o valor para o último site.

Comparando dois eventos aos quais foram anexos relógios vectoriais, um evento α acontece primeiro que um evento β (*happens-before*) se os seus relógios forem diferentes e, para qualquer k em {site 1… site m}, relógio α [k] ≤ relógio β [k]. Caso o relógio do evento α seja diferente do relógio do evento β e, nenhum dos eventos acontecer antes do outro pela definição anterior, então são eventos concorrentes.

Entre colaboradores, as operações são propagadas em lote. A cada lote é anexo um relógio vectorial para que possa ser feita a ordenação dos lotes, detectando ainda lotes submetidos de forma concorrente. Operações concorrentes que não aconteçam sobre os mesmos objectos não são problema, pelo que além de detectar a concorrência, é necessário verificar se existe conflito no acesso de escrita ao objecto. Para verificar esses conflitos, em cada lote de operações é anexa a informação de pré-condições, que dizem respeito às versões que os objectos devem ter para que as operações possam ser aplicadas sobre eles. Caso alguma das pré-condições não se verifique, existe um conflito no acesso aos objectos e tem de ser resolvido antes que qualquer operação possa ser aplicada.

Na Figura 29 exemplifica-se a detecção de conflitos. Cada site mantém um relógio vectorial com a contagem de lotes de operações submetidos. Quando é submetido um lote de operações, é identificado através do relógio vectorial que lhe é anexo. O relógio vectorial tem o valor da contagem de lotes submetidos até ao momento, incrementando de 1 o valor de lotes submetidos no próprio site. Cada faixa ou bloco áudio mantém o seu número de versão, utilizando para o efeito um relógio vectorial. Como se vê na figura, este relógio é iniciado com o valor {0,0,0}, sendo alterado a cada lote que

contém operações sobre o objecto. Verifica-se no exemplo apresentado que o "Site 1" e o "Site 2" submetem lotes de operações aplicadas ao mesmo bloco áudio "b1". Neste caso existe conflito.

A detectar o conflito, o "Site 1" começa por observar que o relógio anexo ao lote {1,1,0} é diferente do relógio do site, que tem o valor {2,0,0} e não existe relação de *happens-before* entre eles (visto que os sites receberam os lotes após aplicarem os seus próprios lotes). Sendo concorrente, verifica-se se o lote concorrente contém alguma operação sobre algum objecto já modificado. No caso, o lote {1,1,0} contém uma operação a aplicar sobre o bloco "b1" na versão {0,0,0}. No "Site 1" o bloco "b1" já se encontra na versão {1,0,0} e não {0,0,0}, sendo assim detectado conflito. O mesmo processo é feito nos restantes sites.

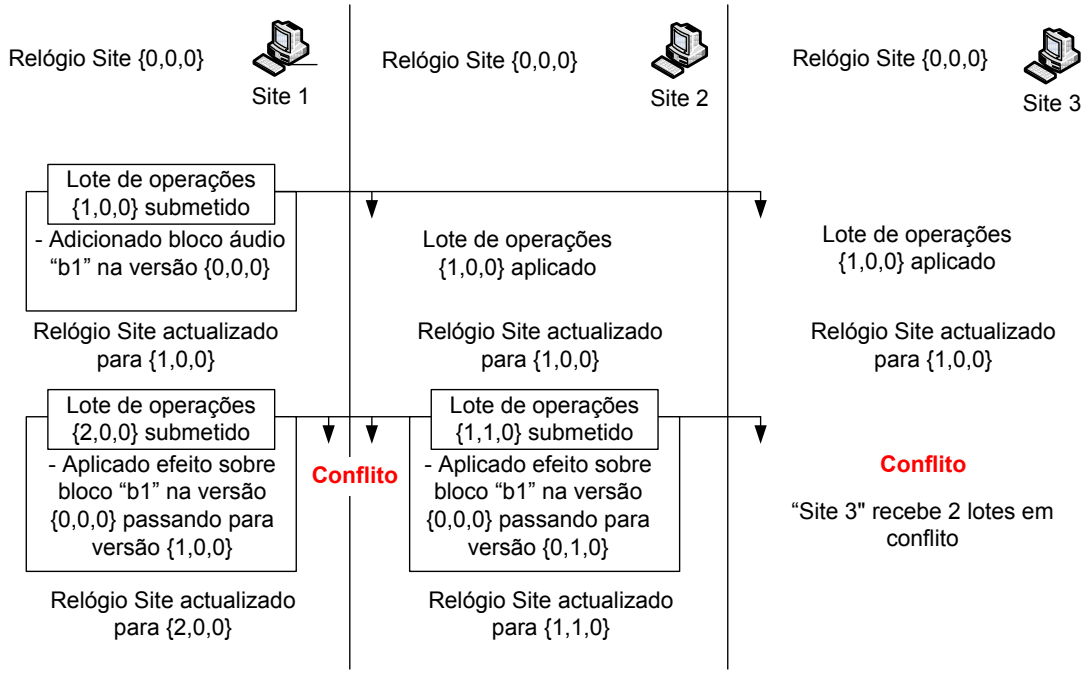

Figura 29 - Detecção de conflitos – acesso concorrente a um bloco áudio

Além da ocorrência de conflitos entre lotes de operações já submetidos, podem ocorrer conflitos quando um colaborador está a aplicar operações no seu computador. Durante o trabalho do colaborador, antes que este termine a aplicação das operações que pretende e as submeta (o lote actual), chega um lote de operações que contém uma ou mais operações sobre um objecto (ou mais) que o colaborador está a alterar. Existe conflito também neste caso.

4.2.2 – Resolução de conflitos

A resolução de conflitos é feita por arbitragem do utilizador. Aquando da detecção de conflito, é perguntado aos colaboradores que trabalham nos computadores que detectaram esse conflito, o que pretendem fazer.

Para resolver o conflito, os colaboradores têm duas hipóteses: aceitar ou rejeitar o lote de alterações submetido por outro colaborador. Se o colaborador aceitar, os lotes de operações em conflito com o novo são desfeitos (*undo*), são aplicadas as operações do novo lote e é enviada uma notificação de aceitação do lote aos restantes colaboradores. Se a opção tomada for rejeitar, as operações já aplicadas são mantidas e é enviada uma notificação para os restantes colaboradores dizendo que o novo lote foi rejeitado e as suas operações devem ser desfeitas nos computadores em que já foram aplicadas.

De notar que os conflitos são detectados em mais que um computador, pelo que as opções tomadas pelos colaboradores devem estar de acordo (por exemplo, se o colaborador 1 e o colaborador 2 submetem lotes de operações em conflito, se o colaborador 1 aceita, então o 2 deve rejeitar e vice-versa). Caso as opções tomadas não estejam de acordo, ou seja, todos os lotes em conflito são aceites ou rejeitados, todos serão desfeitos. No primeiro caso, os lotes serão desfeitos pois tal resultaria em réplicas inconsistentes entre sites. No segundo caso, todas são rejeitadas pelos colaboradores, tendo de ser desfeitas.

O comportamento transaccional do sistema, com a possibilidade de desfazer lotes de operações, reforça a necessidade de utilizar como técnica de propagação de alterações *operation transfer*. Desta forma, os lotes de operações são enviados e registados, podendo estes dados registados ser utilizados posteriormente para desfazer alterações sem necessitar de outra informação de outros colaboradores e ainda para ir buscar os lotes a serem propagados. Para obter este comportamento utilizando a técnica de propagação *state transfer*, é necessário guardar todas as versões de cada objecto para que seja possível voltar a uma delas. Uma implementação deste tipo aumentaria (em relação a *operation transfer*) o espaço de armazenamento necessário, de forma a guardar a informação dos objectos em cada uma das suas versões (sobretudo no caso dos ficheiros áudio). A largura de banda necessária para propagar as mesmas alterações seria, pelo mesmo motivo, também superior.

Na Figura 30 apresenta-se um exemplo de detecção de conflito (explicada previamente) e respectiva resolução. Após detecção de conflito, a aplicação presente em cada site vai consultar o utilizador de forma a determinar a resolução do conflito. No exemplo apresentado, os utilizadores chegam a consenso que dos dois lotes concorrentes, o lote {1,1,0} é o que se deve manter. Assim, no "Site 1" é aceite o lote {1,1,0} e por consequência, desfeito o lote {2,0,0}. No "Site 2", onde foi submetido o lote {1,1,0}, este já está aplicado e basta que o seu concorrente não o seja. No "Site 3", visto que o primeiro lote recebido foi o {2,0,0}, foi imediatamente aplicado pois não existia qualquer conflito. Ao chegar o lote {1,1,0}, sendo este o aceite, o lote {2,0,0} é desfeito.

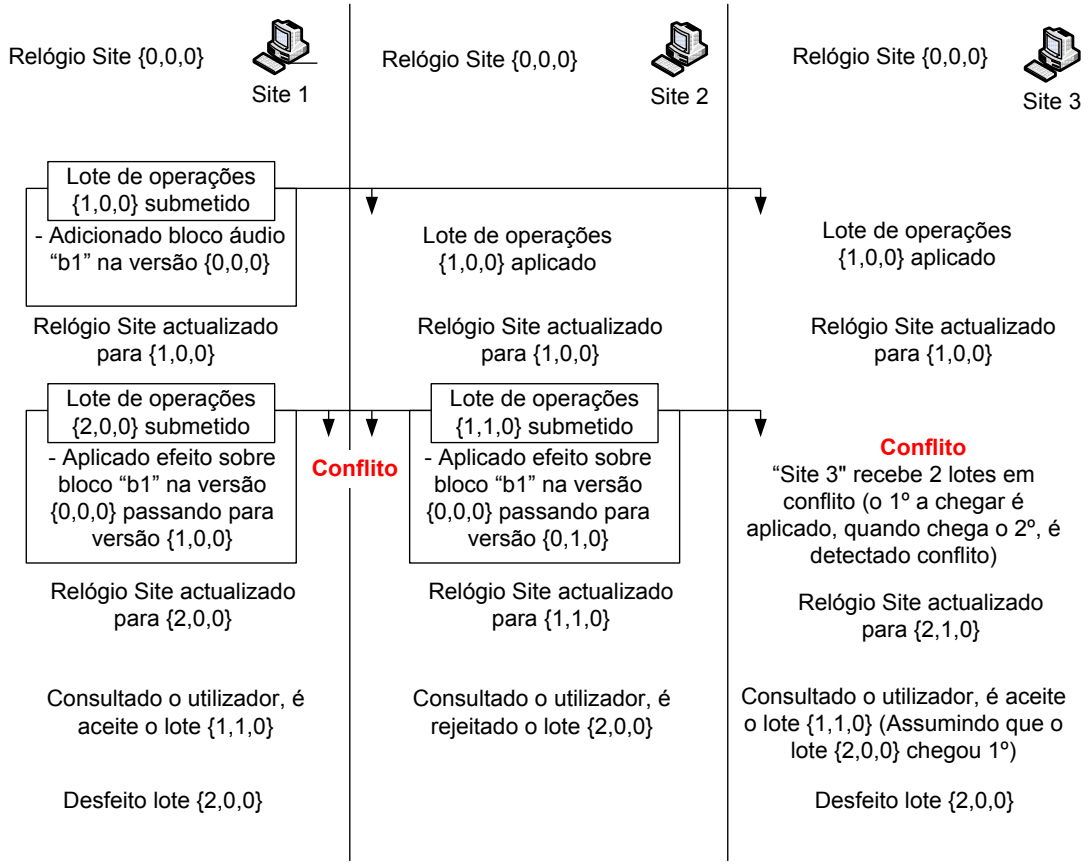

Figura 30 - Resolução de conflitos por arbitragem

Sempre que chega um lote de operações, é enviada uma notificação de aceitação ou rejeição para os restantes sites. Se não existir conflito, o lote é aceite. Em caso de conflito, é consultado o utilizador. Estas notificações não estão incluídas na Figura 30 nem a sua influência no relógio do site (cada aceitação/rejeição de lote resulta também no incremento do valor do relógio).

4.2.3 – Modelo de utilização de lotes de operações

Com a introdução do conceito de lote de operações, sendo a unidade mínima que se pode enviar para que sejam feitas alterações sobre os objectos, torna-se importante ter em conta a relação entre operações.

A relação entre operações não é conhecida pela aplicação: trata-se de uma relação atribuída mentalmente pelos utilizadores quando aplicam uma série de alterações sobre um ou mais objectos de forma a obter determinado resultado. Visto que em caso de conflito entre lotes de operações, alguns serão desfeitos, pode acontecer que nos lotes estejam presentes operações não relacionadas entre si, e que são desfeitas quando o conflito se deve a outras operações do mesmo lote.

Para evitar que operações sejam desfeitas desnecessariamente, os lotes de operações devem ser utilizados considerando que cada lote é uma transacção, é feito tudo ou nada. Assim, os colaboradores devem iniciar um lote de operações, aplicar as operações que desejam, tendo em conta algum tipo de relação que entendam e, quando terminam a aplicação de operações relacionadas, submetem o lote. Para o colaborador, um lote será como um compromisso que este assume, em como determinado grupo de operações só faz sentido em conjunto.

