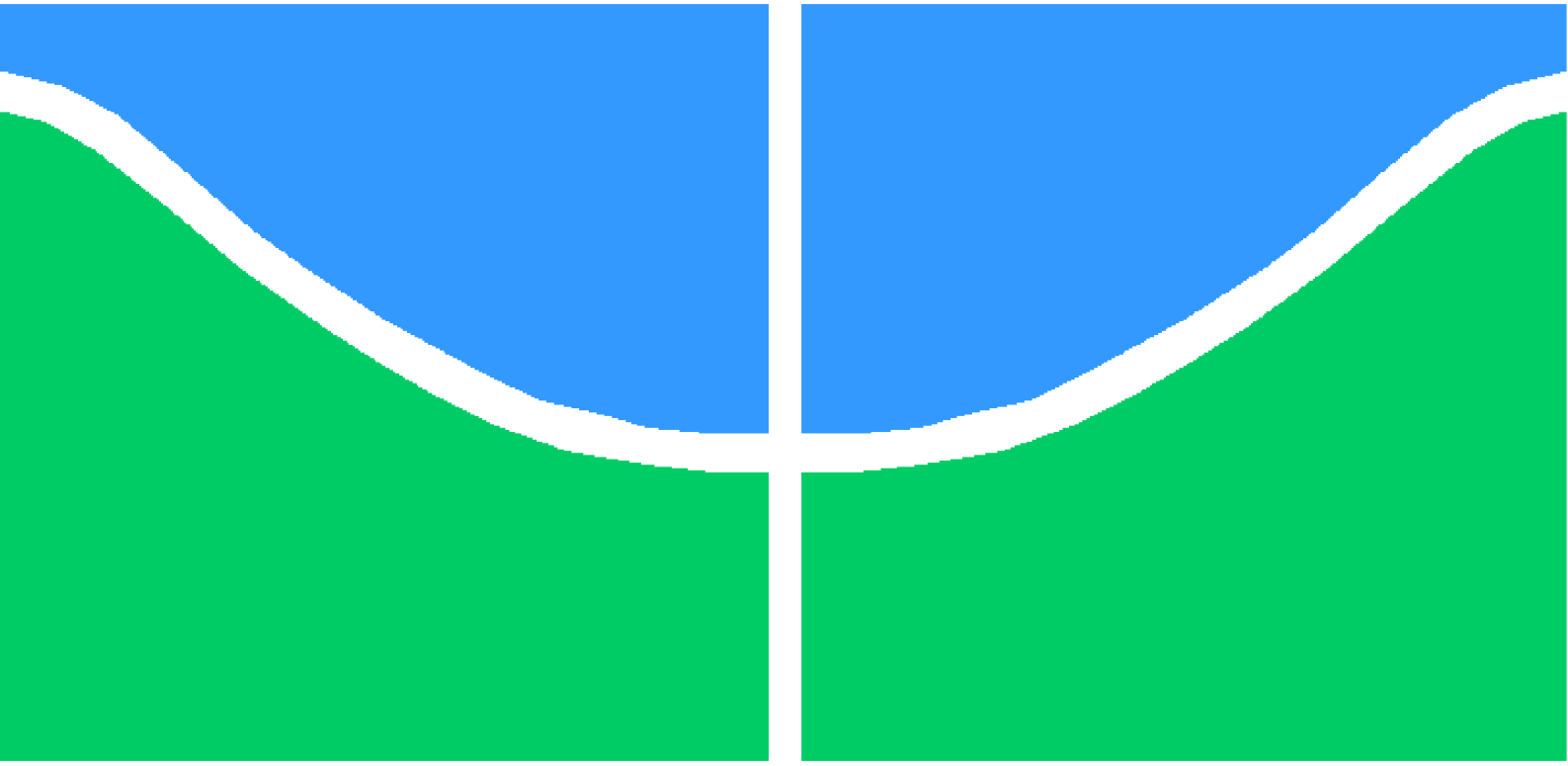

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de *Software*

Especificação de um formato de arquivo *open source* para *audiobooks* com suporte a marcações de conteúdo

Autor: Herbert Costa dos Reis

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2013

Herbert Costa dos Reis

Especificação de um formato de arquivo *open source* para *audiobooks* com suporte a marcações de conteúdo

Monografia submetida ao curso de graduação em Engenharia de *Software* da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de *Software* .

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2013

Herbert Costa dos Reis

Especificação de um formato de arquivo *open source* para *audiobooks* com suporte a marcações de conteúdo/ Herbert Costa dos Reis. – Brasília, DF, 2013-

201 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2013.

1. audiobook. 2. marcação de conteúdo. I. Prof. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Especificação de um formato de arquivo *open source* para *audiobooks* com suporte a marcações de conteúdo

CDU 02:141:005.6

Herbert Costa dos Reis

Especificação de um formato de arquivo *open source* para *audiobooks* com suporte a marcações de conteúdo

Monografia submetida ao curso de graduação em Engenharia de *Software* da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de *Software* .

Trabalho aprovado. Brasília, DF, 13 de Dezembro de 2013:

Prof. Dr. Edson Alves da Costa Júnior
Orientador

Prof. Dr. Sérgio Antônio Andrade de Freitas
Convidado 1

Prof. Dr. Henrique Gomes de Moura
Convidado 2

Brasília, DF
2013

Aos meus pais.

"My code doesn't work, I have no idea why.
My code works, I have no idea why".
(Desconhecido)

Resumo

Neste trabalho é desenvolvida uma especificação de um formato livre e *open source* para *audiobooks* com suporte a marcadores de conteúdo. Além da especificação em si, são desenvolvidos mais dois produtos e uma pesquisa de opinião de potenciais usuários para a validação desse novo formato. O primeiro produto é o Editor de arquivo, ele é responsável por criar os arquivos no formato *RAB* (*RIFF Audiobook*) a partir de um arquivo do tipo *WAVE*, através da inserção de metadados e de marcações de conteúdo. O segundo produto é o Tocador de *audiobooks*. Nele, o usuário tem a capacidade de navegar como quiser através das marcações inseridas no arquivo pelo Editor.

Palavras-chaves: Marcação de Conteúdo. *Audiobooks*. *WAVE*. *Open Source*.

Abstract

In this project a specification of a free and open source format for audiobooks with bookmarking content support is developed. Besides the specification itself, another two products and a survey of potential users for the validation of this new format are developed. The first product is the file editor, it is responsible for creating the files in RAB (Audiobook RIFF) format from a file of type WAVE, by inserting metadata and tags content. The second product is the audiobook Player. In it, the user has the ability to navigate through the tags inserted in the file by Editor.

Key-words: Bookmarking Content. Audiobooks. WAVE. Open Source.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

O *audiobook* (audiolivro ou livro falado) é a reprodução falada de uma óbra literária gravada em alguma mídia, como CD (*Compact Disc*), fitas cassetes, arquivos de áudio de computadores, etc. Na década de 1980, os Estados Unidos foram os pioneiros na popularização dos *audiobooks* e, atualmente, eles representam o maior mercado do mundo nesse segmento. De acordo com a APA (*Audio Publisher Association*), as vendas dos livros falados crescem numa taxa de 10 porcento ao ano e movimentam cerca de 800 milhões de dólares. Na Europa, os *audiobooks* são populares principalmente na Alemanhã e na Grã-Bretanha. Na Inglaterra, por exemplo, os audiolivros estão presentes na grande maioria das livrarias e os preços deles são muito próximos aos preços dos livros impressos (PALLETA; WATANABE; PENILHA, 2008).

O mercado brasileiro para os livros falados está em forte crescimento graças aos investimentos de empresas privadas que apostam nos grandes temas, como astrologia, autoajuda, *best-sellers*, concursos públicos, direito, ficção, literatura infantil, medicina, psicologia, entre outros (FARIAS, 2012). Outro ponto que também contribui para o esse crescimento é a compatibilidade dos *audiobooks* com os dispositivos móveis, ajudando aqueles que não tem tempo de ler em casa, mas têm tempo livre no ônibus ou no metrô.

Muito além de ser apenas um instrumento para facilitar a vida das pessoas na agitação do dia a dia, os audiolivros contribuem com um papel muito importante na inclusão social para os deficientes visuais (JESUS, 2008). Por serem arquivos digitais, os *audiobooks* apresentam uma série de benefícios (PALLETA; WATANABE; PENILHA, 2008), dentre os quais podemos citar:

- não ocupam espaços físicos nas prateleiras ou bibliotecas;
- não sofrem deteriorização com o passar do tempo, ou qualquer tipo de desgaste no decorrer do uso;
- possibilitam aos leitores a realização de outras atividades enquanto ouvem o livro.

A Audible, uma companhia da Amazon, é uma das grandes responsáveis pela difusão do *audiobook*. Ela iniciou seus trabalhos no ramo com áudio em 1997 e, já no ano seguinte, publicou um *site* no qual era possível baixar arquivos no seu formato proprietário, o AA (Audible *Audiobook File*). Em 2003, a Audible negociou com Apple a criação do catálogo de *audiobooks* na iTunes Store, disponibilizando arquivos no formato M4B (*Moving Picture Experts Group 4 Audiobook File*), variação do MP4 (*Moving Picture Experts Group Audio Layer 4*) (AUDIBLE, 2013). Segundo o próprio *site* da Apple, a iTunes

Store conta com mais de 20 mil horas de *audiobooks* disponíveis para *download* (APPLE, 2003).

A *web* é uma ótima fonte de *audiobooks* gratuitos sobre diversos temas. O Guia de Ensino [1] cita dez endereços eletrônicos que possibilitam o acesso sem custos aos *audiobooks*.

Assim como a música, os *audiobooks* são amplamente difundidos pela internet, em sua grande maioria no formato MP3 (*Moving Picture Experts Group Audio Layer 3*). Um dos motivos do sucesso da difusão desse formato é fato de ele poder armazenar áudio de alta qualidade ocupando pouco espaço em disco, por conta do uso de uma técnica de compressão que consegue diminuir consideravelmente o tamanho do arquivo (HACKER, 2000). Entretanto, como desvantagem, quaisquer marcações são ligadas diretamente ao tocador, e não ao arquivo, diferentemente dos arquivos da Audible (DAVISON, 2007).

A motivação deste trabalho está relacionada ao fato de não existir um formato de arquivo livre e de código aberto que tenha suporte a marcadores de conteúdo. Não existem formatos *open source* disponíveis com essa característica, os que possuem são protegidos por leis propriedade intelectual.

Neste trabalho, foi desenvolvida a especificação de um formato aberto (*open source*) para *audiobooks* com suporte a marcadores de conteúdo, isto é, marcadores de posição que representem a estrutura lógica de um livro físico (capítulos, seções, parágrafos, versículos, etc.). Além da especificação em si, foram desenvolvidos um Editor um Tocador (*Player*) compatíveis com o formato especificado.

A escolha do *WAVE* como formato base está pautada na simplicidade de sua estrutura e na extensibilidade, que é uma propriedade do *RIFF*. Dessa maneira, todas as capacidades e características do *RIFF* e do *WAVE* estarão presentes no novo formato especificado neste trabalho.

O trabalho está organizado em capítulo. Este capítulo apresentou a definição de *audiobooks* , o contexto e as motivações para o desenvolvimento da especificação com as marcações de conteúdo.

No capítulo 2, é apresentada toda a revisão de bibliografia necessária para o entendimento dos conceitos usados para o desenvolvimento do projeto. São apresentados os conceitos sobre o som, os formatos de arquivo de áudio, estruturas de dados, padrões de projeto, reuso de *software* e projeto e relatório de pesquisa.

No capítulo 3, são apresentadas todas as etapas realizadas para o desenvolvimento dos artefatos, a especificação do novo formato, o desenvolvimento do Editor e do Tocador e criação da pesquisa de opinião dos potenciais usuários.