4.2.4 – Protocolo de comunicação entre sites

A comunicação entre sites é feita através das seguintes mensagens:

- Envio de relógio vectorial de um site, para informar da submissão de lotes de operações e para obter lotes em falta;
- Notificação de aceitação ou rejeição de lote de operações;
- Pedido de bloco áudio - um bloco áudio gravado pode corresponder a um ficheiro de dimensão significativa. Por este motivo, a sua transferência pode ser feita à parte das restantes operações, tirando partido de mecanismos de transferência de ficheiros que não são necessários para a propagação de operações. Ao ser feita uma operação de adição a uma sessão de um bloco áudio, o registo dessa operação conterá o identificador do bloco adicionado, utilizando-se esse bloco para pedir a outro colaborador o ficheiro áudio correspondente.

4.3 – Componente Servidor

A componente Servidor da aplicação distribuída mantém informação dos utilizadores ligados, como o endereço IP e porto em que a aplicação cliente (em execução no computador de cada utilizador ligado) está à escuta de pedidos. Assim, quando um utilizador liga a aplicação cliente, esta obtém a partir do servidor os endereços dos seus colaboradores para que possam comunicar.

No servidor é também mantida informação de registo dos utilizadores da aplicação, bandas a que pertencem (criadas na aplicação) e sessões de trabalho (músicas) associadas às bandas.

O servidor expõe as seguintes operações:

- Registar, ligar ou desligar o utilizador e alterar a palavra-chave de um utilizador.
- Criar uma banda, convidar um utilizador para uma banda.
- Criar uma sessão de trabalho associada a uma banda, fazer iniciar ou terminar o trabalho nessa sessão.
- Verificar conectividade de um site.

Detalhando algumas destas operações, o acto de um utilizador se ligar, cria na base de dados do servidor uma entrada de um utilizador ligado. Nesta entrada é registado o endereço base do site, para que possa ser fornecido aos restantes. Não é nesta altura que o endereço é fornecido aos restantes sites, mas apenas quando se encontram a trabalhar sobre a mesma sessão. Além do endereço base, é guardado na base de dados um *token* (recorrendo a um Globally Unique Identifier) que identifica o utilizador como ligado, sendo enviado para o servidor como parâmetro a cada operação pedida. Se o utilizador se ligar noutro site, o *token* é sobreposto, não podendo o utilizador estar ligado em mais do que um site (ou instância da aplicação cliente).

Os sites podem trocar mensagens entre si após ser feito iniciar o trabalho numa sessão. Ao ligar-se à sessão, o site recebe os endereços dos restantes já ligados à sessão em causa, recebendo estes uma notificação de novo site ligado.

A notificação dos sites aquando do início e término de trabalho de outro site numa sessão de trabalho é feita de forma assíncrona, recorrendo a outro serviço. Desta forma

impede-se que a aplicação cliente que realizou o pedido de início ou término fique a aguardar o retorno da operação por parte do servidor, enquanto este envia a devida notificação aos restantes.

Quando um site não consegue comunicar com outro (tendo já obtido o respectivo endereço) faz um pedido ao servidor para que este verifique se também não consegue. O servidor após tentar comunicar com o site, caso não consiga, dá o site como desligado e remove a sua entrada na base de dados (na tabela de utilizadores ligados) e notifica os restantes sites. Caso a comunicação seja realizada, o servidor retorna o endereço do site, pois a falha de comunicação pode ter resultado da mudança do endereço (endereços IP atribuídos dinamicamente).

4.3.1 - Aspectos de implementação

A palavra-chave do colaborador (utilizador) é alvo de uma função de *hash* uma vez do lado da aplicação cliente e novamente no lado do servidor, neste caso juntamente com um *salt* guardado na informação sobre o utilizador. O *salt*, gerado aleatoriamente, adiciona complexidade à descoberta da palavra-chave de um utilizador, quando é feito um ataque de dicionário conhecendo o *hash* da palavra-chave guardado na base de dados [19].

Para possibilitar o envio de informação sobre erros às aplicações que invocam o serviço, são estabelecidos no seu contrato os tipos de falhas que podem ocorrer. Desta forma é possível, do lado da aplicação cliente, identificar o que correu mal e agir em consonância. Por exemplo, se um utilizador tenta fazer o registo com um email inválido é enviado um erro do tipo FaultException<**InvalidEmailFaultDetail**>, como se pode ver na Listagem 1.

```
if (!UtilityMethods.ValidateEmail(email))
    throw new FaultException<InvalidEmailFaultDetail>(
                new InvalidEmailFaultDetail("The email entered is not valid.")
    );
```

Listagem 1 - Erro no servidor, email inválido

A implementação do servidor recorre a Windows Communication Foundation (WCF), sendo um serviço acedido pela aplicação cliente via protocolo HTTP. O serviço de notificação está também implementado em WCF, mas é acedido pelo primeiro serviço utilizando Microsoft Message Queuing (MSMQ). O acesso à base de dados é feito com

ADO.NET Entity Framework, que permite um desenvolvimento fácil e rápido do acesso aos dados. A base de dados foi desenvolvida com SQL Server 2008.

4.4 - Componente Cliente

Em cada site está instalada uma aplicação cliente, na qual é feita a gravação e edição da música. Sendo utilizada numa arquitectura *peer-to-peer* híbrida, esta aplicação tratará do envio e recepção de pedidos entre sites. De forma a estar à escuta de pedidos, a aplicação hospeda serviços acedidos via protocolo TCP-IP.

4.4.1 – Serviços hospedados pela aplicação

Para a aplicação receber pedidos, hospeda 3 serviços. A separação em vários serviços deve-se à divisão de funcionalidades distintas. Para cada um destes serviços, foi definido um contrato com as funcionalidades que apresenta.

- **Serviço cliente** – apresenta funcionalidades que são invocadas pelo servidor, como notificação de *login* e *logout* de outros utilizadores. Permite também que o servidor verifique a conectividade à aplicação cliente.
- **Serviço de transferência de operações** – apresenta funcionalidades para a troca de lotes de operações realizadas e notificações de aceitação ou rejeição de lotes submetidos.
- **Serviço de transferência de ficheiros** – permite realizar pedidos de blocos áudio.

A definição destes contratos é realizada em WCF. Nesta tecnologia, a definição de um contrato de serviço é feita através de uma interface em .NET. Para cada serviço foi definida uma interface, com um método por cada funcionalidade a disponibilizar pelo serviço. Posteriormente, o contrato definido em .NET é transformado no formato WSDL (*Web Services Description Language*).

Os 3 contratos referidos atrás, são definidos pelas interfaces .NET **IClientSiteService**, **ISiteOperationTransferService** e **ISiteFileTransferService** respectivamente.

Relativamente à interface **ISiteFileTransferService**, de referir que na definição do método de obtenção de um bloco áudio, o retorno é feito através de *streaming*, permitindo que a transferência seja interrompida a meio e retomada a partir do momento de interrupção. Além desta possibilidade, a utilização de *streaming* em detrimento de

uma transferência *buffered* (retorno de um *byte*[]), evita que todo o bloco áudio tenha de ser todo colocado e mantido em memória enquanto é feita a transferência.

A implementação destes serviços servirá apenas como ponto de ligação entre sites, bem como sites e servidor. A lógica de processamento de pedidos consiste no encaminhamento destes (através de eventos disponibilizados pelos serviços) para uma entidade responsável por gerir a colaboração em cada site.

4.4.1.1 – Objectos de transporte (*Data Transfer Objects*)

Na biblioteca **OperationObjects.dll** foram definidos os tipos de objectos de transporte a usar no serviço com o contrato **ISiteOperationTransferService**. Estes tipos consistem na definição dos dados que têm de ser comunicados entre instâncias da aplicação, surgindo aqui o lote de operações, notificação de aceitação/rejeição de um lote e as operações propriamente ditas.

Além de utilizados no transporte da informação, estes objectos, nomeadamente as operações, também são utilizados na aplicação. Esta opção foi tomada em detrimento da definição de novos tipos internos à aplicação, que não fossem conhecidos pelos serviços, visto que as definições seriam semelhantes, pois estes objectos contêm apenas a informação necessária para aplicação das operações. Mesmo admitindo que a definição destes tipos possa sofrer alterações, estas alterações teriam de se reflectir também nos tipos definidos, pois essas alterações certamente trariam nova informação necessária à aplicação das operações.

Na Figura 31 pode-se observar a definição do lote de operações, notificação de aceitação/rejeição e os tipos que por eles são utilizados.

O lote de operações (**OperationGroup**) e a notificação (**OperationGroupNotification**) implementam a interface (**ITransferObject**) que indica o site que submeteu o objecto e o identificador deste objecto. O lote de operações mantém as pré e pós condições necessárias à aplicação das suas operações. Estas condições são mantidas por instâncias do tipo **OperationGroupConditions** e, para cada faixa ou bloco áudio usados/modificados pelas operações do lote, indicam a versão em que se devem encontrar (pré condições) ou vão ficar (pós condições) essas faixas e blocos. As versões são representadas por instâncias do tipo **ObjectVersion**, que mantém um relógio vectorial e o número de tentativas de chegada à versão em causa. Este número de

tentativas é usado nos casos em que uma versão é recusada, a versão anterior reposta e ao fazer de novo avançar a versão, o relógio vectorial apresenta os mesmos valores. Finalmente, o lote de operações mantém uma colecção de operações, instâncias de tipos que implementam a interface **IOperation**.

A notificação de aceitação/rejeição de lote define, para além dos membros de **ITransferObject**, o identificador do lote que deve ser aceite/rejeitado e um valor booleano indicado se o lote é ou não para aceitar.

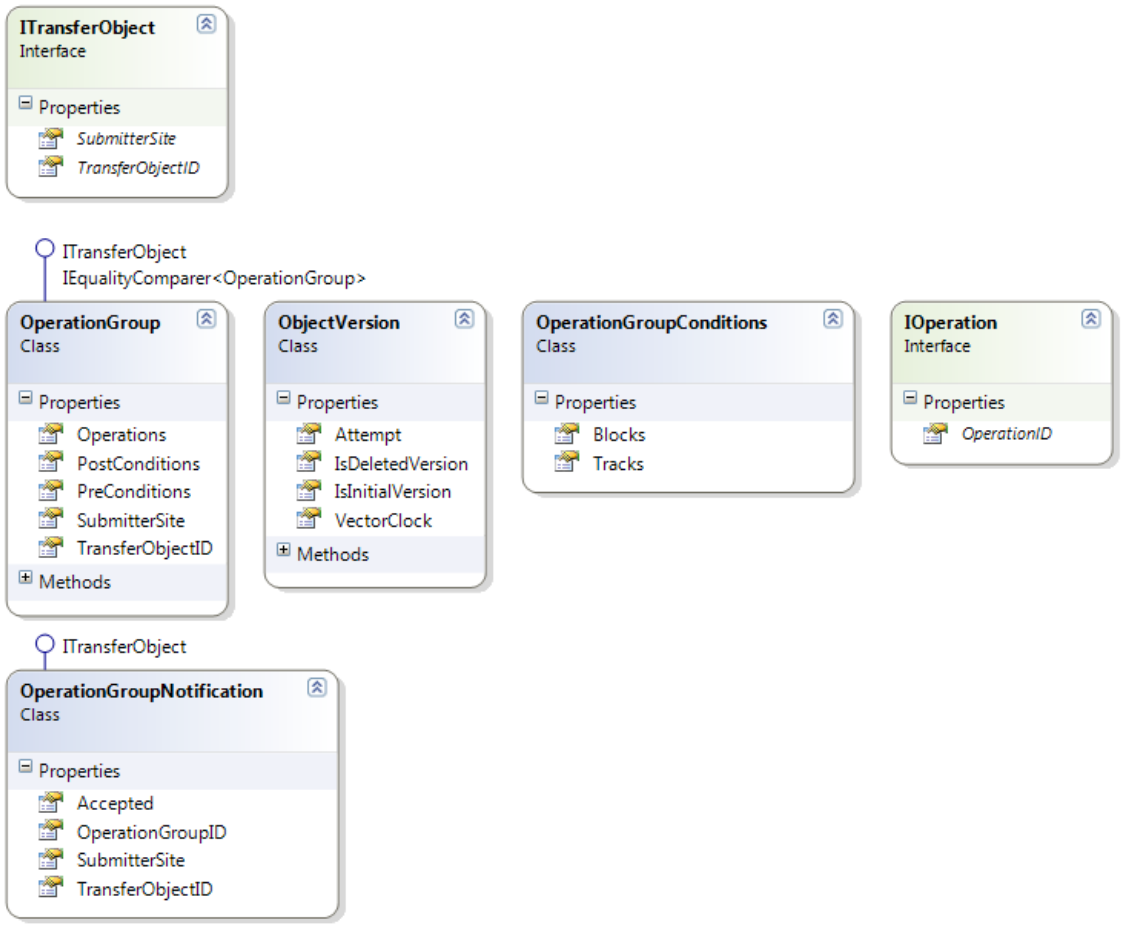

Figura 31 - Lote de operações, notificação e tipos relacionados

Na Figura 32 mostram-se algumas das operações definidas; neste caso, as operações sobre uma sessão. Existem além destas, as operações sobre faixas (**ITrackOperation**) e operações sobre blocos áudio (**IBlockOperation**). Foram definidas duas interfaces para diferenciar 2 tipos distintos de operações sobre uma sessão, as que envolvem faixas e as que envolvem blocos áudio (**ITrackInSessionOperation** e **IBlockInSessionOperation**). Estas operações consistem em adicionar

(**AddTrackToSession** e **AddBlockToSession**) e remover (**RemoveTrackFromSession** e **RemoveBlockFromSession**) faixas e blocos áudio presentes na sessão.