No capítulo 4 são apresentados os resultados obtidos no projeto, mostrando o

[1] http://goo.gl/6yP3J

arquivo gerado pelo Editor, o comportamento do Tocador ao carregar e executar o arquivo que simulou o *audiobook* e os resultados da pesquisa de opinião.

E por último, no capítulo 5 são apresentadas as conclusões do projeto, abordando as contribuições geradas através do seu desenvolvimento e também os trabalhos futuros.

2 Fundamentação Teórica

2.1 O Som

O som é uma onda mecânica que, ao se propagar, provoca o efeito de compressão e descompressão no meio físico. Esse efeito – chamado de rarefação – é causado quando há uma acentuada variação de pressão nas moleculas do ambiente – sólido, líquido ou gasoso – no qual a onda de propaga, tal como acontece com um auto-falante. A Figura 2.1 mostra como é a propagação onda sonora no ar (HANSEN, 2001).

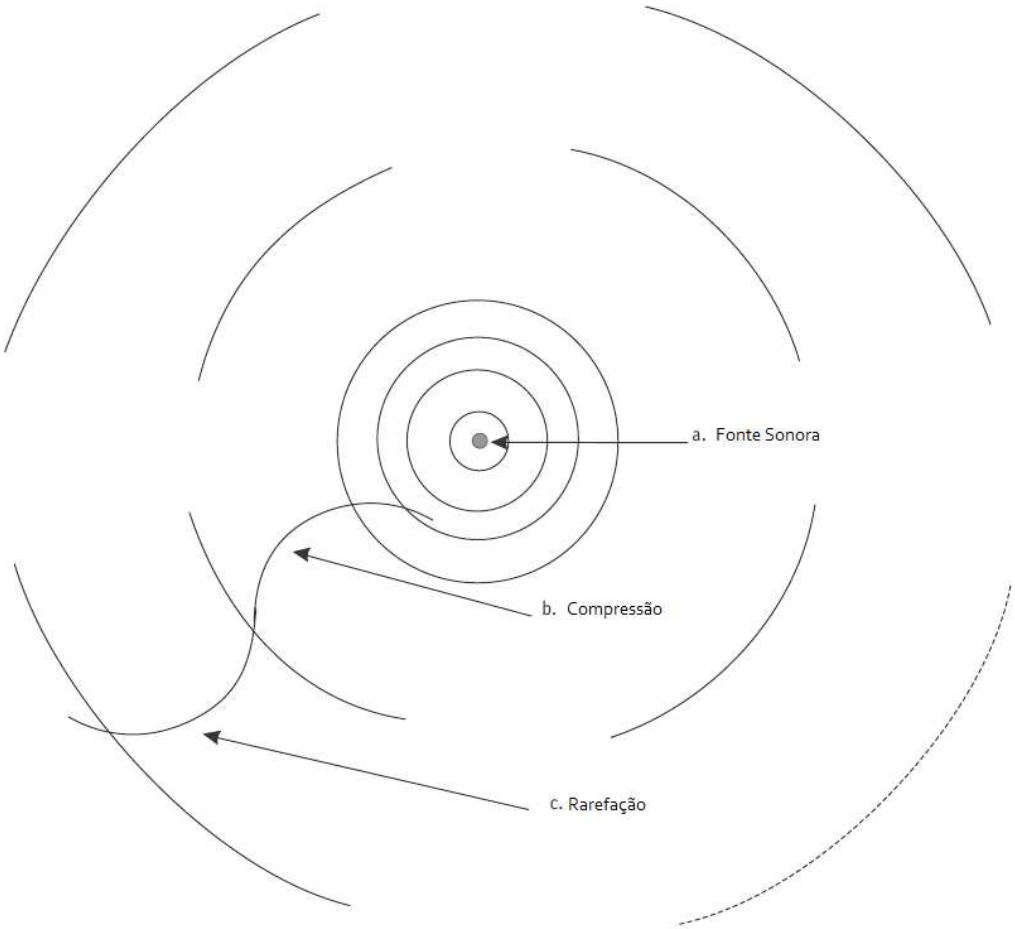

Figura 1 – Propagação do som no ar. (KEFAUVER; PATSCHKE, 2007)

A onda se propaga unidirecionalmente a partir do ponto **a**, que é a fonte sonora. A energia sonora da fonte sonora é transferida para o meio através dos movimentos de compressões (ponto **b**) e rarefações (ponto **c**) do ar .

Os componentes principais que compõem o som são a frequência, a altura, a duração, a fase, o comprimento de onda, a velocidade, modulação e o conteúdo harmônico.

A **frequência**, calculada em *hertz* (Hz), representa o número de ciclos realizados em um período de tempo, em segundos. É ela que define se o som é grave ou agudo. Os sons mais graves são aqueles que possuem frequências mais baixas, entre 0 e 300Hz. A faixa de frequência audível pelo homem está entre 20Hz e 20kHz, e, para valores diferentes desse intervalo, o corpo humano sente reações, as quais podem comprometer a sua saúde. Medida em decibéis (dB), a **altura** é a grandeza que representa a intensidade do som. O ouvido humano consegue distinguir sons na faixa de amplitude entre 0 até 120 dB. Entretanto, a partir de 90 dB, é possível que o ouvido sofra danos irreversíveis (LEWIS et al., 2010). A **duração** ou período é a grandeza representa o tempo necessário para a onda sonora complete um ciclo. Ela calculada a partir do inverso da frequência. A **fase** é o valor em graus percorrido pela onda sonora numa circunferência no sentido anti-horário. (BARBOSA, 1999)

A Figura 2 apresenta a relação entre a fase de uma onda sonora com a sua altura.

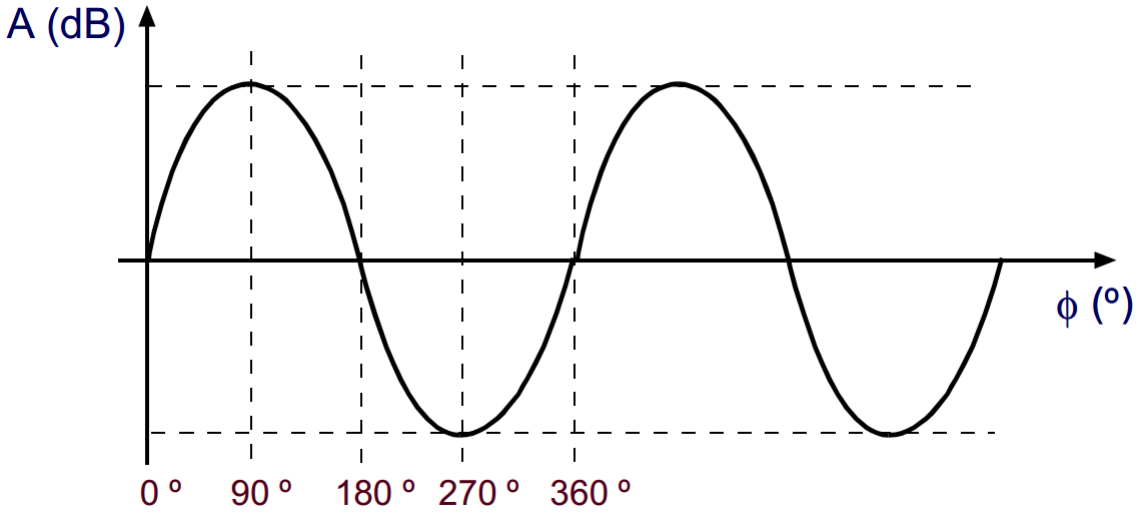

Figura 2 – Altura em função da Fase.(BARBOSA, 1999)

A **velocidade** do som ao nível do mar a 21 graus Celsius é de cerca de 344 metros por segundo. Essa velocidade depende diretamente do ambiente que o som se propaga. Por exemplo, a velocidade do som cresce à medida em que a temperatura do ambiente fica mais alta. A fórmula que expressa a velocidade do som em função da temperatura é

$$49\sqrt{490,4 + 1,8T} \tag{2.1}$$

em que T é a temperatura em graus Celsius. O **comprimento de onda** do som produzido é definido através da distância entre as sucessivas rarefações. O **conteúdo harmônico** corresponde à caracteristica que permite distinguir os sons de mesma frequência, ou seja, ele representa o timbre do som (KEFAUVER; PATSCHKE, 2007).

2.2 Formatos de Arquivos de Áudio

Os arquivos de áudio são utilizados para armazenar os dados de áudio digital em computadores, mídias, *smartphones*, entre outros aparelhos. Para que o áudio esteja pronto para o uso em qualquer plataforma digital, é necessário que seja utilizado um processo de conversão analógico/digital, a não ser que o som já tenha sido criado em um ambiente digital. O processo de conversão de um sinal de analógico para digital é baseado na quantificação do valor da amplitude do sinal em vários instantes no tempo, sendo estes valores gravados num arquivo que pode ser manipulado em um computador. Esta quantificação pontual é chamada de amostragem e é feita numa frequência definida como frequência de amostragem (número de amostras por segundo) (KEFAUVER; PATSCHKE, 2007), conforme apresentado na Figura 3.

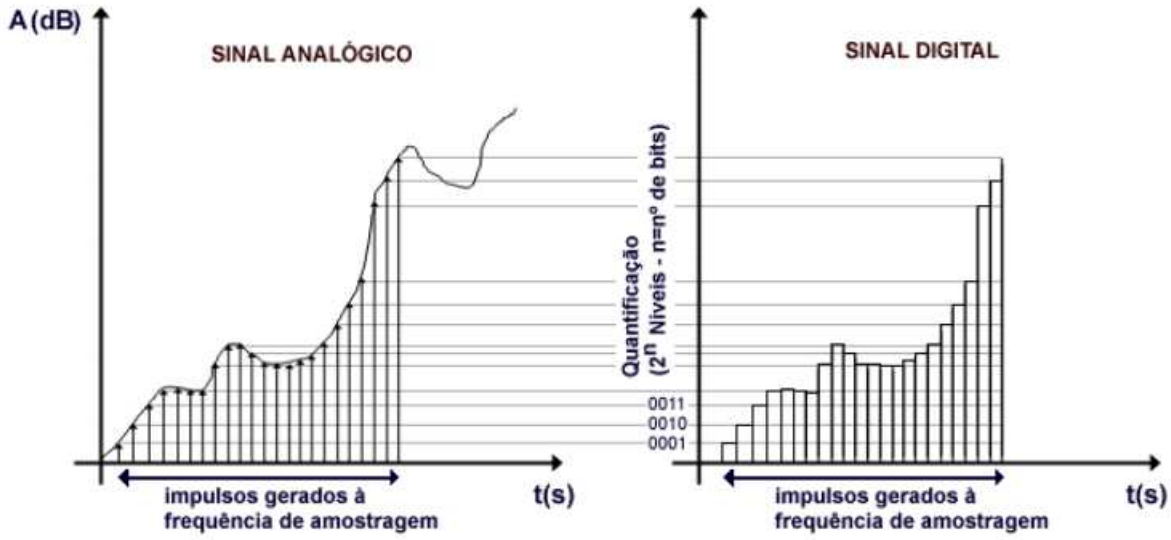

Figura 3 – Conversão de analógico/digital. (BARBOSA, 1999)

Ao utilizar a conversão analógico/digital, dois conceitos fundamentais deve ser entendidos: frequência de amostragem e quantificação. A **frequência de amostragem** pode ser entendida como a resolução com que o sinal é adquirido em termos da sua variação ao longo do tempo. Em geral, as frequências de amostragem utilizadas em gravações de CD são de 44KHz, sendo comum encontrar amostragens a 22KHz, quando se deseja reduzir o espaço que ocupado em disco (BARBOSA, 1999).

A **quantificação** corresponde ao número de níveis de amplitude que serão utilizados para amostrar o sinal, ou seja, pode-se pensar na quantificação como sendo a resolução em termos do valor de amplitude que o sinal pode ter num determinado instante. É comum quantificar o sinal utilizando 16 *bits* em gravações com qualidade de CD, sendo também normal encontrar quantificações de 8 *bits* em sinais que não precisam de uma boa qualidade. Atualmente é possível digitalizar um sinal em até 32 *bits* (BARBOSA,

1999).