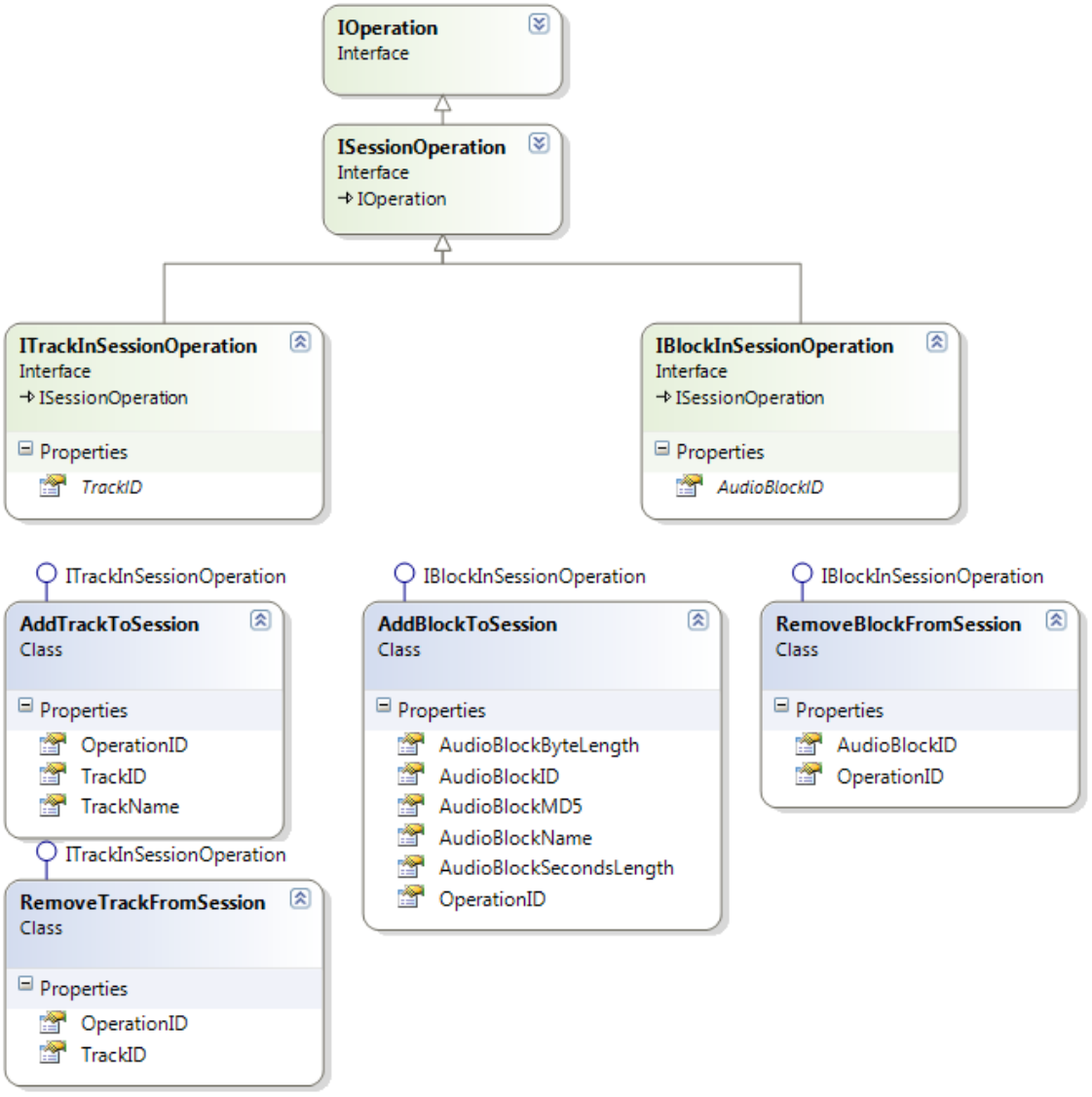

Figura 32 - Operações sobre a sessão

As operações sobre faixas (implementações de **ITrackOperation**) consistem em adição e remoção de blocos áudio a uma faixa e colocar uma faixa com ou sem som (*mute*). As operações sobre blocos (implementações de **IBlockOperation**) consistem na aplicação de edições como amplificação, multiplicação por ondas ou aplicação de ecos sobre blocos áudio.

4.4.2 – Componentes que implementam a lógica da aplicação

Na aplicação, os 2 componentes essenciais no que diz respeito à lógica são o gestor local e o gestor de colaboração (cujos contratos são definidos pelas interfaces **ILocalManager** e **ICollaborationManager** respectivamente).

O gestor local é responsável pela manutenção dos objectos de trabalho localmente (sessão, faixas e blocos áudio), pela aplicação de operações sobre estes objectos (adição, edição e remoção de objectos, assim como cancelamento de operações previamente aplicadas). Tanto as operações submetidas localmente, na instância da aplicação cliente a que o gestor pertence, como as operações recebidas via internet, submetidas em outros sites serão aplicadas pelo gestor sobre os objectos mantidos localmente, pelo que o gestor tem preocupações com a concorrência no acesso a estes objectos.

O gestor de colaboração é responsável por hospedar os serviços para receber mensagens do servidor e restantes sites, tratar as mensagens recebidas (registando-se nos eventos disponibilizados pelos serviços) e enviar mensagens informando de acontecimentos ocorridos localmente. O gestor de colaboração mantém ainda o registo de acontecimentos (lotes de operações e notificações de aceitação/rejeição de lotes submetidos) recorrendo a uma implementação da classe **ILogManager**.

Na Figura 33 observa-se a organização das principais classes da componente cliente, com ênfase na relação entre os gestores local e de colaboração e a classe que representa a aplicação, a classe que trata as definições da aplicação, a classe que trata do registo de ocorrências e as classes que representam os serviços.

A aplicação (representada pela classe **App**) mantém uma referência para o gestor local (**LocalManager**). O gestor local, armazena em disco as definições da aplicação (através da classe **ApplicationSettings**) bem como informação das sessões presentes localmente (estado da sessão e seus objectos). O gestor de colaboração, referenciado pelo gestor local, mantém em disco o registo de ocorrências através do **LogManager**. O gestor de colaboração comunica ainda com outras instâncias da aplicação cliente via internet, enviando mensagens e tratando as mensagens recebidas pelos serviços hospedados.

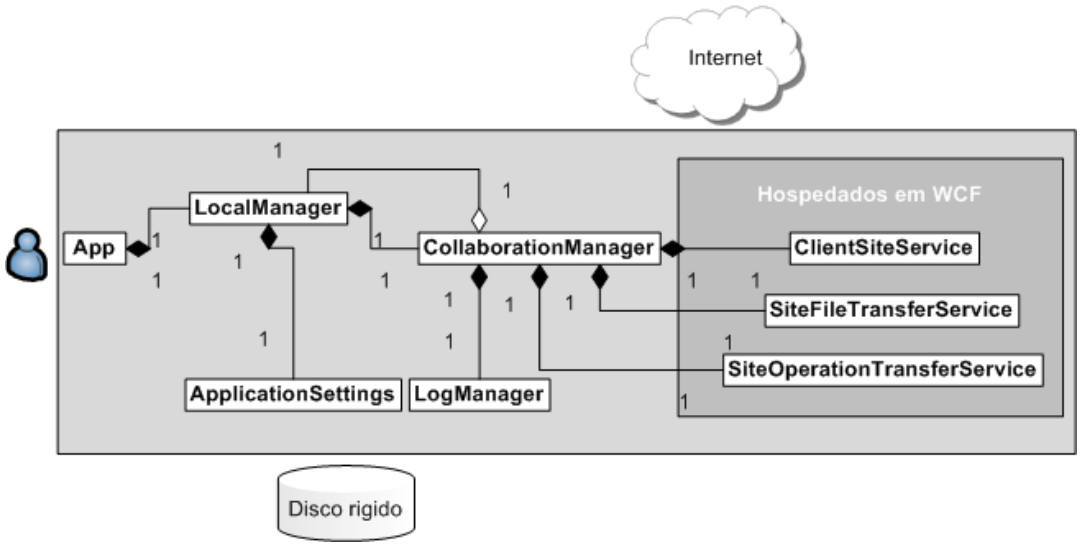

Figura 33 - Organização da componente cliente

4.4.2.1 – Implementação do gestor local (LocalManager)

O gestor local mantém os objectos bem como sobre eles aplica (ou cancela) operações. É mantida uma referência para a sessão corrente, instância do tipo **LocalSession,** implementação da interface **ILocalSession**, que se pode ver na Figura 34, contendo informação como o nome e identificador da sessão, directoria onde os seus dados são guardados no disco, faixas e blocos áudio. Também na Figura 34 se podem ver as interfaces que representam as faixas (**ILocalTrack**) e os blocos áudio (**ILocalAudioBlock**), implementadas pelas classes **LocalTrack** e **LocalAudioBlock** respectivamente. Cada faixa mantém uma colecção de blocos áudio, aos quais associa alguma informação: o identificador do bloco na faixa, o tempo em que o bloco se coloca na faixa e se este bloco está ou não com som (*mute*). O bloco áudio mantém o áudio, informação para obter a representação gráfica do áudio, o nome do ficheiro em disco associado ao bloco. O bloco mantém ainda a indicação se existe localmente (pode ter sido introduzido por outro utilizador e ainda não ter sido transferido) e o *hash* MD5 [20] para verificação aquando da transferência entre sites.

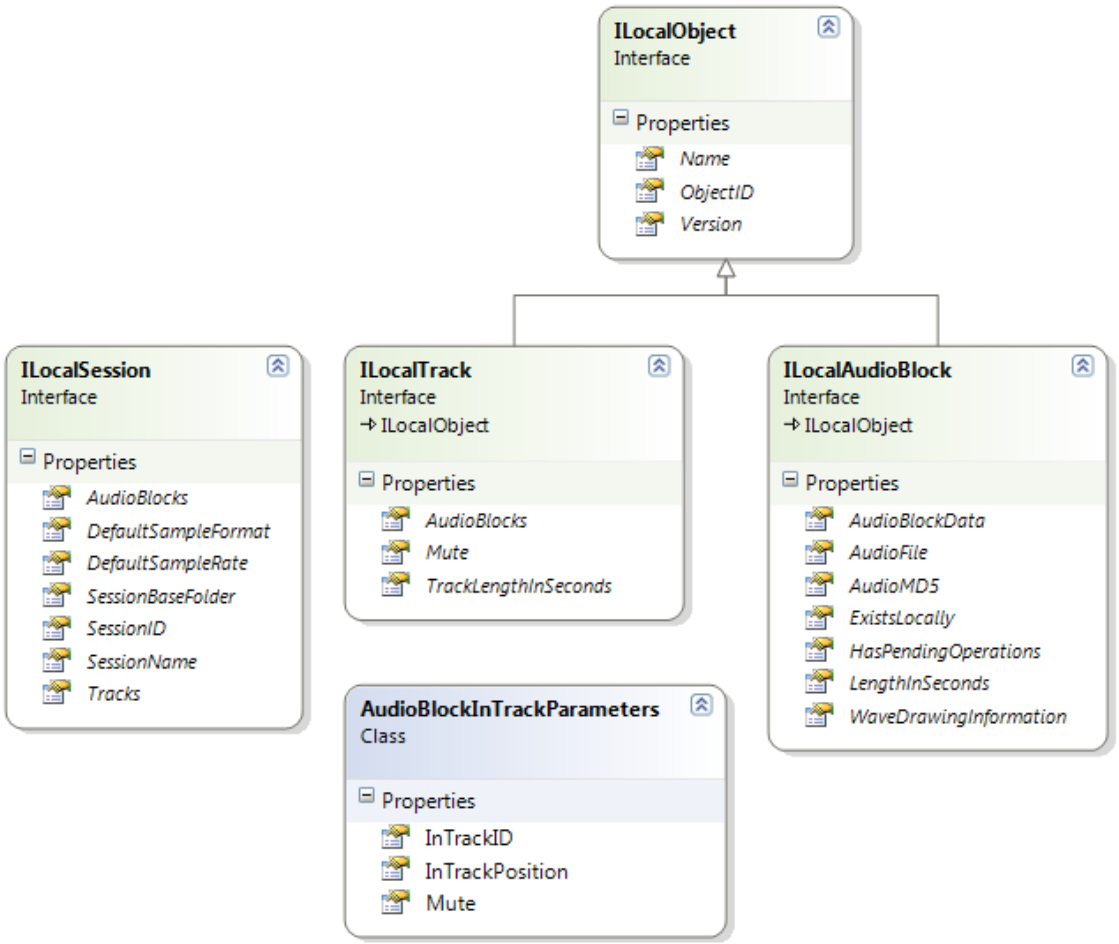

Figura 34 - Representação dos objectos locais

Além dos objectos de trabalho, o **LocalManager** mantém os caminhos relativos para as directorias onde vai armazenar os dados. Na Figura 35 observa-se a organização em disco dos dados. Como se pode ver nesta figura, será guardado um ficheiro com as definições da aplicação. Além deste ficheiro, existirá uma directoria para cada sessão mantida localmente, onde serão guardadas as definições (e estado) da sessão, o registo de ocorrências e os blocos áudio usados na sessão. Os blocos áudio estão organizados em várias directorias que consistem em:

- Ficheiros originais – são mantidos numa directoria os ficheiros correspondentes aos blocos áudio sem edições aplicadas. Desta forma, recorrendo ao registo de ocorrências, é possível chegar a uma qualquer versão do bloco áudio, aplicando as operações registadas.
- Ficheiros submetidos – são mantidos os ficheiros correspondentes aos blocos áudio cujas operações foram submetidas. Isto permite que ao iniciar a aplicação,

se carreguem rapidamente os blocos áudio, correspondendo o áudio no ficheiro à última versão do bloco áudio submetida localmente.