A representação digital dos dados do áudio oferece vantagens interessantes: imunidade a ruidos e estabilidade. O áudio no formato digital também permite a o desenvolvimento de diversas funções de processamento de áudio, como a equalização, mixagem e filtragem (PAN, 1993).

Os dados dos arquivos de áudio podem ter sido submetidos a tecnicas de compressão, caso deseja-se reduzir o tamanho total do arquivo. A compressão do áudio digital permite a eficiência no armazenamento e na transmissão dos dados (PAN, 1993).

2.2.1 Formatos proprietários

O formato *MPEG Audio Layer 3* (MP3) é padrão mais conhecido e utilizado atualmente para o armazenamento de áudio de qualquer espécie. O MP3 utiliza um modo de compactação que visa a redução do tamanho sem comprometer a qualidade final do áudio do arquivo através de uma técnica denominada *Perceptual Audio Coding*. Ela desconsidera as frequências que são inaudíveis ou praticamente imperceptíveis ao ouvido humano, fazendo com que o arquivo MP3 tenha até menos de 10 porcento do tamanho do arquivo correspondente em outros formatos que não utilizam compactação (HACKER, 2000).

O formato M4B é o formato oficial da Apple para *audiobooks* baixados da iTunes Store. Baseado no contâiner MPEG-4, o M4B é semelhante ao M4A, e é voltado especificamente para *audiobooks* e pode ser marcado (marcação de pausa) por alguns *players* de áudio, possibilitando que o usuário volte a ouvir o livro naquele ponto onde foi pausado. Os *audiobooks* nesse formato podem ter as cópias protegidas para que sejam executados apenas em computadores, *iPods*, *iPhones* e *iPads* autorizados.

O formato AA é o oficial para os *audiobooks* da Audible. Nesse formato, é possível realizar marcações de conteúdo no arquivo para definir os capítulos do livro e também as marcações de pausa (*bookmarks*), caso o usuário *pause* o aparelho e depois volte a usá-lo. Ao fazer isso, o *player* retoma a execução a partir do local onde houve essa marcação.

O formato AAX é outro que também é voltado para *audiobooks*, também desenvolvida pela Audible. Esse formato possui as mesmas capacidades de marcações que o AA oferece, mas ele inclui suporte a imagens e vídeos. Devido a essa capacidade, este formato é normalmente usado para *audiobooks* de histórias infantis.

2.2.2 Formatos abertos

O formato WAVE (cujos arquivos possuem a extensão *.wav*) é um dos tipos de arquivo mais comuns para o armazenamento de áudio digital para computadores pessoais. A representação do áudio nesse tipo de arquivo ocorre através da **modulação por código de pulso** (*Pulse Code Modulation - PCM)*, na qual cada ponto do sinal sonoro é

amostrado e medido, obtendo-se uma sucessão de valores numéricos que codificam o som original. Nessa digitalização não há perdas e nem compressão dos dados para diminuição do tamanho do arquivo, de modo que os dados são exatamente iguais ao sinal digitalizado (IBM; MICROSOFT, 1991).

O uso de arquivos WAVE oferece algumas vantagens interessantes, como a compatibilidade com diversos *softwares* de aúdio e o fato de que não é necessário utilizar um sitentizador para a reprodução, pois uma placa de som simples já é suficiente para sua interpretação e execução. A principal desvantagem é o tamanho do arquivo final, que chegar ser até dez vezes maior do que os formatos que utilizam alguma técnica de compressão (HACKER, 2000).

O Ogg Vorbis (cujos arquivos tem extensão *.ogg*) é um formato de áudio livre mantido pela fundação Xiph.org, e é muito utilizado na internet e nas transmissões de rádio ao vivo. Em termos de qualidade, os arquivos Ogg são equivalentes a outros formatos como o MP3 e o AAC, da Apple, que são largamente utilizados no mercado.

Quando o projeto Ogg Vorbis começou, em 1993, o nome inicial a ser usado era Squish. Entretanto esse nome já estava registrado por outra empresa e, então, decidiram pelo nome de OggSquish. Esse nome foi utilizado até 2001, quando passou a ser usado apenas Ogg para referenciar o contâiner, que hoje faz parte do projeto de multimídia da Xiph.org (XIPH, 2013).

O formato Ogg está presente em diversos projetos – Opus, FLAC, Vorbis, etc – de codificação e decodificação de conteúdo de multimídia que possuem o código aberto.

2.2.3 O formato RIFF

Desenvolvido para o uso em plataformas de multimídia, o *Resource Interchange File Format (RIFF)* é uma especificação geral sobre a qual diversos formatos de arquivo podem ser definidos. A principal vantagem dos arquivos no formato RIFF é a sua extensibilidade, que garante o seu funcionamento ainda que o arquivo seja modificado e ampliado com novas informações não definidas anteriormente. Por conta disso, diz-se que os arquivos baseados no formato RIFF são “à prova do futuro”(IBM; MICROSOFT, 1991).

Existem diversos formatos de arquivos de multimídia baseados no formato RIFF. Dentre eles o PAL (*RIFF Palette File Format*, para representação de paletas de cores no formato *RGB*), o RDIB (*RIFF Device Independent Bitmap Format*, para a representação de imagens), o RMID (*RIFF MIDI Format*, para representação de música), o RMMP (*RIFF Multimedia Movie File Format*, para representação de vídeo), e o WAVE (*Waveform Audio Format*), um dos formatos de áudio mais conhecidos e que servirá de base para este trabalho.

A estrutura básica de um arquivo RIFF é formada por um cabeçalho e por vários

blocos de dados conhecidos como *subchunks*. O cabeçalho e cada *subchunk* contém um identificador (denominado *chunkID*), um número inteiro que indica o tamanho da área de dados que se segue (*chunkSize*) e os dados propriamente ditos, que podem ser compostos por outros *subchunks* internos a ele ou outros dados.

A Figura 4 ilustra a explicação sobre a estrutura geral do formato RIFF:

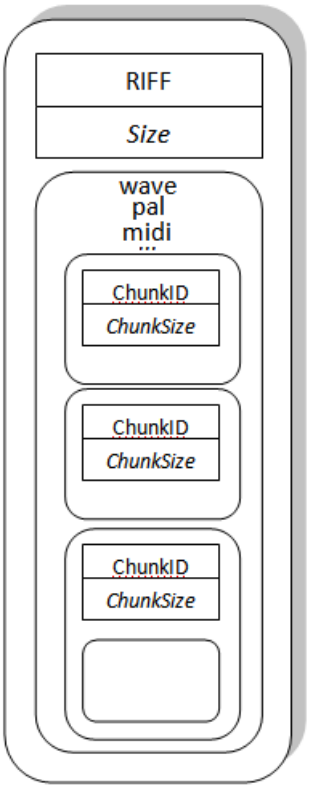

Figura 4 – Diagrama geral do formato RIFF.

O cabeçalho do formato RIFF, denominado *RIFF header*, é o *chunk* principal e tem a seguinte estrutura:

1. O *chunkID* é formado por quatro caracteres: `'R', 'I', 'F', 'F'`.

2. O *chunkSize* é um número de 32 *bits*, não sinalizado. Esse número não leva em consideração o tamanho do *chunkID* e nem seu próprio tamanho, mas somente o tamanho do bloco de dados. O seu valor é o tamanho do arquivo menos 8;

3. A parte referente aos dados é iniciada com quatro caracteres, que definem o formato do arquivo. Este formato determinará quais outros *subchunks* obrigatórios e opcionais que completarão este bloco de dados.

A forma canônica do formato WAVE inicia com o RIFF *header*, tendo como identificador de formato os quatro caracteres da palavra `WAVE`. Após a especificação do formato,

são obrigatórios dois *subchunks*, denominados ***fmt*** e ***data***, que devem manter esta ordem relativa entre eles (isto é, *fmt* deve preceder *data*).

O *subchunk fmt* descreve o formato em que o som foi representado. Ele é formado pelos seguintes itens:

Subchunk1ID: de tamanho de 4 *bytes*, contém os caracteres "`fmt `";

Subchunk1Size: é o tamanho da área de dados desse *subchunk*. Ocupa 4 *bytes*;

AudioFormat: caracteriza, em 2 *bytes*, a representação do áudio, e tem o valor 1 como padrão, indicando a modulação PCM. Outros valores indicam algum tipo de compressão;

NumChannels: especifica o número de canais: 1 para mono e 2 para stereo. Ocupa 2 *bytes*;

SampleRate: representa o número de amostras por segundo (11025, 22050, 44100). Ocupa 4 *bytes*;

ByteRate: representa o número de *bytes* por segundo. Ocupa 4 *bytes* e o seu valor é derivado da seguinte expressão:

$$ByteRate = SampleRate \times BlockAlign$$

BlockAlign: Representa o tamanho do bloco de dados. Ocupa 2 *bytes* e o seu valor é determinado pela fórmula a seguir:

$$BlockAlign = \frac{SampleRate \times NumChannels \times BitsPerSample}{8}$$

BitsPerSample: especifica o número de *bits* de dados usado para representar cada amostra de cada um dos canais. Ocupa 2 *bytes* e pode assumir valores iguais a 8 ou 16.

O *subchunk* "`data`" contém as amostras do som, de acordo com o formato indicado e o seu tamanho, em *bytes*. Ele é formado por três campos: *SubchunkID2, Subchunk2Size* e *Data*. O campo *Subchunk2ID* contém a palavra de 4 *bytes* "`data`", e *Subchunk2Size* contém o tamanho do bloco de dados, que pode ser determinado pela expressão a seguir:

$$Subchunk2Size = \frac{NumSamples \times NumChannel \times BitsPerSample}{8}$$

Por fim, *Data* contém as amostras de todos os canais arranjadas em um grande vetor de *bytes*.

2.3 Estruturas de Dados

As estruturas de dados são soluções organizadas de armazenamento de informações que, dependendo do contexto, permitem a utilização de dados de forma otimizada. Ao dispor todos esses dados de forma coesa dentro de uma forma definida, tem-se, portanto, uma **estrutura de dados**. Seguindo a definição de `struct` da linguagem C, uma estrutura de dados é um grupo de variáveis unidas por um nome específico (MIZRAHI, 1990). Estas estruturas são chamadas de **registros**. O estudo das estruturas de dados é a base para a construção de diversas áreas da ciência da computação e muitos conhecimentos relacionados às estruturas são fundamentais a todos os que trabalham no desenvolvimento de *software* (DROZDEK, 2002).

A seguir são apresentadas as estruturas de dados clássicas existentes na literatura.

2.3.1 Vetores

Os vetores são estruturas de dados bastante simples, implementadas nativamente por várias linguagens de programação. Eles são estruturas lineares e estáticas, uma vez que o seu tamanho é finito e deve ser definido em tempo de compilação. A principal vantagem do uso de vetores está no acesso imediato aos seus elementos, uma vez conhecido o índice do elemento a ser acessado.

Entretanto as operações de inserção e remoção de elementos, no início ou no meio do vetor, são extremamente ineficientes, pois implicam na transposição de elementos, para que o vetor mantenha a posição relativa dos elementos que já estavam no vetor antes da operação a ser realizada (GUIMARAES; LAGES, 1994).

2.3.2 Listas

As listas são estruturas lineares como os vetores, mas, diferente dessas, são dinâmicas e os seus elementos não estão necessariamente dispostos de forma sequêncial na memória.

Em geral, as listas são **simplesmente** encadeadas e **duplamente** encadeadas (ou ligadas). Nas listas simplesmente encadeadas (ou apenas listas encadeadas), cada elemento da lista (denominado **nó**) possui uma referência para próximo elemento da lista. Nas listas duplamente encadeadas, os nós possuem referência tanto para o sucessor quanto o antecessor (LAFORE, 2004).

Para compor uma lista, é necessário conhecer ao menos um elemento, a partir do qual os demais serão acessados através das referências contidas nos nós. Em geral, são escolhidos ou o primeiro elemento (denominado *head*), ou o último elemento (*tail*), ou ambos.

As listas possuem certas vantagens em relação aos vetores: a inserção e remoção, tanto no início quanto no final, tem ordem de complexidade $O(1)$. Também a inserção entre elementos intermediários dessa estrutura não acarreta no reposicionamento dos demais. Entretanto, há também desvantagens: o acesso aleatório a elementos diferentes do primeiro (*head*) e do último (*tail*) não é imediato como nos vetores, tendo ordem de complexidade $O(n)$ no caso médio.

Uma variação comum das listas encadeadas é a **lista circular**, na qual o último elemento tem uma referência direta para o primeiro elemento e vice-versa. Listas circulares podem ser implementada a partir de listas tanto simplesmente quanto duplamente encadeada.

Embora as listas simplesmente e duplamente encadeadas exijam a busca sequencial, é possível, com o uso das listas auto-organizadas, melhorar a eficiência dessa operação organizando dinamicamente a lista de uma certa maneira. Essa organização depende da configuração dos dados; assim o fluxo de dados exige a reorganização dos nós já contidos na lista. Existem diversas formas de organizar as listas. (DROZDEK, 2002) cita quatro exemplos:

1. **Método de mover-para-frente**. Depois de localizar o elemento desejado, coloque-o à frente da lista;
2. **Método da transposição**. Depois de localizar o elemento desejado, troque-o com o seu predecessor, a menos que ele já esteja na primeira posição da lista;
3. **Método da contagem**. Ordene a lista pelo número de vezes que os elementos estão sendo acessados;
4. **Método da ordenação**. Ordene a lista usando certos critérios comuns para a informação sob escrutínio.

2.3.3 Listas de salto

As **listas de salto** (*skip lists*) são variantes das listas encadeadas ordenadas, propostas por William Pugh, em (PUGH, 1990). Essa estrutura ajuda a minimizar um sério problema existente nas listas encadeadas: a localização de um certo elemento na lista exige a travessia sequêncial de todos os elementos (ASCENCIO; ARAUJO, 2010).

Os nós de uma lista encadeada deve ser visitado sequencialmente até que o elemento em questão seja encontrado. Nas *skip lists*, certos nós podem ser saltados, com o objetivo de principal de evitar esse processamento linear nas buscas.

Em uma lista de salto de n nós, para cada k e i tal que $1 \leq k \leq \lfloor \log_2(n) \rfloor$ e $1 \leq i \leq n/2^{k-1} - 1$, o nó na posição $i \times 2^{k-1}$ aponta para o nó na posição $(i+1) \times 2^{k-1}$.

Isso significa que a cada segundo nó aponta para o nó duas posições à frente, a cada quarto nó aponta para o nó quatro posições à frente e assim por diante. Isso é realizado com diferentes números de ponteiros em nós na lista: metade dos nós tem somente um ponteiro, um quarto dos nós tem dois ponteiros, um oitavo dos nós tem tem três ponteiros e assim sucessivamente. O número de ponteiros indica o nível de cada nó. Os nós com a maior quantidade de ponteiros definem o nível da *skip list* (DROZDEK, 2002).

Pugh (PUGH, 1990) mostra que se em uma lista for possível saltar de um nó para o segundo nó a sua frente, então são visitados não mais que o total de $(n/2)+1$ nós, onde n é a quantidade de elementos da lista. A Figura 5 abaixo mostra como isso acontece.

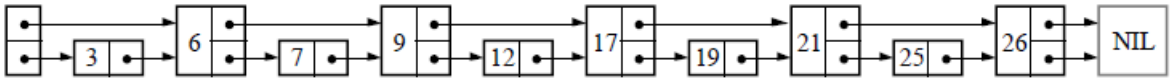

Figura 5 – *Skiplist* de nível 2 (PUGH, 1990).

Ele também mostra que, se a cada quarto dos nós da lista consegue apontar para os quatro nós posteriores, então não mais que $(n/4)+2$ nós são visitados, como mostra a Figura 6.

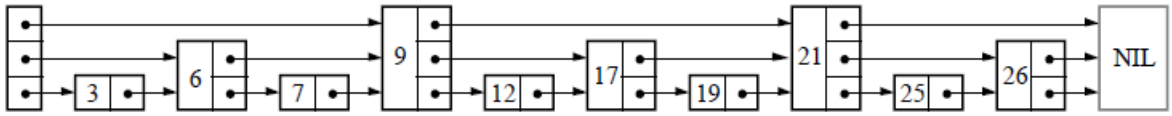

Figura 6 – *Skiplist* de nível 3 (PUGH, 1990).

Em suma, se o nó da posição 2^i possui ponteiros para até 2^i sucessores, se existirem, então o número de nós que devem ser visitados pode ser reduzido para apenas $\log_2(n)$. Portanto, as *skip lists* proporcionam pesquisas mais eficientes do que as listas encadeadas. Entretanto, as inserções e as remoções nessa mesma estrutura de dados possuem desempenho menos eficiente do que as outras listas (PUGH, 1990).

2.3.4 Filas e Pilhas

As filas são estruturas lineares que obedecem ao princípio FIFO (*first in, first out*), isto é, o primeiro elemento a entrar é o último a sair. Elas possuem estrutura semelhante a das listas encadeadas, mas a inserção e a remoção de elementos só ocorrem em extremos opostos, e nunca em outras posições, exceto se for em uma fila de prioridade (LAFORE, 2004).

As pilhas são estruturas lineares que obecedem ao princípio do LIFO (*last in, first out*), isto é, o último elemento a entrar é o primeiro a sair. Elas possuem estrutura semelhante às listas encadeadas, mas os elementos são inseridos e removidos apenas em um dos extremos (LAFORE, 2004).

2.3.5 Árvores

As **árvores** são definidas recursivamente da seguinte maneira (DROZDEK, 2002):

1. Uma estrutura vazia é uma árvore vazia;
2. Se $a_1, a_2, ..., a_n$ são árvores disjuntas, então a estrutura cuja raiz tem como filhas as raízes de $a_1, a_2, ..., a_n$ também é uma árvore;
3. Apenas as estruturas formadas pelas regras anteriores são árvores.

As **árvores binárias** são formadas por um conjunto finito de nós que pode estar vazio ou segmentado em três partes: a raiz da árvore, a subárvore binária da esquerda e a subárvore binária da direita. Todo nó que possui as subárvores da esquerda e da direita vazias é chamado de folha da árvore binária (TENENBAUM; LANGSAM; AUGENTEIN, 1995).

Seguindo a definição de árvores descrita no início desta seção, é possível que cada nó tenha m filhos, para $m > 2$. Uma árvore com essa característica é chamada de **árvore múltipla de ordem** m ou **árvore** m**-ária**. Numa árvore de ordem m (DROZDEK, 2002):

1. cada nó tem m filhos e $m - 1$ chaves;
2. as chaves em cada um dos nós estão em ordem ascendente;
3. as chaves nos primeiros i filhos são menores do que na chave i-ésima;
4. as chaves nos últimos $m - i$ filhos são maiores do que na chave i-ésima.

Uma **árvore-B** de ordem m é uma árvore de busca mútipla se possui as propriedades abaixo (DROZDEK, 2002):

1. A raiz tem pelo menos duas subárvores, a menos que ela seja uma folha;
2. Cada nó não-raiz e não-folha contém $k - 1$ chaves e k ponteiros para as subárvores, onde $\lfloor m/2 \rfloor \leq k \leq m$;
3. Cada nó folha contém $k - 1$ chaves, onde $\lfloor m/2 \rfloor \leq k \leq m$;
4. Todas as folhas estão no mesmo nível.

Com as condições acima, uma árvore-B sempre estará cheia pelo menos até a metade, com poucos níveis e sempre balanceada.

2.4 Padrões de Projeto

Padrões de projeto descrevem as soluções para problema frequentes e que, portanto, podem ser aplicadas em diversos contextos de desenvolvimento de *software*. Um padrão é um par problema/solução bem conhecido que pode servir para resolver as demandas de qualquer tipo de projeto. Ainda que o campo de desenvolvimento de *software* seja considerado jovem, os padrões de projeto têm seus nomes, descrições e formas de implementação bem estabelecidos na literatura (LARMAN, 2012).

2.4.1 Padrão MVC

O MVC (*Model*, *View*, *Controller*) é um padrão de arquitetura que sugere a separação das classes em três áreas independentes: Modelo, Visão, Controladora. O modelo é formado por classes que representam as entidades do mundo real ou do negócio em questão. A visão contém todas as classes de interface ou de apresentação do sistema, e a controladora é composta por classes que representam os casos de uso do sistema (SHAW; GARLAN, 1996). Ela fica situada entre a visualização e o modelo, pronta para receber as solicitações dos usuários (FREEMAN; FREEMAN, 2005). Utilizando a abordagem problema/solução, (LARMAN, 2012) fala sobre a Controladora com o seguinte questionamento: qual é o primeiro objeto, além da camada de visão, que recebe e coordena uma operação do sistema? Ele diz que a solução para isso é atribuir a obrigação a uma classe que representa o caso de uso a ser executado. A Figura 7 ilustra de maneira simples um exemplo de aplicação do padrão MVC.

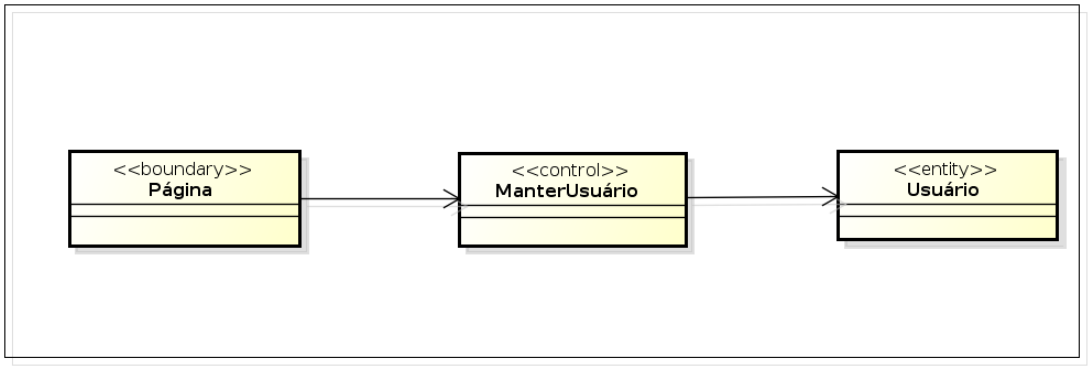

Figura 7 – Padrão MVC.

A classe `Página` representa a interface pela qual o usuário interage e realiza as operações no sistema, logo, ela faz parte da Visão. Após invocar uma operação para alterar, criar ou excluir um *usuário*, a classe `ManterUsuário` é chamada para realizar a operação, utilizando objetos da classe `Usuário`. Dessa maneira, conclui-se que a classe `ManterUsuário` faz parte da controladora e que a classe `Usuário` está inserida no modelo.

2.4.2 Fábrica Abstrata

O padrão Fábrica Abstrata (*Abstract Factory*) faz parte do conjunto de padrões de projeto conhecido como *Gang of Four* (*GoF*). Este padrão descreve o seguinte problema/contexto: como criar famílias de classes relacionadas que implementam uma interface comum? A solução é definir uma interface de fábrica e uma classe de fábrica concreta para cada família de itens a criar. Opcionalmente, pode ser definida uma classe abstrata, que implemente a interface de fábrica e forneça serviços comuns para as fábricas concretas (LARMAN, 2012).

A Figura 8 mostra um exemplo geral da aplicação do padrão *Abstract Factory*.