- Blocos áudio a ser editados – enquanto são aplicadas operações pelo gestor local, enquanto o estado não é o final (relativamente ao conjunto de operações que estão ou vão ser aplicadas) os blocos áudio editados ficam, nesta directoria. Quando o estado é final, os blocos são movidos para a directoria de ficheiros submetidos.
- Ficheiros removidos – Quando um bloco áudio é removido, o ficheiro original é movido para a directoria de ficheiros removidos. Isto é feito pois a manutenção de operações aplicadas está dependente de, existindo conflitos no acesso aos objectos, as novas operações recebidas serem aceites ou rejeitadas. Imaginando um caso em que é aplicada uma operação de remoção de bloco áudio, ao ser recebida noutro computador, o seu utilizador recusa a aplicação desta operação que está em conflito com outro já submetida localmente. No computador em que a operação já foi submetida, recorrendo ao registo e ao ficheiro guardado na directoria de ficheiros removidos, é possível repor o bloco áudio no estado anterior à operação.

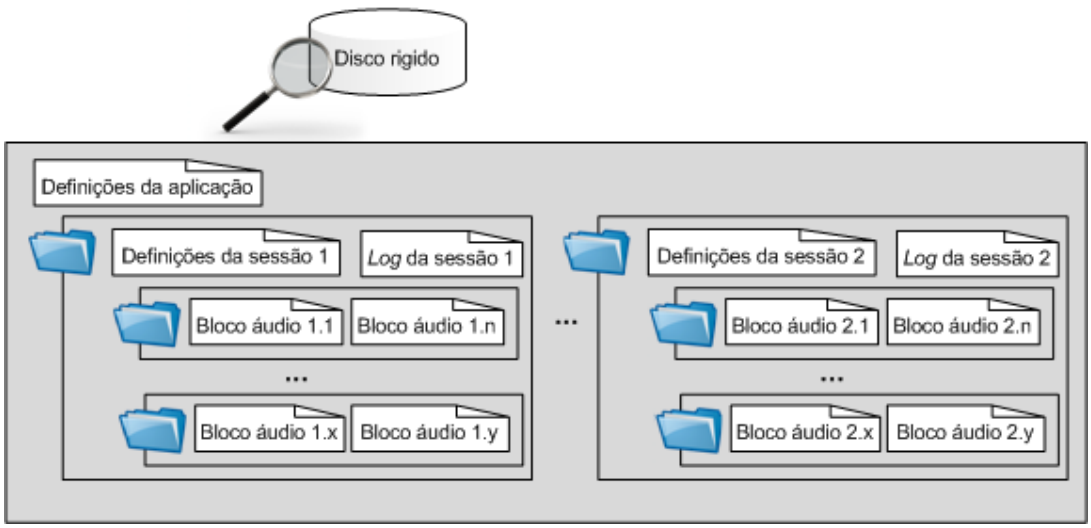

Figura 35 - Organização dos dados da aplicação em disco

O gestor local apresenta na sua interface pública métodos a serem invocados pela interface gráfica, para iniciar, cancelar ou submeter um lote de operações, adicionar uma nova operação ou cancelar a última aplicada. A Listagem 2 apresenta a assinatura destes métodos.

```
void AddOperation(IOperation operation, object extraParameters);

void CancelLastOperation();

void StartNewOperationGroup();

void SubmitOperationGroup();

void CancelOperationGroup();
```

Listagem 2 - Assinatura dos métodos de aplicação de operações

A assinatura do método de adição de operações (AddOperation). Este método recebe como parâmetros a operação a aplicar (objecto do tipo **IOperation**) e parâmetros extra. Os parâmetros extra aparecem com o tipo Object, pois são necessários em poucas operações (na implementação actual apenas para uma operação) e com tipos expectavelmente distintos. Na implementação actual, a única operação que tem parâmetros extra, além dos que seguem no objecto do tipo **IOperation** é a operação de adição de bloco áudio, que recebe o áudio gravado (ou importado).

Para realizar a aplicação, submissão e cancelamento de operações foi criada a classe abstracta **LocalOperationApplier** (na Figura 36). Por cada tipo de operação existe uma concretização desta classe. Sempre que é adicionada uma operação (através do método AddOperation de **LocalManager**) é instanciada uma concretização de **LocalOperationApplier**, sendo iniciadas as referências: para a instância de **LocalManager** (propriedade LocalManagerInstance), a operação **IOperation** adicionada e os parâmetros extra. Os parâmetros extra aqui usados são uma concretização da classe abstracta **OperationExtraParameters**, onde serão guardadas informções, necessárias de início (como o caso do áudio a adicionar na operação **AddBlockToSession**) e dados que são obtidos na chamada ao método Apply de **LocalOperationApplier** e que são necessários para submeter (método Submit) ou cancelar (método Undo) a operação.

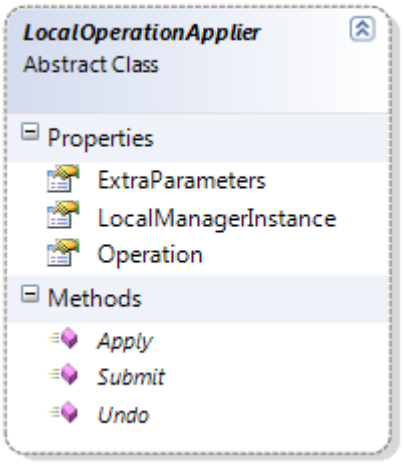

Figura 36 - Classe abstracta LocalOperationApplier

À medida que são adicionadas operações, as instâncias do tipo **LocalOperationApplier** são guardadas numa colecção e é invocado o método Apply. Quando se pretende submeter ou cancelar as operações aplicadas, são invocados, respectivamente, os métodos SubmitOperationGroup e CancelOperationGroup. O primeiro itera sobre a colecção de objectos do tipo **LocalOperationApplier** e invoca o método Submit, enquanto o segundo realiza a mesma iteração invocando o método Undo. Além da iteração, o método SubmitOperationGroup recorre à instância de **CollaborationManager** referenciada pela instância de **LocalManager** para registar as operações e actualizar as versões dos objectos modificados.

Ao aplicar (bem como a submeter e cancelar) operações localmente, existe a possibilidade de haver concorrência no acesso aos objectos, sabendo que operações submetidas por outro site são aplicadas automaticamente quando recebidas. Por esta razão, ao aplicar uma operação, é necessário verificar se existe conflito com outra (ou outras) que esteja a ser processada em simultâneo. Para verificar estes conflitos, foram criadas 6 colecções:

- LocallyCurrentlyAlteredTracks e LocallyCurrentlyAlteredAudioBlocks - informação das faixas e blocos áudio modificados ou usados por operações submetidas localmente.
- LoggedCurrentlyAlteredTracks e LoggedCurrentlyAlteredAudioBlocks - informação das faixas e blocos áudio modificados ou usados por operações submetidas noutro computador e actualmente a serem submetidas localmente.

- UndoLoggedCurrentlyAlteredTracks e UndoLoggedCurrentlyAlteredAudioBlocks - informação das faixas e blocos áudio modificados ou usados por operações submetidas localmente, mas que foram rejeitadas e estão em processo de cancelamento (*undo*).

Nestas colecções estão presentes objectos dos tipos **AlteredTrack** e **AlteredAudioBlock**, que representam uma modificação ou uso de um objecto (uma faixa ou um bloco áudio respectivamente). Na Figura 37 podem ver-se as propriedades destes 2 tipos. A destacar a propriedade AlterationType, que indica o tipo de operação que ocorreu sobre a faixa ou bloco. AddToSession e RemoveFromSession indicam que o objecto foi adicionado ou removido à sessão, respectivamente. Modify indica que o objecto foi alvo de uma edição e Used indica que um bloco áudio (não é aplicável a faixas) foi utilizado numa faixa.

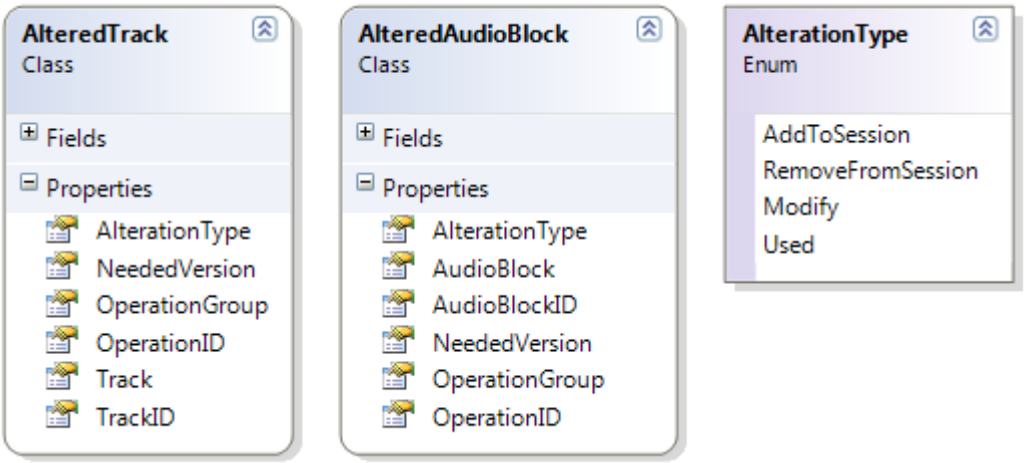

Figura 37 - Informação sobre objectos modificados/utilizados

Quando se pretende aplicar localmente operações, verificam-se as colecções "Logged" e "UndoLogged", de forma a saber se os objectos estão a ser acedidos concorrentemente. Se existirem acessos concorrentes, em que pelo menos um deles é de escrita, não é possível aplicar a operação local. Sendo detectado conflito, não pode ser aplicada a operação localmente até que as operações recebidas tenham a sua aplicação concluída. Assim, é lançada a excepção **ObjectAccessConflictException**. Se não existir conflito, é adicionada à colecção "Locally" adequada a informação sobre o objecto a modificar/utilizar. Todo este processo de verificação de conflitos é feito no contexto de um *lock*, de forma a garantir a exclusão mútua visto que existem duas *threads* que acedem a estes dados: uma para operações locais e outra para operações recebidas. A

salientar que, no âmbito da aplicação de operações locais ou recebidas, o acesso às colecções de objectos ou às colecções com informação dos objectos utilizados é sempre feito em exclusão mútua recorrendo ao mesmo monitor, de forma a garantir a consistência e visibilidade dos dados.

O uso de colecções para verificar conflitos em vez de adquirir e manter um *lock* ao longo de toda a aplicação de operações permite minimizar o tempo em que as threads aguardam a posse do *lock*, podendo realizar o seu trabalho em paralelo se não existirem conflitos.

Como exemplo da implementação de uma operação local, na Figura 38, Figura 39 e Figura 40 apresentam-se os fluxogramas dos métodos Apply, Submit e Undo respectivamente, para a operação de remoção de um bloco áudio da sessão. Estes métodos são implementados na classe **LocalRemoveAudioBlockFromSession**, concretização da classe abstracta **LocalOperationApplier**.

O método Apply (Figura 38) verifica se é possível remover o bloco (se não existe acesso concorrente, se o bloco está disponível e se não é usado em nenhuma faixa) e de seguida remove-o, adicionando à colecção LocallyCurrentlyAlteredAudioBlocks a informação que este bloco foi removido.

O método Submit (Figura 39) completa o trabalho iniciado no método Apply. Se o ficheiro correspondente ao bloco áudio removido existir localmente, o ficheiro é transferido para uma pasta de blocos "reciclados", onde ficam para nova adição, caso o utilizador o pretenda aproveitar. Se o bloco áudio não estiver na versão inicial, significa que existe uma versão posterior submetida, sendo o ficheiro correspondente eliminado da directoria de blocos áudio submetidos.

O método Undo (Figura 40) desfaz o que foi feito no método Apply. O bloco áudio removido volta a ser colocado na colecção de blocos áudio da sessão e é removida a entrada com informação de bloco modificado.

Na classe **LocalManager**, os métodos AddOperation, SubmitOperationGroup, CancelOperationGroup e CancelLastOperation são executados na posse do *lock* sobre o objecto preProcessingLock. Isto é feito para evitar que sejam executados em concorrência com o pré-processamento de operações recebidas (a discutir posteriormente). Neste processo são analisados possíveis conflitos com as operações a

aplicar localmente, bem como as que já o foram e se encontram no registo, pelo que não podem ocorrer alterações aos dados que estão a ser analisados.