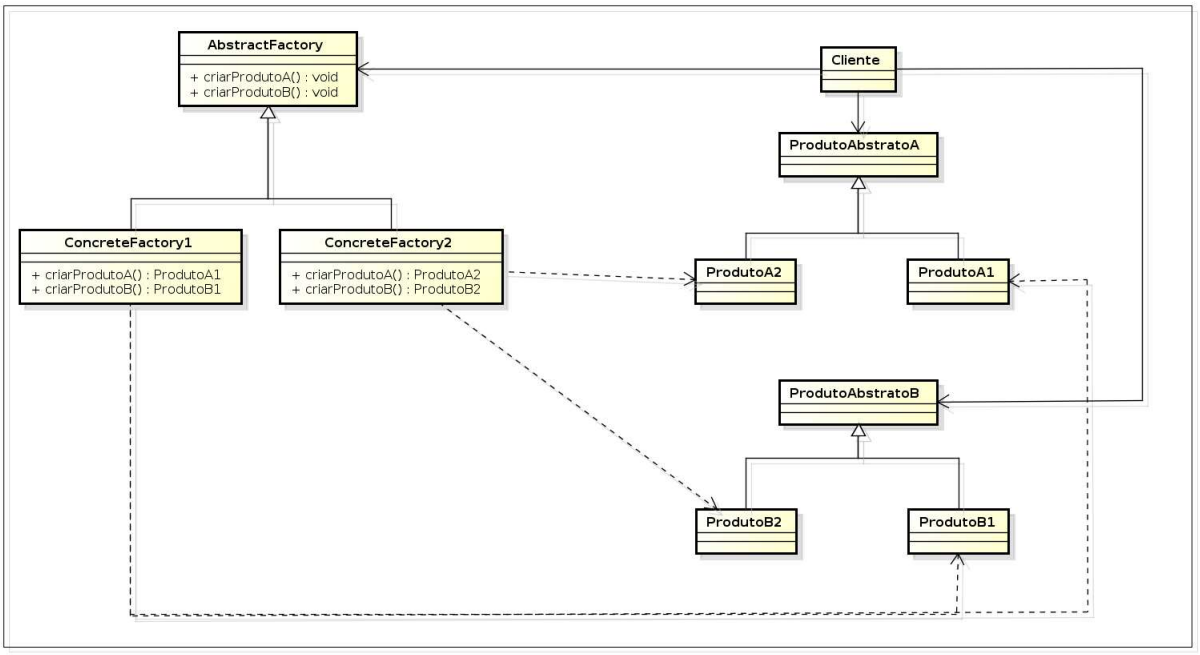

Figura 8 – Padrão Fábrica Abstrata. (DEVMEDIA, 2013)

A classe `AbstractFactory` é quem fornece os serviços comuns para as duas fábricas concretas `ConcreteFactory1` e `ConcreteFactory2`. Elas criam os produtos concretos de uma família de objetos relacionados. No exemplo acima, a fábrica `ConcreteFactory1` tem a responsabilidade de criar os produtos `ProdutoA1` e `ProdutoB1`; e a fábrica concreta `ConcreteFactory2` se encarrega de produzir `ProdutoA2` e `ProdutoB2`.

2.5 Reuso de *Software*

Reuso de *software* é entendido como a aplicação de práticas, conhecimentos, artefatos, processos, ou códigos previamente desenvolvidos em uma nova aplicação ou contexto (BIGGERSTAFF, 1989).

O reuso surgiu a partir da observação de que, independente dos contextos, os softwares costumam seguir tendências parecidas. Assim, é possível reutilizar uma solução de um problema já resolvido em outros projetos. (MEYER, 1988).

A utilização de bibliotecas é um grande exemplo para reuso na Engenharia de *Software*. As bibliotecas estáticas e dinâmicas são componentes de *software* que oferecem um conjunto de funcionalidades específicas. Entretanto, essas funcionalidades podem ser a solução de um problema inserido em dois ou mais projetos ao mesmo tempo. Dessa maneira, a criação de bibliotecas auxilia na prática de reuso, e contribui para a diminuição do esforço de desenvolvimento em um projeto.

3 Desenvolvimento

3.1 O Ambiente

O processo de implementação do Editor e do Tocador de *audiobooks* foi iniciado com a preparação do ambiente local de desenvolvimento, através da escolha de uma série de ferramentas. Foi selecionado o editor de texto Kate (*KDE Advanced Text Editor*) para a escrita dos códigos em *C++*. O compilador utilizado foi o *g++* versão 4.7.2 e o depurador escolhido foi o *gdb* versão 7.5 para Ubuntu. O QtCreator (DIGIA, 2013), versão 2.7.2, foi a ferramenta de desenvolvimento rápido[1] usada na prototipação e construção da interface gráfica do Tocador. Para o desenvolvimento do Editor e do Tocador, verificou-se a necessidade de saber a estrutura em hexadecimal dos arquivos *wave*: para isso, utilizou-se a ferramenta Hexdump. De acordo com o manual do Linux, o Hexdump é um filtro que exibe os arquivos, em um formato escolhido pelo usuário. Esse formato pode ser hexadecimal, octal, ASCII e decimal. E por último, o Dropbox foi utilizado para armazenar os códigos desenvolvidos e garantir a segurança. Quanto ao ambiente remoto, o GitHub foi a ferramenta de gerência de configuração de *software* selecionada para o controle de versão do código fonte. A utilização do Dropbox garante a segurança para os dados, porém não realiza o versionamento dos arquivos como o GitHub faz.

3.2 Documentação do Projeto

A documentação do projeto foi iniciada com a criação do Documento de Visão. Nele estão descritos os requisitos e as restrições do Editor e do Tocador em alto nível, para que fossem detalhados nos próximos documentos. Esse artefato também continha a identificação do problema e visão geral da solução. Para a especificação dos requisitos funcionais, foi escolhido o Documento de Requisitos do RUP (*Rational Unified Process*). Nele foram descritos os casos de uso do projeto e demais requisitos funcionais de forma mais detalhada, mostrando as entradas, saídas, pré-condições, pós-condições, regras de negócio, os atores e demais detalhes a respeito da iteração entre o usuário e o sistema. O próximo documento escrito foi a Especificação Suplementar. Ele capturou diversos requisitos que não são detalhados no documento de Especificação de requisitos. Ou seja, todos os requisitos não-funcionais do Tocador e do Editor, mostrando todos os aspectos como Usabilidade, Confiabilidade, Desempenho e Suportabilidade.

[1] *Rapid Application Development* (RAD)

3.3 Metodologia de Desenvolvimento

Mesmo utilizando artefatos do RUP e algumas práticas de métodos ágeis, durante as atividades de implementação, não foi seguido, de forma rigorosa, o processo RUP. De modo geral, o processo foi *ad hoc*, pois o conjunto de atividades desenvolvidas não foi separada em Concepção, Elaboração, Construção e Transição, conforme o processo especifica. Foi feita uma espécie de Concepção muito rápida e logo seguiu-se para a Construção até o fim do projeto. Alguns fatores foram fundamentais para isso acontecer. O principal fator foi o prazo curto para a realização das atividades. Assim, as modelagens e o levantamento de requisitos foram feitas de forma simples e direta, com o intuito de logo partir para as codificações. Outro ponto de bastante influência foi o fato de a equipe ser composta por apenas um membro, tornado os artefatos das metodologias sem sentido de uso, pois, os documentos dos processos ágeis ou do RUP, por exemplo, são direcionados para comunicação entre os membros que compõem a equipe. Com a equipe formada por uma pessoa, grande parte dos artefatos perdem o sentido de uso.

Mesmo sem utilizar o *RUP* de forma rigorosa durante o desenvolvimento do Editor e Tocador, o uso de práticas ágeis e de padrões de projeto esteve presente. Houve sessões semanais de programação em pares, prática do *Extreme Programming* (XP); refatoração de código, para melhorar a estrutura interna dos programas, aumentando, assim, a legibilidade. A aplicação de padrões de projeto também esteve presente na implementação dos produtos. Percebeu-se que a leitura dos blocos de dados (*chunks*) do *wave* poderia ser feita pela implementação do padrão Fábrica Abstrata, pois, mesmo contendo características diferentes, todos os blocos possuem uma estrutura geral semelhante. Além de proporcionar maior legibilidade ao código, o padrão Fábrica Abstrata facilitou as atividades de manutenção do *software*, por tornar flexível a criação e remoção dos *chunks*. O padrão MVC (*Model*, *View*, *Controler*) foi implementado no Tocador de *audiobook* para separá-lo em três componentes descritos no padrão. Essa separação contribui aumento da coesão e diminuição do acoplamento, além de contribuir para o reuso de código.

3.4 O Editor de *Audiobook*

Neste trabalho, o Editor é a parte responsável pela criação arquivos no formato *RAB* [2], a partir de um *WAVE*. Logo após carregar desse tipo, o usuário pode inserir todos os dados do *audiobook*, ou seja, os seus metadados, e também todas as marcações lógicas de acordo com a estrutura do índice do livro impresso. Após inserir todos os dados referentes aos metadados e às marcações de conteúdo, o Editor salva todas essas informações novas, criando um novo arquivo com a extensão *RAB*.

[2] *RIFF Audiobook*. Formato aberto baseado no *WAVE* do *RIFF* capaz de armazenar as marcações de conteúdo de um *audiobook*.

O protótipo do Tocador, implementado na primeira parte deste trabalho, foi de suma importância para validação dos conhecimentos teóricos sobre o formato do *RIFF*, especificamente do formato *WAVE*. Isso permitiu entender melhor as estruturas dos blocos de dados presentes nele, e assim, poder carregá-los em tempo de execução e também formular as estruturas dos novos blocos, que são o de metadados e o de marcações de conteúdo.

Quando se iniciou o desenvolvimento do Editor, percebeu-se que o processo de leitura dos blocos de formato WAVE estava muito repetitivo e extenso, tornando-o muito suscetível a propagações de erros. Então, foi feito o seguinte raciocínio: ainda que todos os blocos de dados possuam características próprias, todos eles possuem estrutura geral semelhante. Dessa maneira, a melhor forma de representar a criação dos blocos de dados facilitando o processo de leitura e escrita foi através do padrão de projeto conhecido como Fábrica Abstrata. Isso permitiu que a criação de blocos se tornasse um processo flexível e de fácil manutenção. A implementação desse padrão no código do editor eliminou o excesso de repetição, tornou o código reutilizável e melhorou, consideravelmente, a legibilidade do código.

Para executar os processos de leitura e escrita dos blocos do arquivo carregado, foram criadas, em cada classe que representa um *chunk*, funções de decodificação (*decode*) e de codificação (*encode*). A função *decode* é responsável por transformar a informação em hexadecimal proveniente de um bloco em algo legível para o usuário. Por outro lado, a função de *encode* faz o processo inverso: toda a informação do bloco é transformada em hexadecimal para, assim ser salva num arquivo final. Para realizar o processos de codificação e decodificação, foi utilizado o método conhecido como *Type-Lengh-Value* (tipo-tamanho-valor) (ITU, 2013). Ou seja, para realizar os dois processos, para cada campo dos blocos, já eram conhecidos, de antemão, o tipo do dado, o seu tamanho e o seu respectivo valor. Com essas três dados previamente conhecidos, os processos de codificação e decodificação se tornaram previsíveis e de fácil entendimento.

Concluídas as etapas anteriores, chegou a hora de validar os processos de leitura e escrita de blocos. Para isso, um arquivo *WAVE* foi carregado pelo programa, todos os blocos que nele existiam passaram pelo processo de *decode*, depois pelo processo de *encode*, e após isso, foram todos eles salvos em um novo arquivo com a mesma extensão. Quando comando `diff` do Linux foi utilizado para comparar o arquivo de entrada com o de saída, verificou-se que ambos eram exatamente iguais. Ou seja, os processos de leitura e escrita funcionaram corretamente.

O próximo passo foi verificar a possibilidade de salvar um novo bloco sem comprometer a integridade dos dados dos outros blocos. Então, no momento de escrever blocos já carregados em um novo arquivo, foi adicionado um bloco genérico. E o resultado foi que o *WAVE* gerado funcionou corretamente, sem qualquer dado corrompido. Concluiu-se,

portanto, que era possível inserir novos blocos junto com os que já existiam no arquivo original. Logo, o caminho estava aberto para a inserção dos novos dois blocos de dados do arquivo *RAB*: o bloco *META*, que contém todos os metadados e o bloco LGMK, responsável por armazenar as marcações de conteúdo *audiobook*.

3.4.1 O Bloco *META*

O *chunk META* é responsável por armazenar todos os dados referentes ao livro original. Este bloco é formado, em seu cabeçalho, por um campo formado por quatro caracteres que o identificam unicamente e um campo formado por um inteiro não sinalizado que indica o tamanho da sua parte de dados, onde estão armazenados os dados do *audiobook*. A Figura 27 mostra os detalhes deste bloco.

Figura 9 – O bloco *META*

No campo de identificação do bloco estão os quatro caracteres `'M', 'E', 'T', 'A'`, e, em seguida, o campo *Size* armazena a soma dos tamanhos dos próximos campos. Após o cabeçalho, todos os demais campos são formados por texto proveniente do usuário através do Editor, pela linha de comando. Quando o usuário encerra a inserção desses dados, ele é instruído a inserir os dados das marcações de conteúdo.

3.4.2 O Bloco *LGMK*

O *chunk LGMK* é responsável por guardar as marcações de conteúdo definidas pelo usuário do Editor. Cada marcação representa o tempo, em segundos, onde a marcação inicia. Além do tempo em segundos, o usuário deve inserir um nome para identificar a respectiva marcação. O editor permite que sejam criadas marcações em até dois níveis. O primeiro são inseridas as marcações de nível 1, depois as de nível 2, e as marcações de primeiro nível são de hierarquia superior as de segundo nível. A Figura 28 mostra a estrutura do *LGMK*.

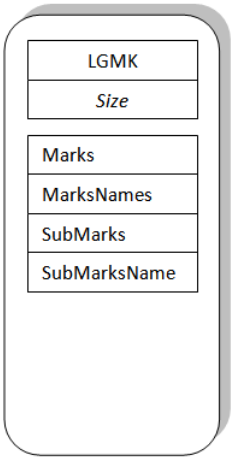

Figura 10 – O bloco *LGMK*

O bloco é fomado, em seu cabeçalho, pelo campo identificador, que contém os caracteres `'L', 'G', 'M', 'K'` e um campo formado por um inteiro não sinalizado que indica o tamanho da sua parte de dados. Na parte de dados, estão presentes as marcações de primeiro e segundo níveis, bem como os seus respectivos nomes. Por conta do prazo curto para o desenvolvimento, deixou-se de utilizar a estrutura de dados *skiplist* para as marcações de conteúdo. No seu lugar foi utilizada a estrutura *vector*, que já é nativamente implementada no C++. Mesmo implementando numa estrutura de dados mais simples, os resultados não foram comprometidos, e o arquivo *RAB*, com os metadados persistidos e as marcações de conteúdo definidas, foi criado com sucesso.

Durante o desenvolvimento, verificou-se que o Tocador utilizava um conjunto de classes semelhantes ao conjunto de classes do Editor. Toda vez que era realizada uma alteração em qualquer classe comum de um lado, era necessário replicar essa alteração no outro módulo, sob o risco de propagação de erros no código. Observou-se que isso tomava tempo de desenvolvimento. Para resolver esse problema, foi criada uma biblioteca estática, contendo todas as classes comuns entre eles. Esta biblioteca tornou-se o módulo Core (núcleo) destes programas.

3.5 O Tocador de *Audiobook*

A codificação do tocador foi iniciada com a criação da sua interface através do QtCreator. A Figura 11 mostra a interface do tocador desenhada nessa ferramenta.

O tocador é responsável por mostrar todos os metadados gravados no bloco META. Os rótulos (*labels*) ***Title***, ***Author***, ***Language***, ***Publisher***, ***Address***, ***Pages*** e ***Year*** são estáticos, ou seja, eles permanecem inalterados durante a execução do programa. Por outro lado, os *labels* TitleLabel, AuthorLabel, LanguageLabel, PublisherLabel, AddressLabel, PagesLabel e YearLabel são substituídos de acordo com os metadados que estão gravados no

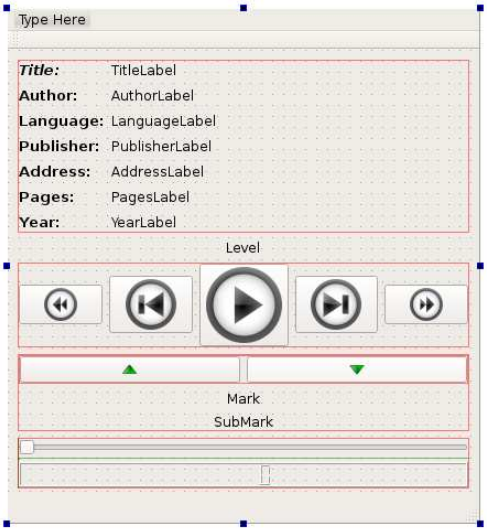

Figura 11 – Interface do Tocador no QtCreator

bloco *META* do arquivo carregado. Os outros três rótulos apresentados, Level, Mark, SubMark estão relacionados ao bloco *LGMK*, onde estão as marcações de conteúdo. Quando um arquivo é lido no tocador, Level inicia no nível 1, que é o nível mais alto de navegação nas marcações. Mark e SubMark indicam, respectivamente, o nome da primeira marcação de nível 1 e o nome da primeira marcação de nível 2, e ambos são substituídos pelos valores registrados no bloco de marcações de conteúdo.

As funcionalidades dos botões apresentados na interface do tocador não diferem muito das funcionalidades dos tocadores comuns. A diferença é que os botões de *forward* e *rewind* são usadas para navegar nas marcações do respectivo nível e os botões situados logo abaixo são usados para mudar o nível de navegação nas marcações. O usuário poderá alternar entre o nível 1 e o nível 2.

3.6 A Biblioteca Estática

Todo o código que era comum entre o Tocador e o Editor foi reunido e separado em um único local para formar a biblioteca estática que ambos pudessem reutilizar. Dessa maneira, as mudanças nesse componente são centralizadas, evitando propagação de erros por conta de operações de copia-cola, como ocorreu durante uma parte do projeto. A biblioteca estática recebeu o nome de Core e está disponível no endereço http://goo.gl/VKLx2u do repositório.

A Figura 12 mostra o diagrama formado pelos componentes do projeto: o Tocador, o Editor e o Core. As setas apontando para o Core indicam que o Tocador e o Editor compartilham as funcionalidades da biblioteca.

Detalhando o Core, a Figura 13 mostra o diagrama de classes desse componente.

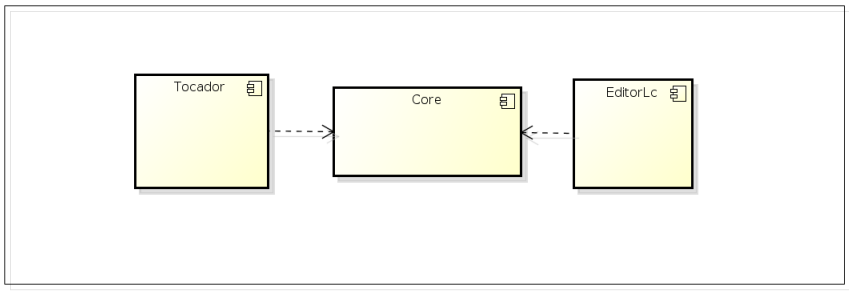

Figura 12 – Diagrama de Componentes

Observa-se que este diagrama implementa uma variação mais simples do padrão Fábrica Abstrata apresentado na seção 2.4.2. A classe `ChunkFactory` é a fábrica responsável por criar os produtos concretos, que são todas as subclasses de `Chunk`.

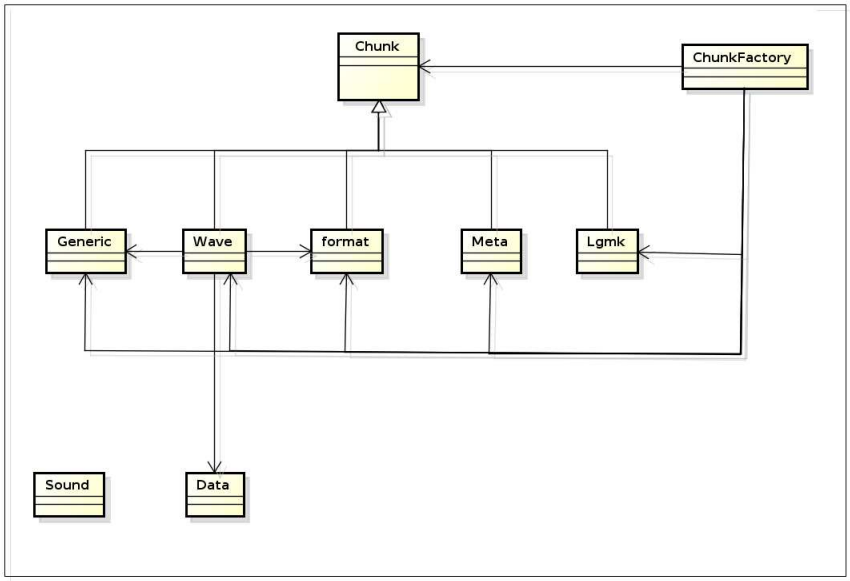

Figura 13 – Diagrama de Classes do Core

A Figura 14 apresenta o diagrama de classes do Tocador, separando os componentes entre Modelo, Visão e Controlador tal qual é especificado no padrão MVC, presente na Seção 2.4.1. O pacote Core representa a biblioteca estática.

Por fim, a Figura 15 mostra o diagrama de classes do Editor. A classe `Main` utiliza os serviços das classes da biblioteca Core e implementa as funções de iteração com usuário através de linha de comando.

3.7 A pesquisa de opinião

Para avaliar a aceitação do novo formato, foi desenvolvida uma pesquisa de opinião para potenciais usuários do produto. Ela foi estruturada com base na estrutura de um