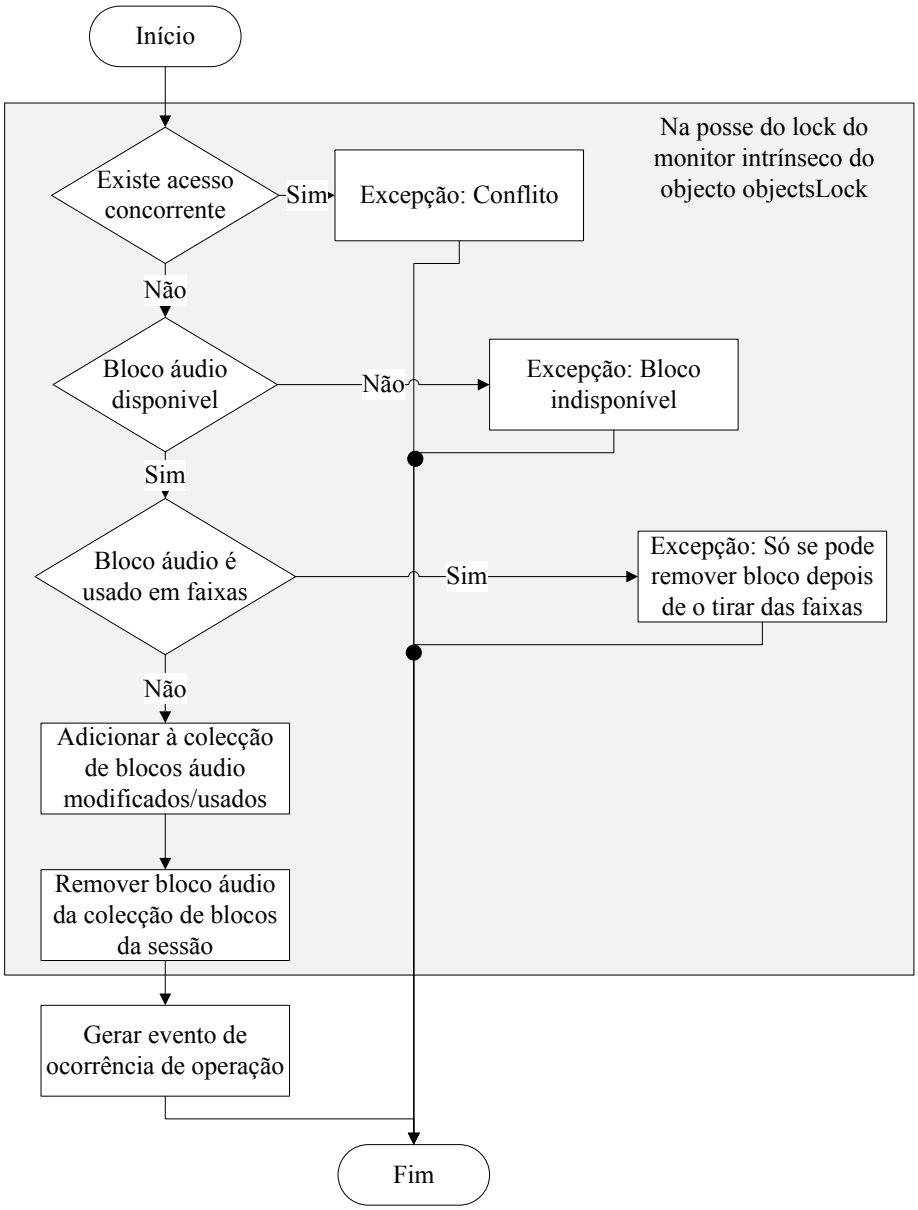

Figura 38 - Fluxograma do método Apply da classe LocalRemoveAudioBlockFromSession

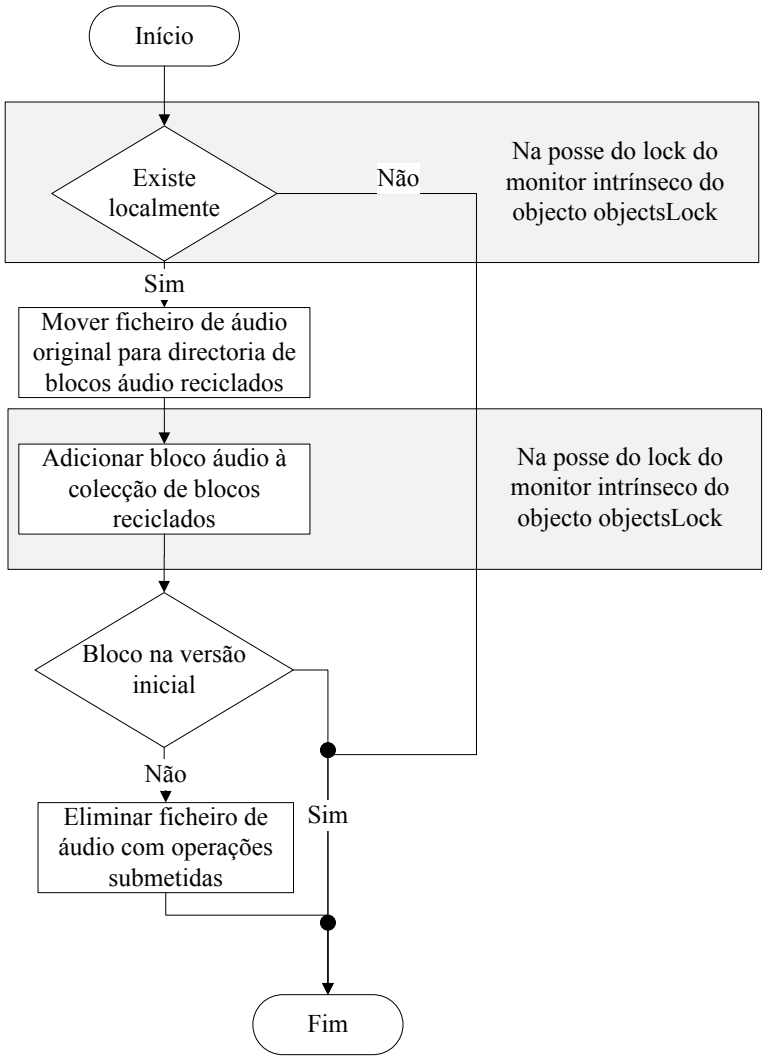

Figura 39 - Fluxograma do método Submit da classe LocalRemoveAudioBlockFromSession

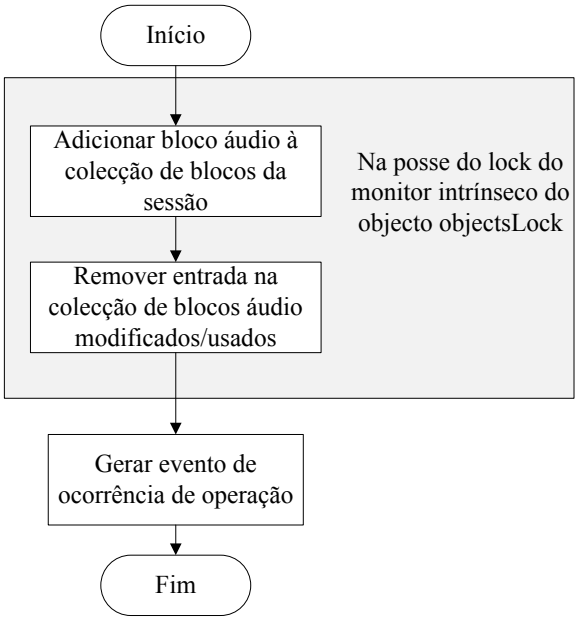

Figura 40 - Fluxograma do método Undo da classe LocalRemoveAudioBlockFromSession

A classe **LocalManager** é também responsável pela aplicação das operações recebidas. Para realizar este trabalho, existem os métodos ApplyOperationGroups e UndoRejectedOperationGroups. Estes métodos não fazem parte da interface pública da classe (visibilidade *internal*) pois são invocados pelo gestor de colaboração (**CollaborationManager**) que faz parte da mesma biblioteca. O primeiro aplica novas operações recebidas enquanto o segundo desfaz operações que já foram aplicadas localmente.

A aplicação de operações (método ApplyOperationGroups) tem duas fases: pré-processamento e aplicação de operações. A primeira fase consiste em, para cada lote de operações a aplicar, obter os objectos que este modifica/utiliza. Na posse desta informação é possível verificar se existe conflito entre as operações recebidas e o lote de operações local. Além da verificação com o lote local, verifica-se no registo de operações (recorrendo à instância de **CollaborationManager**) se existem lotes concorrentes ao lote a ser pré-processado. Em caso afirmativo, verifica-se se existe conflito entre os lotes concorrentes (modificação dos mesmo objectos). Uma última verificação a efectuar, consiste na possibilidade de, apesar de não existir conflito com o lote actual, este modificar/utilizar objectos que estão em determinado estado devido aos lotes já submetidos que estão em conflito com o que se encontra em pré-processamento. Neste caso, por sequência de acontecimentos, também existe conflito com o lote local actual.

Na posse de toda a informação acerca de conflitos, se estes existirem é consultado o utilizador para realizar a resolução dos conflitos. Aqui, o utilizador escolhe se pretende manter o lote local actual e os já registados ou aceitar o novo lote recebido. Para obter a resposta a esta pergunta, é gerado um evento do tipo **OperationGroupConflict**. Se o utilizador rejeitar o novo lote, este não é aplicado e é criada uma notificação de rejeição de lote a enviar aos restantes colaboradores. Caso o utilizador aceite o novo lote é criada uma notificação de aceitação deste, o lote local actual é desfeito (se estiver em conflito) e os lotes submetidos em conflito são registados para serem desfeitos após o pré-processamento.

Para terminar o pré-processamento, existe uma preparação dos lotes submetidos a desfazer, sendo também aqui criadas notificações de rejeição dos lotes que vão ser desfeitos, que serão enviadas aos restantes sites para que estes cancelem os lotes (se já

aplicados). Cada vez que é iniciado o pré-processamento de um novo lote, se algum dos lotes recebidos rejeitado tiver de ser aplicado antes do actual (operações prévias sobre algum objecto) este novo lote é automaticamente dado como rejeitado também.

Terminado o pré-processamento, são desfeitos os lotes em conflito de acordo com a arbitragem do utilizador e aplicados os novos lotes que foram aceites e os que não estão em conflito. Após cada uma destas duas operações são actualizadas as versões dos objectos.

Neste processo de aplicação de novos lotes recebidos, para cada operação (a aplicar ou a desfazer) é utilizada uma instância de um tipo que concretiza a classe abstracta **LoggedOperationApplier** (Figura 41). Na fase de pré-processamento é invocado o método PreProcess, para preencher as colecções de objectos modificados/utilizados. No caso da preparação das operações a desfazer, é usado o método PreProcessUndo. Depois da fase de pré-processamento, os métodos ApplyAndSubmit e Undo são usados para aplicar a operação do novo lote e desfazer operação de um lote em conflito respectivamente. A classe **LoggedOperationApplier** é utilizada neste caso, como em operações locais é usada a classe **LocalOperationApplier** explicada previamente. A principal diferença é que enquanto uma operação local pode ser desfeita enquanto não é submetida (utilizador desiste ou existe conflito com operações recebidas), existe um método para aplicar e outros para submeter ou cancelar. No caso das operações recebidas, quando é aceite a sua aplicação é também aceite a sua submissão, passando-se o mesmo com o cancelamento de operações já aplicadas.

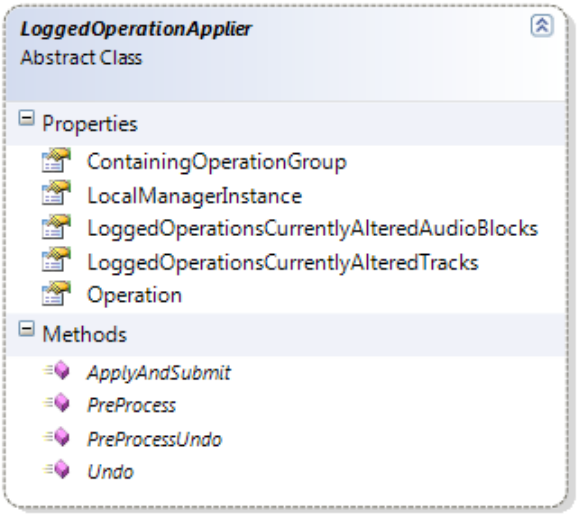

Figura 41 - Classe abstract LoggedOperationApplier

Observa-se na Figura 42, o fluxograma que descreve o processo de aplicação de novas operações recebidas.