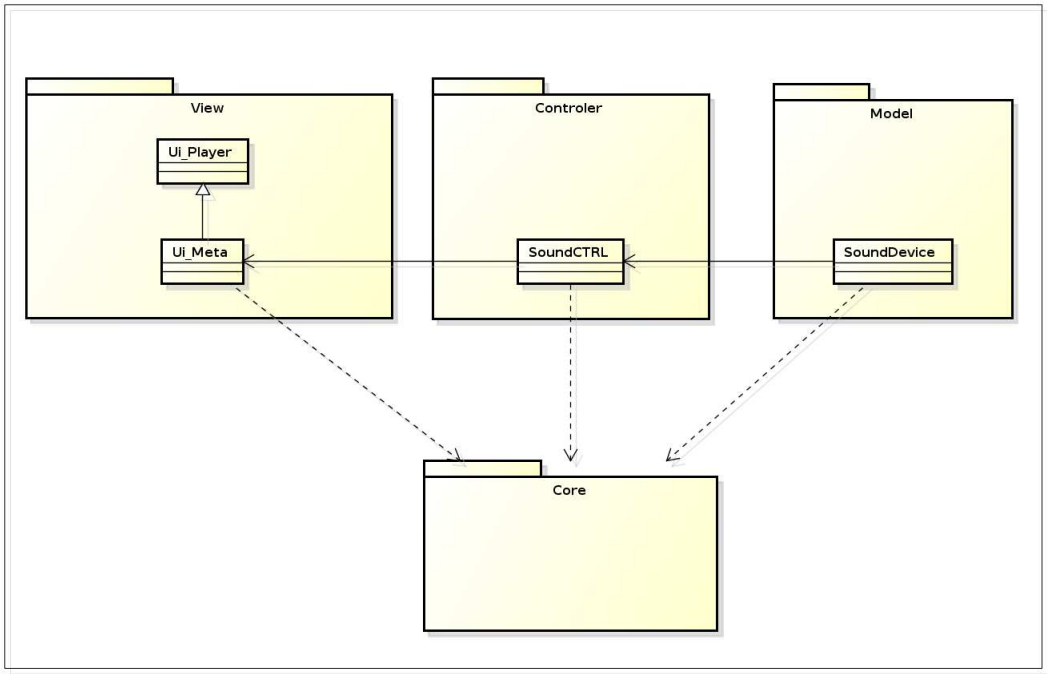

Figura 14 – Diagrama de Classes do Tocador

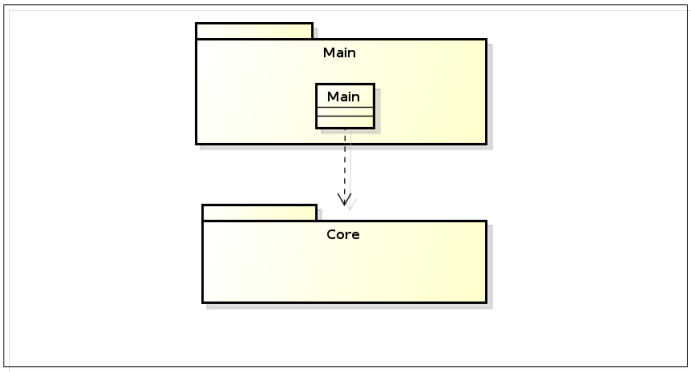

Figura 15 – Diagrama de Classes do Editor

Projeto e Relatório de Pesquisa proposto por (LAKATOS; MARCONI, 2012), descrito no Anexo F. O questionário da pesquisa de opinião foi desenvolvido no aplicativo Google Forms [3], que é um programa específico para desenvolver questionários com qualquer tipo de questão. O aplicativo organiza as informações e exporta os dados para uma planilha eletrônica, possibilitando a construção de gráficos. Além do questionário em si, foi disponibilizado uma pasta pública no Dropbox contendo o executável do Tocador e os arquivos no formato *WAVE* e *RAB* para que o voluntário pudesse testar o programa e dar o seu *feedback*.

[3] http://goo.gl/9FBa9X

4 Resultados

4.1 O Ambiente e o Editor de *Audiobook*

As ferramentas selecionadas para o ambiente local e remoto foram instaladas corretamente sem qualquer problema. Os repositórios do Tocador e do Editor estão com os códigos-fontes devidamente atualizados e com as versões estabililizadas. Os códigos do Editor estão disponíveis no Anexo I e os códigos do Tocador podem ser visualizados no Anexo H.

O Editor de *audiobooks* gera, com sucesso, arquivos no formato *RAB* com todos os metadados e as marcações de conteúdo salvas. Mesmo com essa nova extensão, os arquivos gerados pelo Editor ainda podem ser executados em outros *players* com suporte a arquivos *.wav*, como o Amarok, ou Windows Media Player.

Para simular um *audiobook*, foi utilizado o Hino Nacional Brasileiro. Por ser bastante conhecido por todos, é o exemplo perfeito para a demonstração do uso das marcações de conteúdo. Isso se deve ao fato de que o símbolo oficial do Brasil é dividido em duas grandes partes e cada parte é formada por estrofes. Dessa maneira, é possível localizar as marcações de conteúdo desse tema e usá-lo como um *audiobook* para este trabalho. A primeira e a segunda parte desse hino são as marcações de nível 1, e suas estrofes são as marcações de nível 2. O arquivo do Hino Nacional Brasileiro utilizado neste projeto foi obtido do Portal do Planalto no formato MP3 e convertido para o formato *WAVE* com o programa Audacity. Após a conversão, o arquivo gerado foi carregado pelo Editor e foram inseridos os metadados conforme apresentados na Tabela 1.

Tabela 1 – Metadados inseridos no Editor.

Nome	**Valor**
Título	Hino Nacional Brasileiro
Autor	Joaquim Osório & Francisco Manuel
Idioma	Português
Editora	
Local de Criação	Brasil
Número de Páginas	0
Ano de Criação	1822

O campo *Editora* foi deixado em branco, porque o arquivo gerado não se trata de *audiobook* real, logo, não possui editora.

Em relação às marcações de conteúdo, as Tabelas 2 e 3 mostram as informações enviadas para o sistema para definir as marcações.

Tabela 2 – Marcações de Nível 1.

Nome da Marcação	**Tempo(s)**
Parte 1	0
Parte 2	103

Tabela 3 – Marcações de Nível 2

Nome da Marcação	**Tempo(s)**
Introdução 1	0
Primeira Estrofe	28
Segunda Estrofe	44
Terceira Estrofe	60
Quarta Estrofe	64
Quinta Estrofe	79
Sexta Estrofe	91
Introdução 2	103
Sétima Estrofe	132
Oitava Estrofe	148
Nona Estrofe	164
Décima Estrofe	168
Décima Primeira Estrofe	183
Décima Segunda Estrofe	195
Décima Terceira Estrofe	201
Final	206

O resultado da criação do novo arquivo, de nome `HinoNacionalBrasileiro.rab`, pode verificado com uso do Hexdump através do comando no terminal do Linux:

```
$ tail -f HinoNacionalBrasileiro.rab | hexdump -C
```

conforme é mostrado na Figura 16.

4.2 O Tocador de *Audiobook*

O Tocador de *audiobook* carrega com sucesso os arquivos no formato *RAB* e *WAVE*, mesmo que arquivos neste formato não possua os blocos de metadados e das marcações lógicas. Um arquivo sem metadados e sem marcações é reproduzido como se fosse um arquivo de áudio comum. Entretanto, não é possível fazer uso das marcações e todos os metadados são preenchidos com a palavra `UNKNOWN`, indicando que a não existência do bloco *META*. A Figura 17 mostra o que foi explicado, ao carregar o arquivo `HinoNacionalBrasileiro.wav` no Tocador.

O arquivo `HinoNacionalBrasileiro.rab` foi executado no Tocador e os seguintes resultados foram obtidos:

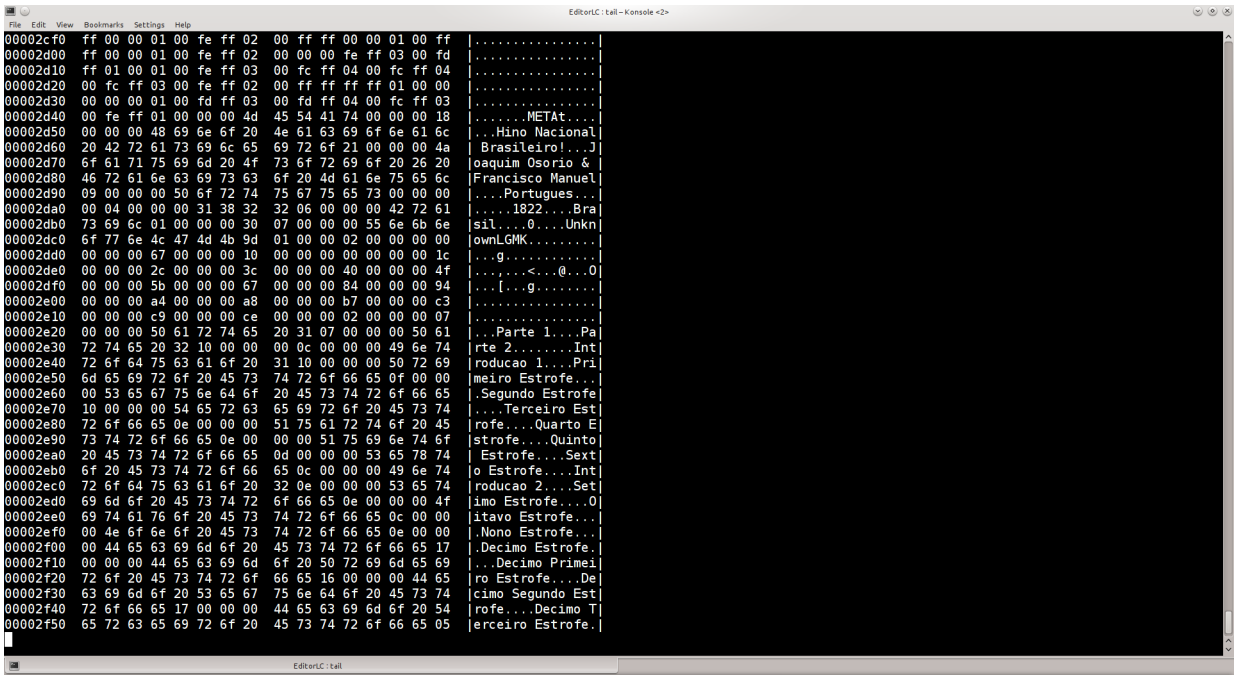

Figura 16 – Utilização do Hexdump no arquivo *RAB*.

Figura 17 – Exemplo de um *WAVE* carregado no Tocador

- Ao ser executado do início até final dos arquivos, não houve problemas de interrupção inesperada e nem travamentos do programa;
- Durante as execuções com arquivos *RAB*, os nomes das marcações foram alterados na ordem e no tempo certos;
- Todos os metadados dos arquivos *RAB* foram carregados corretamente, mas apresentaram problemas quanto aos caracteres acentuados;
- As navegações através dos botões de *fast forward*, de *fast rewind* e da barra de navegação foram executadas com sucesso;

- As navegações por entre as marcações de conteúdo usando os botões de *forward* e de *rewind* também funcionaram conforme o esperado, ajustando os *labels* das marcações para os nomes corretos e direcionando devidamente o áudio para o tempo registrado no Editor.

A Figura 18 apresenta o Tocador de *audiobooks* em execução, com o arquivo `HinoNacionalBrasileiro.rab` carregado:

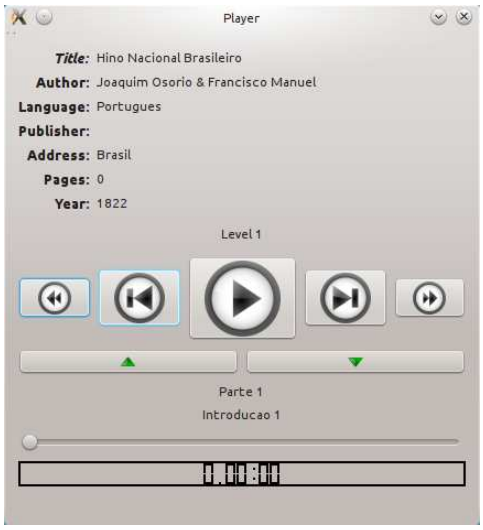

Figura 18 – Exemplo de *audiobook* carregado no Tocador

4.3 A Pesquisa de Opinião

O questionário desenvolvido no Google Forms está disponibilizado através do endereço http://goo.gl/fpDk9G e presente no anexo D. No cabeçalho do questionário estão descritas todas as instruções para que o voluntário tenha plenas condições de responder às oito perguntas, das quais, seis são objetivas e duas subjetivas. O questionário esteve disponível para os voluntários entre os dias 17 de Novembro e 24 de Novembro de 2013, e obteve 12 respostas.

4.3.1 Apresentação e análise dos dados

O objetivo desta subseção é apresentar e analisar os dados coletados pelo questionário de opinião. Deseja-se verificar se a hipótese apresentada na seção D.3.2 foi ou não confirmada.

O questionário é iniciado com a seguinte pergunta: *Quanto à localização e apresentação dos metadados (Título, Autor, Editora, Idioma, ...) do audiobook carregado no Tocador. O que você achou?* Na primeira pergunta, foram obtidos os resultados apresentados na Figura 19.