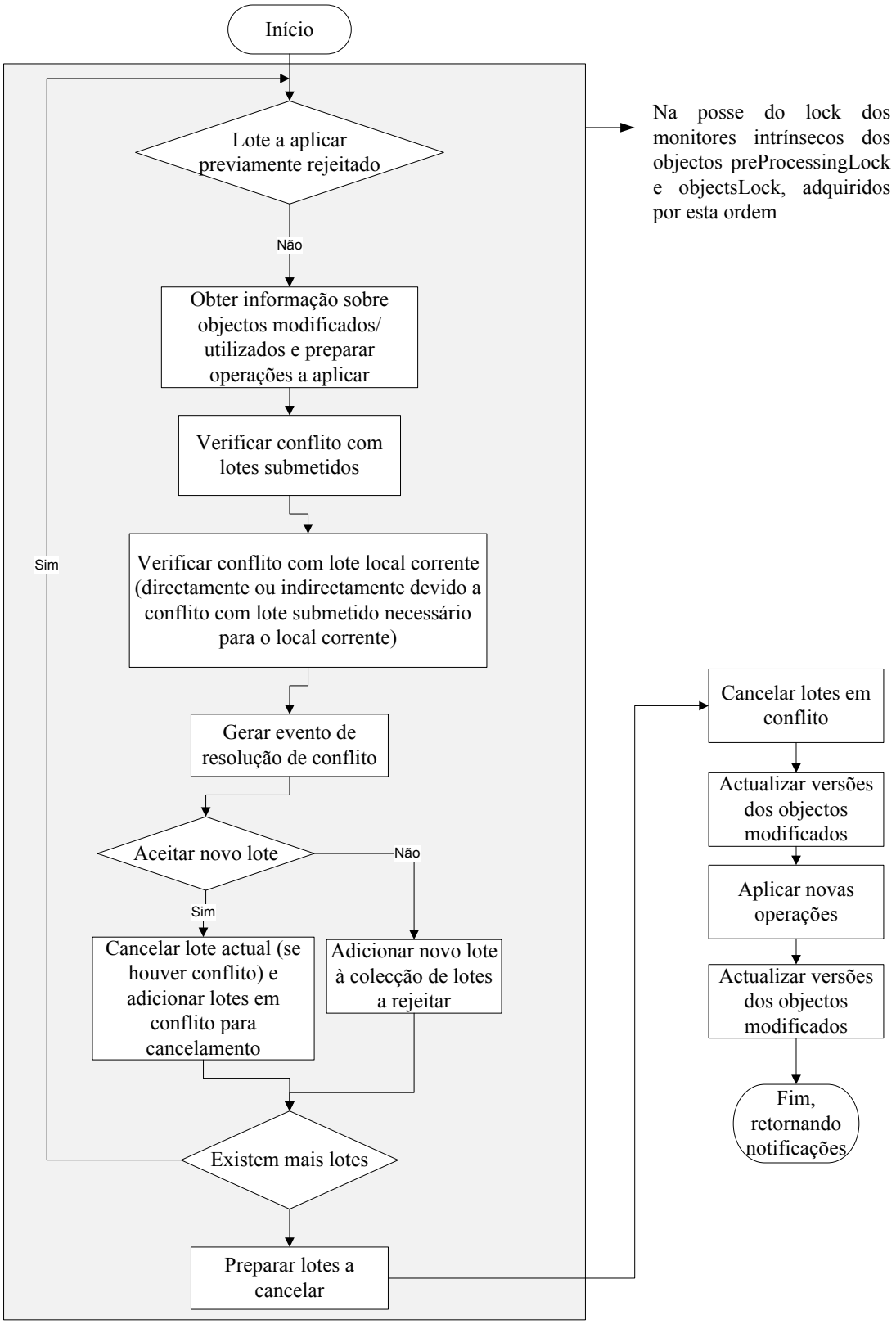

Figura 42 - Fluxograma da aplicação de lotes recebidos

O cancelamento de operações já aplicadas (método UndoRejectedOperationGroups) está, tal como a aplicação de novas operações, divido em duas fases: pré-processamento e cancelamento de lotes submetidos. A fase de pré-processamento consiste em, para cada lote a cancelar, obter os lotes sequenciais (que só podem estar aplicados se o lote a rejeitar também o estiver) e obter informação sobre os objectos modificados/utilizados pelas suas operações. Após obter esta informação, verifica-se a existência de conflito entre as operações a desfazer e o lote local corrente. Em caso de conflito, o lote local corrente é cancelado, sem haver hipótese por parte do utilizador de rejeitar, já que este é cancelado devido à rejeição por parte de outro utilizador.

Terminado o pré-processamento, os lotes são cancelados e o método termina. Na Figura 43 observa-se o fluxograma que descreve o processo de cancelamento de lotes submetidos atrás apresentado.

Tal como na aplicação de novas operações recebidas, são utilizadas instâncias de concretizações da classe abstracta **LoggedOperationApplier** para preparar e realizar o cancelamento de cada operação.

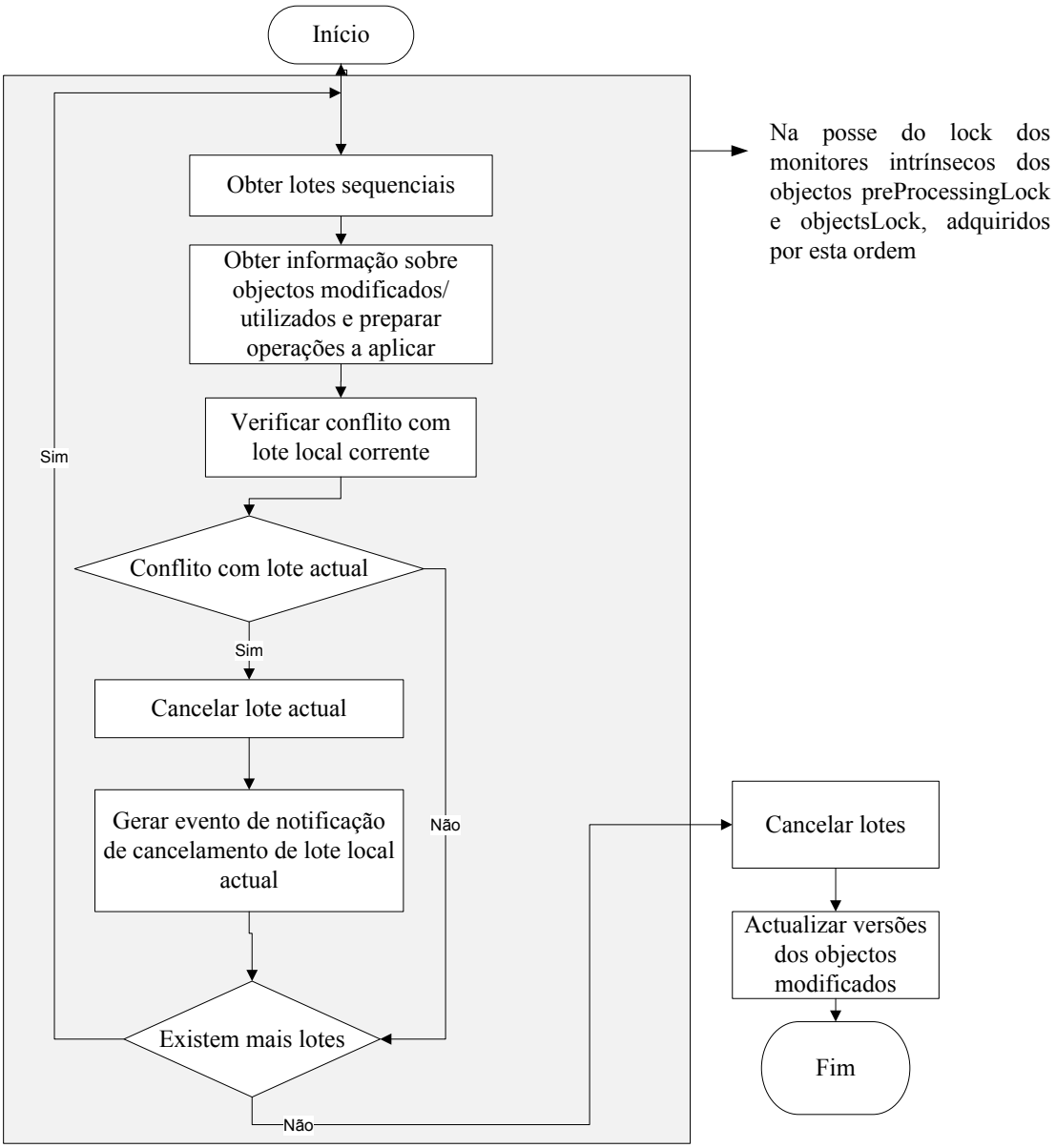

Figura 43 - Fluxograma do processo de cancelamento de lotes de operações submetidos

4.4.2.2 – Implementação do gestor de colaboração (CollaborationManager)

O gestor de colaboração é responsável pela comunicação com o servidor, gestão do processo de aplicação de operações recebidas e propagação de operações. O gestor mantém ainda um registo de acontecimentos (lotes de operações e notificações de aceitação/rejeição de lotes submetidos) para a sessão.

A comunicação com o servidor consiste no registo, ligar e desligar o utilizador, iniciar e terminar o trabalho sobre uma sessão, bem como criação de bandas e sessões. A classe **CollaborationManager** apresenta métodos que comunicam com o servidor para

realizar estas operações. Estes métodos são invocados na classe **LocalManager**, que expõe essas funcionalidades para serem usadas na aplicação cliente.

Ao executar o método para ligar o utilizador, antes de comunicar com o servidor o gestor de colaboração realiza a hospedagem dos serviços em WCF. São instanciados objectos dos tipos que representam os serviços (**ClientSiteService**, **SiteFileTransferService** e **SiteOperationTransferService**). Nestas instâncias o gestor de colaboração regista-se nos eventos para processamento de mensagens. De seguida os serviços são hospedados, fornecendo as instâncias dos tipos dos serviços, seguindo um padrão de desenho *singleton*, sendo cada instância de um serviço a única a receber todos os pedidos para aquele tipo de serviço. Quando é feito *sign out* a hospedagem dos serviços é terminada.

O processo de aplicação de operações recebidas e propagação de operações começa assim que se escolhe a sessão sobre a qual trabalhar. Ao escolher a sessão (método LogInToSession) é lançada uma *thread* que vai manter-se até que o utilizador termine o trabalho sobre a sessão. A *thread* irá executar, em paralelo com o trabalho feito localmente pelo utilizador, a aplicação e propagação de operações. Usa-se uma única *thread*, simplificando questões de concorrência que assim existe apenas entre as operações recebidas e o trabalho realizado localmente pelo utilizador. Para futuras referências, atribui-se o nome de *BackgroundWork* às tarefas realizadas por esta *thread*.

Na comunicação entre colaboradores, apenas duas mensagens têm resposta imediata: o pedido de lotes e notificações e o pedido de blocos áudio. Quando chega ao gestor de colaboração um pedido de lotes e notificações, este traz como parâmetro para a obtenção destes dados um relógio vectorial com o estado actual do site que fez o pedido. Usando esse relógio vectorial, o gestor local retorna apenas os lotes e notificações posteriores ou concorrentes aos que existem no site que realizou o pedido. Também com esse relógio vectorial, o gestor consegue verificar se o site que realizou o pedido tem lotes ou notificações que não existem localmente. Se tiver, isso é registado e será feito um pedido no processo de *BackgroundWork*.

Quanto ao pedido de bloco áudio, assim que recebido, se esse bloco existir localmente, é iniciado o processo de transferência.

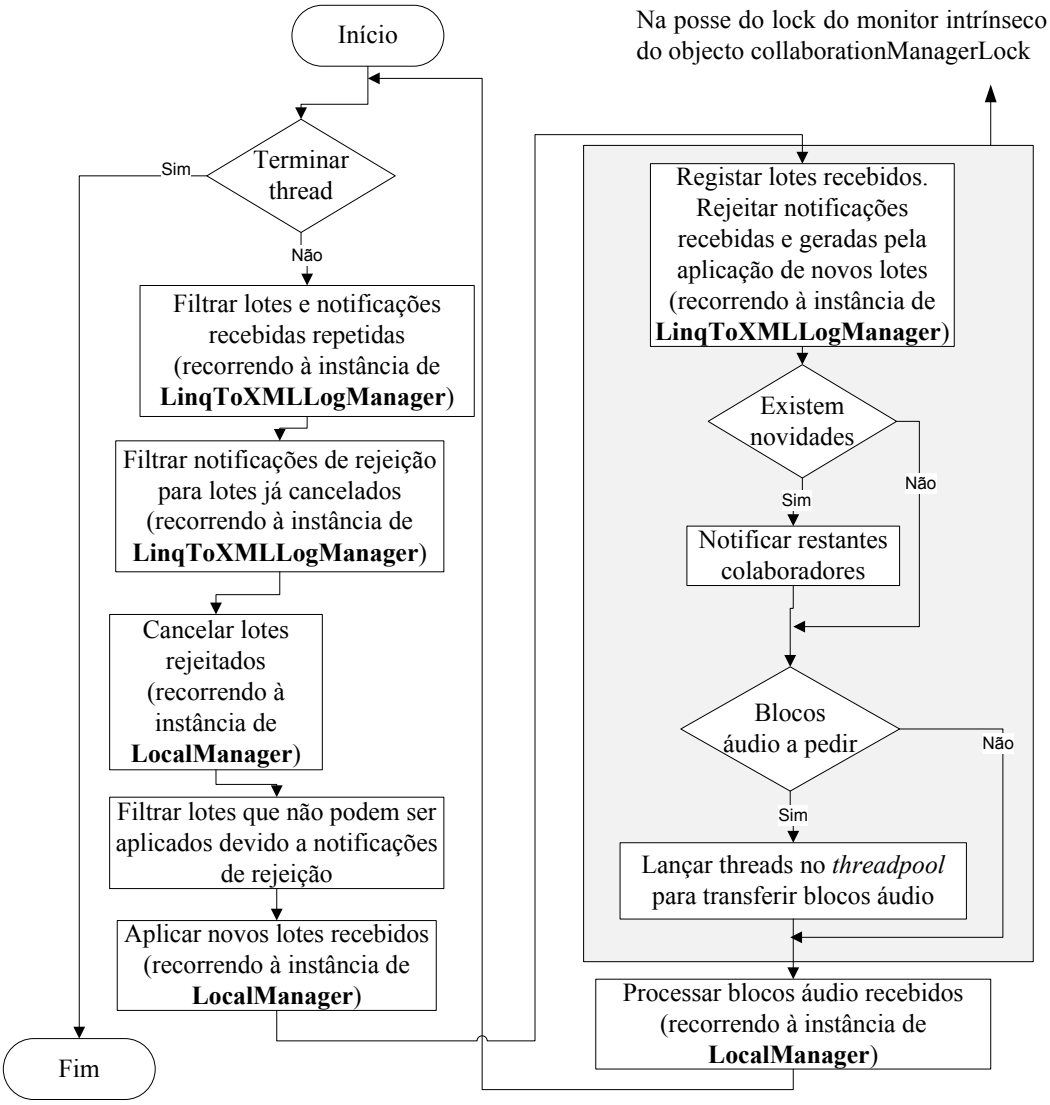

Figura 44 - Fluxograma do processo de *BackgroundWork*

No que diz respeito ao processo de *BackgroundWork*, pode-se observar um fluxograma que o descreve na Figura 44. O código é realizado num ciclo *while* que apenas termina quando se pretende parar de trabalhar na sessão. Cada vez que volta ao início do ciclo, a *thread* bloqueia-se num objecto do tipo **AutoResetEvent**. O objecto é sinalizado sempre que existe trabalho a fazer, desbloqueando a *thread*.

O processo começa com a filtragem de lotes e notificações que já foram tratados localmente. Estes lotes e notificações são obtidos quando são notificados os restantes sites (aparece no lado direito do fluxograma) e são guardados numa colecção. A filtragem remove da colecção de lotes e notificações a processar aqueles que já haviam sido recebidos (e se encontram registados).

Uma vez filtrados os lotes e notificações, o gestor de colaboração invoca o método UndoRejectedOperationGroups de **LocalManager** para cancelar os lotes que foram rejeitados, de acordo com as notificações recebidas.

Dos lotes recebidos, retiram-se os que foram rejeitados e os que não podem ser aplicados devido a rejeição de lotes prévios. Com os lotes filtrados, é invocado o método ApplyOperationGroups de **LocalManager**, para aplicar os novos lotes.

Depois do cancelamento e aplicação de lotes, actualiza-se o registo acrescentando os novos dados recebidos, bem como os gerados (notificações geradas em ApplyOperationGroups). Com esta actualização, também o relógio vectorial local, mantido pelo **CollaborationManager** é actualizado.

Para terminar, se ao longo do processo ocorreram operações, serão notificados os restantes sites, sendo enviado o relógio vectorial local. Se das operações aplicadas fizer parte uma operação de adição de bloco áudio à sessão, é necessário realizar o pedido de transferência desse bloco. É lançada uma thread do *threadpool* para realizar a transferência. Se já existirem transferências concluídas, é realizado pelo método ProcessReceivedAudioBlocks de **LocalManager** o trabalho necessário a ter os novos blocos disponíveis para utilização (guardar na directoria correcta, aplicar as operações pendentes e notificar a interface gráfica).

O registo de lotes de operações e notificações é feito através da classe **LinqToXMLLogManager** que implementa a interface **ILogManager**. Esta classe recebe estes elementos, cria uma representação XML destes e coloca-os no log (fazendo também o processo inverso). Optou-se por colocar a lógica de serialização destes elementos numa classe à parte em detrimento de adicionar a cada objecto um método para esse efeito, evitando compromissos ao nível da tecnologia. Desta forma, os objectos de transferência podem manter-se sem alterações independentemente da forma como são serializados, cabendo à implementação de **ILogManager** essa serialização. O ficheiro de registo apresenta-se em linguagem XML, por ser uma linguagem facilmente legível para um humano (importante na fase de desenvolvimento) e já existirem *parsers* disponíveis. Linq To XML é uma tecnologia para a criação e leitura de ficheiros XML.

5 – Interface com o utilizador

Este capítulo apresenta a aplicação cliente. Esta tem a interface gráfica a ser utilizada pelos membros de uma banda para desenvolver o seu trabalho. A implementação recorre aos elementos de suporte à manipulação de áudio e suporte ao trabalho colaborativo desenvolvidos. Na Figura 45 observa-se a camada da aplicação abordada neste capítulo.

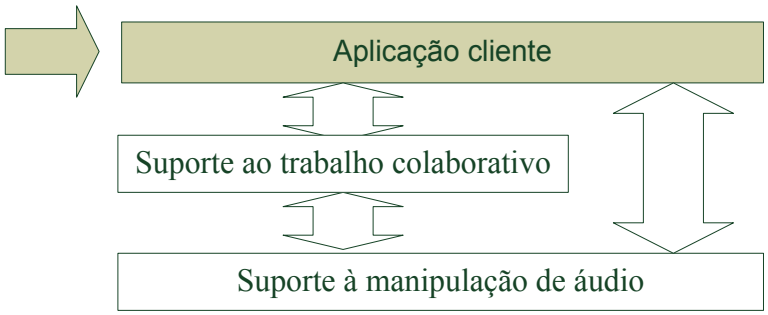

Figura 45 - Elementos constituintes da aplicação - aplicação cliente

A interface gráfica desenvolvida para a aplicação é bastante simples, disponibilizando apenas as funcionalidades indispensáveis para verificar o funcionamento do trabalho desenvolvido. O maior ênfase no desenvolvimento foi colocado nas componentes colaborativa e de processamento de áudio. A interface gráfica corresponde à aplicação **DesktopClient** e está implementada em WPF (Windows Presentation Foundation).

Entre as funcionalidades e requisitos da interface gráfica tem-se:

- Validação dos dados introduzidos pelo utilizador.
- Possibilidade de escolher a língua da aplicação, utilizando *resource files* para manter o texto da aplicação nas várias línguas.
- A interface não bloqueia (deixa de responder ao utilizador).
- Visualização a disposição dos blocos áudio ao longo das faixas, assim como a representação gráfica dos blocos, para facilitar na composição da música.
- Selecção de parte de um bloco áudio sobre o qual aplicar uma edição.
- A interface actualiza-se quando ocorrem modificações sobre a sessão de trabalho.

Para evitar o bloqueio da interface gráfica, sempre que são efectuadas operações possivelmente demoradas, como a aplicação de uma edição áudio ou fazer *login* numa

sessão, este trabalho é realizado numa thread do *thread pool*. Enquanto esse trabalho é realizado, a interface gráfica apresenta uma animação.

Na Figura 46 observa-se a janela de trabalho sobre uma sessão, onde se podem visualizar os blocos áudio disponíveis na sessão (do lado esquerdo), a disposição dos blocos áudio nas faixas (bem como a sua representação gráfica). Também se podem ver alguns botões em baixo do lado esquerdo. Estes botões servem para iniciar, submeter e cancelar um lote de operações. É possível escolher o tempo de início de gravação ou reprodução e posteriormente iniciar essas tarefas com os controlos que se encontram na zona central em baixo da janela.

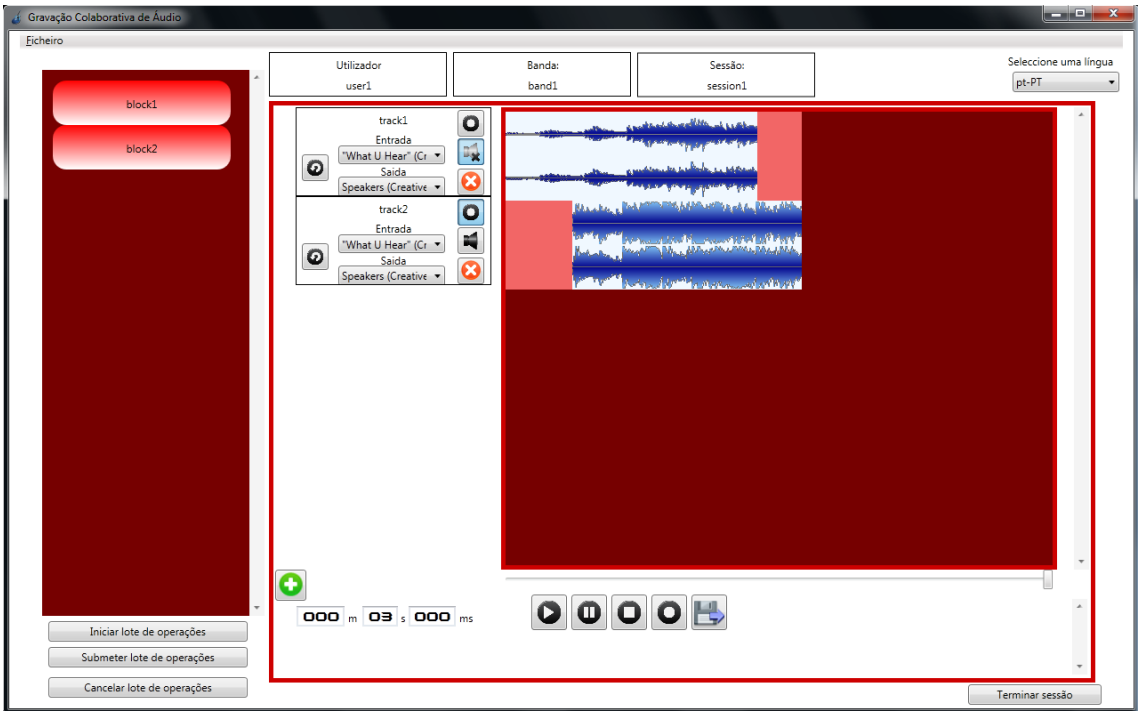

Figura 46 - Janela de trabalho numa sessão

Na Figura 47 observa-se a selecção de parte de um bloco áudio, em que a parte seleccionada apresenta uma cor diferente do restante gráfico. Na Figura 48 observa-se o mesmo bloco áudio da Figura 47 após a aplicação de ecos.

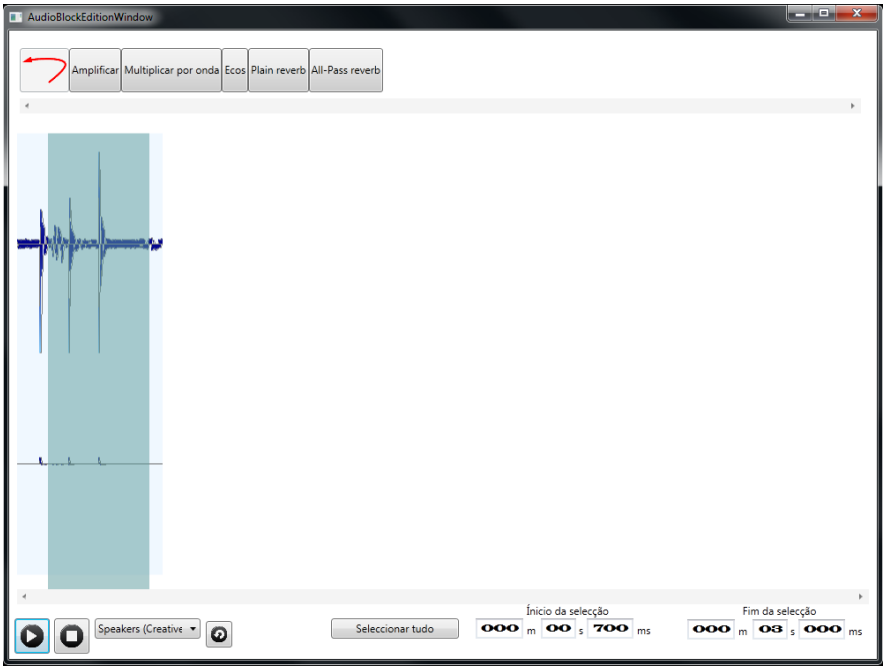

Figura 47 - Janela de edição de áudio - selecção de parte de um bloco áudio

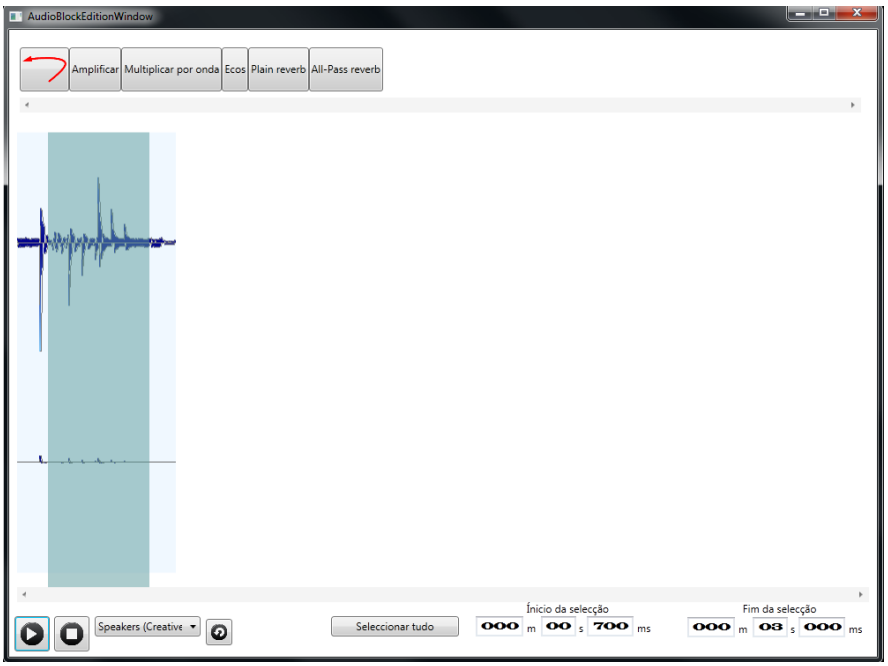

Figura 48 – Janela de edição de áudio – bloco áudio da Figura 47 após aplicação de ecos

Os controlos utilizados na aplicação registam-se no evento OnOperationOccurred da classe **LocalManager**. Desta forma, quando existem modificações aos objectos da sessão a interface pode actualizar-se e apresentar correctamente o estado actual do trabalho ao utilizador.

A aplicação regista-se ainda no evento OperationGroupConflict da classe **LocalManager**. Este evento é gerado quando existe conflito entre um lote recebido e o lote local actual ou os lotes já submetidos localmente. Quando este evento é gerado, surge uma janela para que o utilizador resolva o conflito (escolha aceitar ou rejeitar o novo lote).

Na Figura 49 observa-se a janela de resolução de conflitos. Do lado esquerdo aparecem os lotes que foram aplicados localmente e se encontram em conflito com um lote recebido, que aparece do lado direito. Para verificação dos lotes, são apresentados os pares site e valor do relógio vectorial que identifica cada lote. Antes dos botões de aceitar ou rejeitar novos lotes, é apresentada a indicação de existência de conflito com o lote actual que ainda não foi submetido.

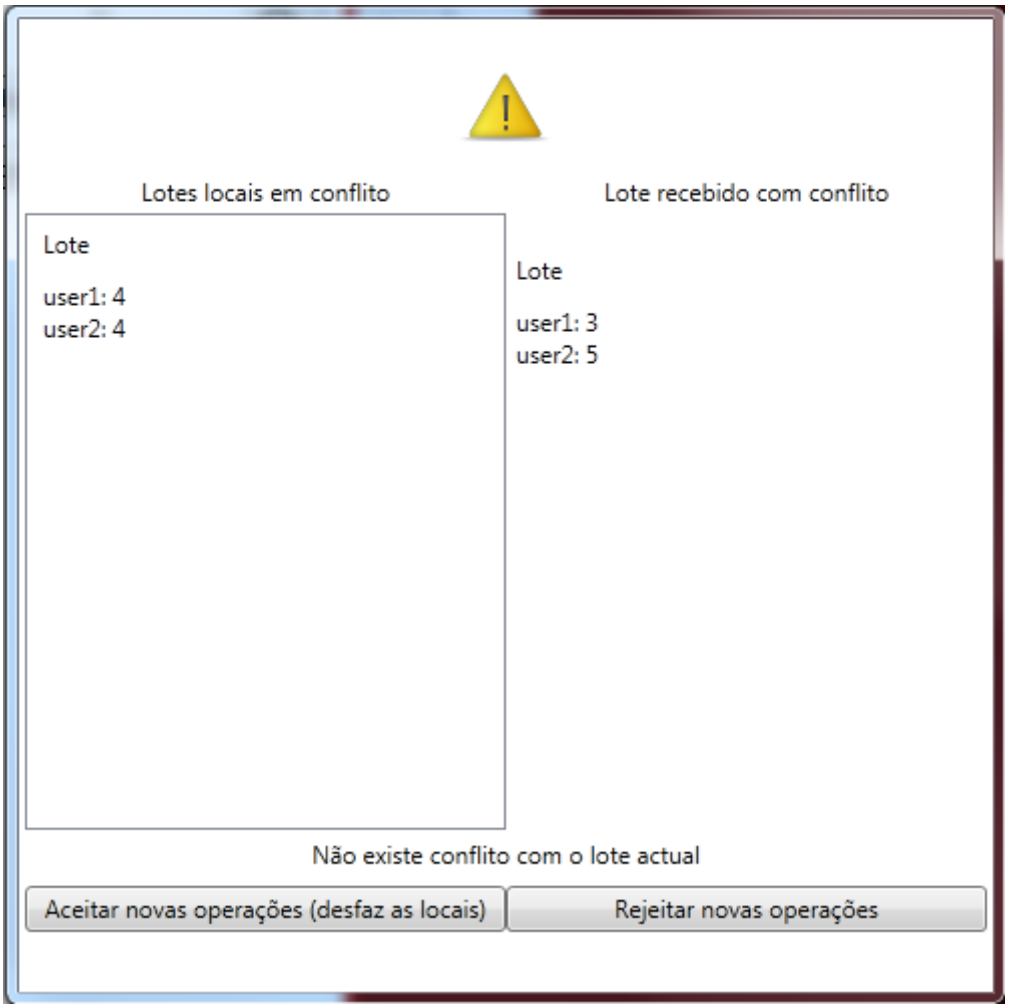

Figura 49 - Janela de resolução de conflitos por arbitragem

6 – Conclusões e trabalho futuro

Ao longo deste trabalho foi desenvolvida uma aplicação distribuída para a criação de música, introduzindo a colaboração entre utilizadores em localizações diferentes.

6.1 – Características essenciais

A aplicação apresenta funcionalidades de manipulação de áudio: gravação, reprodução, codificação e edição. A gravação e a reprodução permitem o uso de múltiplos dispositivos de entrada (no caso da gravação) e saída, mantendo semelhança com outros programas de gravação e edição de áudio em multi-faixa.

São suportadas codificações nos formatos MP3 e FLAC, podendo o MP3 ser utilizado para partilhar o resultado final de uma sessão de trabalho e o FLAC para armazenar e transferir ficheiros sem perda de qualidade, com dimensão inferior aos ficheiros WAVE.

Para a edição de áudio, existem algumas opções, nomeadamente a amplificação/atenuação de um bloco áudio, aplicação de ecos e *reverb*.

No contexto do trabalho colaborativo, tem-se o habitual ambiente multi-faixa com a adição da sincronização de operações realizadas por vários membros de uma banda. Esta sincronização envolve a transferência de ficheiros áudio gravados, a propagação de operações realizadas na sessão de trabalho (com as faixas e blocos áudio) e detecção e resolução de conflitos existentes quando os membros da banda modificam de forma concorrente os mesmos objectos.

6.2 – Conceitos e tecnologias

O resultado obtido é um protótipo ainda longe de uma aplicação comercial, mas que apresenta ideias base que uma aplicação deste género pode seguir. Entre estas ideias, a destacar, num caso em que pretendermos diminuir a carga num servidor, o uso de replicação optimista, com manutenção das réplicas nos computadores dos colaboradores. A propagação de operações, com a poupança de tráfego que traz em comparação com a transferência de objectos (nomeadamente ficheiros áudio), bem

como o recurso a um registo de operações para transitar entre versões de objectos são também ideias interessantes.

Para a realização do trabalho foram desenvolvidos conhecimentos sobre ideias e tecnologias abordadas ao longo do curso, bem como adquiridos novos conhecimentos, sustentados na aprendizagem feita ao longo do curso.

O principal conceito introduzido neste trabalho que não havia sido abordado previamente no curso prende-se com a utilização de técnicas de replicação optimista.

Foi utilizada como tecnologia base do trabalho a Plataforma .NET na sua versão 4.0. No contexto desta tecnologia foi utilizado WCF para a implementação da componente relativa à comunicação, WPF para a implementação da interface gráfica, Linq To XML para a manipulação de ficheiros XML e Linq To Entities na camada de acesso à base de dados. Outras tecnologias Microsoft utilizadas são SQL Server 2008, filas de mensagens MSMQ e o servidor IIS (Internet Information Services) para testar a hospedagem do servidor da aplicação.

Destas tecnologias, WPF, Linq To XML e o servidor IIS não haviam sido abordadas ao longo do curso. Nas restantes foram consolidados e aprofundados os conhecimentos adquiridos.

Outros recursos usados, também nunca utilizados antes deste trabalho, são os codificadores FLAC e LAME, a biblioteca ZLib e a biblioteca BASS.

6.3 - Trabalho futuro

Para transformar o protótipo implementado numa aplicação comercial, existe algum trabalho futuro a desenvolver.

Uma aplicação para a criação de música necessita de uma maior gama de operações a realizar sobre áudio. No trabalho realizado, não é suportada a manipulação de áudio nos formatos inteiro com sinal a 24 bit nem vírgula flutuante a 32 bit (não é suportado na codificação e descodificação). O suporte a estes formatos é importante numa aplicação deste tipo.

A interface gráfica deve ser mais rica, mais apelativa e com mais informação para o utilizador (incluindo operações que vão ocorrendo, utilizadores ligados entre outros).

Uma situação que não ficou resolvida no protótipo foi a tolerância a falhas. Se durante o trabalho numa sessão houver um problema com o computador e a aplicação for encerrada abruptamente, os dados da sessão podem ficar inconsistentes, com dados intermédios guardados em disco.

Referências

[1]. **Goldwave Inc.** GoldWave - Audio Editor, Recorder, Converter, Restoration, & Analysis Software. [Online] Goldwave Inc. [Citação: 5 de Março de 2010.] http://www.goldwave.com/.

[2]. —. MULTIQUENCE - Audio/Video Editing, Mixing and Sequencing Software. [Online] Goldwave Inc. [Citação: 5 de Março de 2010.] http://www.goldwave.com/mqindex.php.

[3]. **Adobe.** Adobe - Audition 3. [Online] Adobe. [Citação: 5 de Março de 2010.] http://www.adobe.com/products/audition/.

[4]. **Drew, Mark S. e Li, Ze-Nian.** *Fundamentals of Multimedia.* s.l. : Pearson Prentice Hall, 2004. ISBN-10 013127256X, ISBN-13 978-0131272569.

[5]. **Orfanidis, Sophocles J.** *Introduction to Signal Processing.* Upper Saddle River, NJ 07458 : Prentice Hall, 1996. ISBN: 0-13-209172-0.

[6]. **Oppenheim, Alan V., Schafer, Ronald W. e Buck, John R.** *Discrete-Time Signal Processing, 2nd edition.* s.l. : Prentice Hall, 1999. ISBN: 0130834432.

[7]. **Nilsson, M.** The audio/mpeg Media Type. [Online] Novembro de 2000. [Citação: 5 de Março de 2010.] http://tools.ietf.org/html/rfc3003.

[8]. **FLAC.** *FLAC - Free Lossless Audio Codec.* [Online] [Citação: 18 de Fevereiro de 2010.] http://flac.sourceforge.net/.

[9]. **Gailly, Jean-loup e Adler, Mark.** *The gzip home page.* [Online] [Citação: 12 de Março de 2010.] http://www.gzip.org/.

[10]. **PKWARE, Inc.** *.ZIP Application Note.* [Online] [Citação: 12 de Março de 2010.] http://www.pkware.com/documents/casestudies/APPNOTE.TXT.

[11]. **Un4seen Developments.** Un4seen Developments - 2MIDI / BASS / MID2XM / MO3 / XM-EXE / XMPlay. *Un4seen Developments - 2MIDI / BASS / MID2XM / MO3 /*

XM-EXE / XMPlay. [Online] Un4seen Developments. [Citação: 9 de Fevereiro de 2010.] http://www.un4seen.com/.

[12]. **Microsoft.** *Introducing DirectX 9.0 for Managed Code.* [Online] [Citação: 12 de Março de 2010.] http://msdn.microsoft.com/en-us/library/bb318659(VS.85).aspx.

[13]. **FFmpeg.** *FFmpeg.* [Online] http://ffmpeg.org/.

[14]. **Baskin, Dave F.** *ZLib .NET Wrapper.* [Online] [Citação: 9 de Fevereiro de 2010.] http://zlibnetwrapper.sourceforge.net/.

[15]. **RAREWARES.** LAME Bundles. *RAREWARES.* [Online] [Citação: 18 de Fevereiro de 2010.] http://www.rarewares.org/mp3-lame-bundle.php.

[16]. **Microsoft.** Microsoft Windows Bitmap Format. [Online] [Citação: 24 de Agosto de 2010.] http://www.fileformat.info/format/bmp/spec/e27073c25463436f8a64fa789c886d9c/BMP.TXT.

[17]. **Saito, Yasushi e Shapiro, Marc.** *Optimistic Replication.* Broadway, New York, NY : ACM, 2005. 0360-0300/05/0300-0042.

[18]. *Virtual Time and Global States of Distributed Systems.* **Mattern, Friedemann.** 1988.

[19]. **Bishop, Matt.** *Introduction to Computer Security.* s.l. : Prentice Hall PTR, 2004. ISBN: 0-321-24744-2.

[20]. **Rivest, R.** The MD5 Message-Digest Algorithm. [Online] Abril de 1992. [Citação: 11 de Setembro de 2010.] http://www.ietf.org/rfc/rfc1321.txt.

[21]. *Detection of mutual inconsistency in distributed systems.* **Parker, Jr., D. Stott, et al.** s.l. : IEEE Trans. Softw. Eng., 1983. 0098-5589/83/0500-0240.