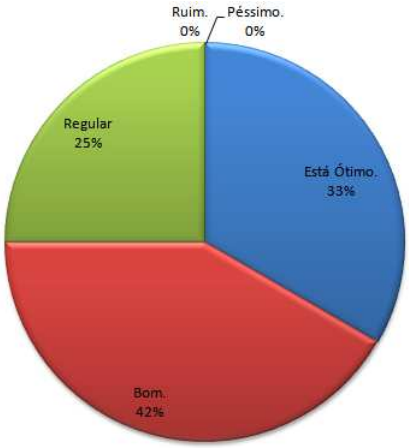

Figura 19 – Resultados da primeira questão

Observa-se que a avaliação da apresentação e localização dos metadados teve resultado positivo, pois dois terços dos voluntários responderam *Está Ótimo* ou *Bom* e nenhum deles respondeu *Ruim* ou *Péssimo*. Um quarto dos participantes foram neutros quanto ao aspecto visual dos metadados apresentados no Tocador.

Na segunda pergunta, o voluntário foi indagado da seguinte forma: *Quanto à localização e apresentação dos botões do Tocador de Audiobooks. O que você achou?* Os resultados desta questão podem ser vistos na Figura 20.

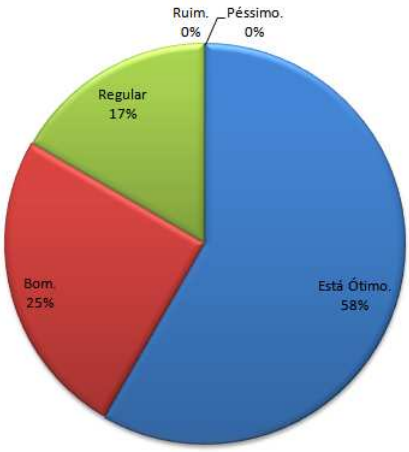

Figura 20 – Resultados da segunda questão

Assim como na primeira questão, o resultado também foi positivo, pois 83% dos participantes responderam *Está Ótimo* ou *Bom*. Para *Ruim* ou *Péssimo*, os resultados foram nulos e 17% dos participantes consideraram indiferente ou comum a posição dos botões do Tocador.

O terceiro questionamento foi feito da seguinte maneira: *Quanto à identificação das funcionalidades do Tocador de Audiobooks. O que você achou?* E os resultados obtidos são apresentados na Figura 21.

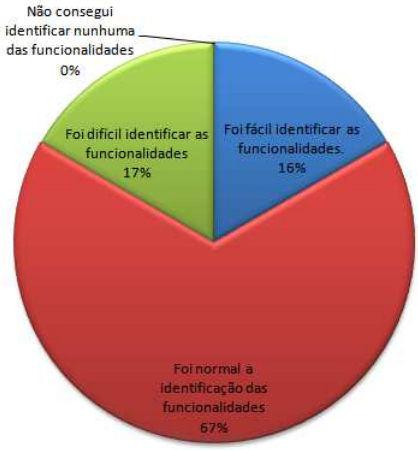

Figura 21 – Resultados da terceira questão

O objetivo da terceira pergunta era verificar se os voluntários conseguiriam perceber o diferencial do Tocador de *audiobooks* para os tocadores comuns. Segundo os resultados, 16% deles identificaram facilmente as funcionalidades, empatando, tecnicamente, com os 17% que acharam difícil a identificação. Entretanto, a indefinição do resultado é eliminada quando é apontado que pouco mais de dois terços dos participantes consideraram normal a identificação de funcionalidades; e mais, nenhum participante deixou de identificar as funções do Tocador. Considerando que não foi passado nenhum tipo de manual para os voluntários antes de testarem o Tocador, conclui-se que ele possui um ótimo nível de usabilidade, pois 83% dos participantes consideraram *fácil* ou *normal* identificação das funções do programa.

Na quarta pergunta, o voluntário avaliou a interface do Tocador: *Que nota geral você atribuiria para os aspectos da interface do Tocador de Audiobooks?* A Figura 22 mostra resultado da pergunta quatro.

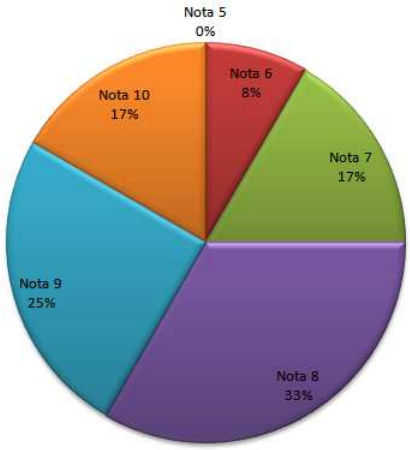

Figura 22 – Resultados da quarta questão

Com os resultados da quarta pergunta, observa-se que o aspecto da interface alcan-

çou bons resultados, concentrando 50% das avaliações no setor intermediário, com uma tendência para as notas mais altas. Diante disso, considera-se que os aspectos visuais da interface ajudou o Tocador a ganhar mais um ponto positivo.

A quinta questão quis saber a opinião sobre as marcações lógicas: *Qual é a sua opinião sobre funcionalidade de navegação no arquivo através das marcações de conteúdo oferecida pelo Tocador de Audiobooks?* Os resultados da pergunta cinco são mostrados na Figura 23.

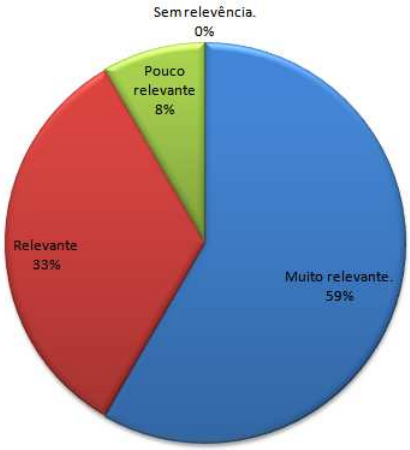

Figura 23 – Resultados da quinta questão

A quinta questão é a peça principal para essa pesquisa, pois, ela visa verificar a importância da navegação pelas marcações lógicas nos arquivos *RAB*. De acordos os resultados dessa pergunta, 92% dos colaboradores disseram que a navegação pelas marcações lógicas é *Muito Relevante* ou *Relevante*, contra, apenas, 8% dos que acharam *Pouco Relevante*. O resultado dessa questão confirma a hipótese, apresentada na Seção D.3.2, do projeto de pesquisa.

Na sexta pergunta, desejou-se saber a opinião geral do voluntário sobre o tocador: *Qual nota final você atribuiria para o Tocador, levando em conta todos os recursos oferecidos por ele?* Foram obtidos os resultados apresentados na Figura 24.

A pergunta, *a priori*, pode parecer genérica, mas ela foi escrita dessa maneira justamente para levar o voluntário a refletir, não somente pelo aspecto funcional, mas também, sob a perspectiva não-funcional do programa. E diante da questão seis, o tocador foi qualificado positivamente. Além de não receber nenhum voto na nota mínima (*nota 5*), 67% dos participantes atribuíram *nota 10* ou *nota 9* – as notas mais altas – contra 33% das demais notas.

Os voluntários da pesquisa também tiveram a oportunidade de expressar suas opiniões através de perguntas abertas. A sétima pergunta do questionário quis saber *Quais seriam os recursos que você adicionaria ao tocador?* Dentre as respostas estavam uma lista de reprodução, sugerindo que o Tocador carregue mais de um arquivo por

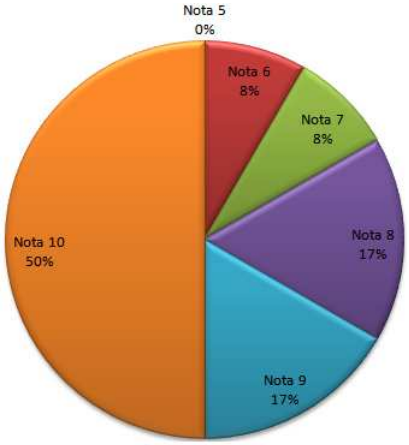

Figura 24 – Resultados da sexta questão

vez. Outros sugeriram dicas ou ajuda para os botões, e uma das respostas sugeriu que o Tocador pudesse carregar a imagem do *audiobook*.

A oitava e última questão pediu para que os participantes citassem críticas ou sugestões ao Tocador. Alguns colaboradores deram sugestões de ampliação para outros sistemas operacionais, como Windows, e também para outras plataformas baseadas no Unix. Um aluno fez um comentário mais completo, sugerindo que a apresentação dos metadados do *audiobook* pudesse ser opcional, que houvesse suporte a outros idiomas e que os botões de *play/pause* e os outros de navegação deveriam ser posicionados na parte inferior do Tocador.

Vale destacar, por fim, que as análises anteriores não são conclusivas, mas sim indicativos de tendência, dado o pequeno número de participantes (apenas 12) desta pesquisa de opinião. A título de trabalho futuro, seria interessante ampliar esta pesquisa com outros usuários, utilizando as redes sociais e a internet para validar ou não os dados aqui apresentados.

5 Conclusão

Este trabalho abordou a especificação de um formato aberto para *audiobooks* que tem suporte a marcadores de conteúdo. Muitos arquivos *RAB* foram criados através do Editor, para possibilitar a validação desse novo formato com o uso do Tocador e da pesquisa de opinião. Este projeto envolveu também a criação de artefatos de projeto de *software*, e eles orientaram o desenvolvimento dos requisitos dos produtos.

Documentos do RUP, práticas de métodos ágeis e padrões de projeto foram utilizados, mas a falta de organização para o uso do RUP contribuiu fortemente para o excesso de retrabalho e por erros encontrados tardiamente durante a codificação. Esses problemas consumiram muito tempo de trabalho, impedindo que o projeto pudesse avançar mais. A maneira ideal de organização do RUP para esse projeto seria o PU ágil - Processo Unificado ágil - recomendado por (LARMAN, 2012), por conta da documentação enxuta, da mesma forma que aconteceu nesse projeto.

Por ser um formato livre e *open source* para todos os interessados, o *RAB* contribuiu para o campo de conhecimento dos formatos de arquivo com sucesso, pois arquivos de documentação aberta com as suas características eram inexistentes até então. A pesquisa de opinião mostrou que os produtos desenvolvidos neste projeto têm potencial de sucesso no mercado entre o público jovem. Porém os resultados não são conclusivos, é necessário que o questionário seja aplicado a um número maior de participantes com faixa etária mais diversificada, para que a obtenção de melhores resultados.

O desenvolvimento desse projeto abraçou um conjunto de áreas de conhecimento muito extenso, de forma que trabalhos sobre este novo formato não se encerra por aqui. O *feedback* dos voluntários nas perguntas abertas do questionário confirma esse fato, mostrando que existe a demanda por suportabilidade, usabilidade, internacionalização, além de outros elementos que possam melhorar a interação entre o Tocador e o usuário.

Tudo que foi desenvolvido até aqui serviu para consolidar e aprofundar os conhecimentos em Engenharia de *Software* absorvidos ao longo do curso, e ajudou a aprender conceitos de outras áreas, contribuindo, assim, para uma melhor formação acadêmica por meio da interdisciplinaridade.

Referências

APPLE. Audible and Apple Announce Availability of Exclusive Content Through Apple iTunes. 2003. Acessado em 20 de Abril de 2013. Disponível em: <http://goo.gl/6asj44>. Citado na página 22.

ASCENCIO, A. F. G.; ARAUJO, G. S. de. Estruturas de Dados: algoritmos, análise de complexidade e implementação em Java e C/C++. São Paulo: Person, 2010. Citado na página 33.

AUDIBLE. History. 2013. Acessado em 14 de junho de 2013. Disponível em: <http://about.audible.com/history/>. Citado na página 21.

BARBOSA, A. M. Edição Digital de Som: Uma abordagem aos fundamentos da escultura sonora orientada para criadores. [S.l.]: Universidade Católica Portuguesa Escola das Artes Som e Imagem, 1999. Citado 4 vezes nas páginas 13, 26, 27 e 28.

BIGGERSTAFF, T. J. Design recovery for maintenance and reuse. Computer, IEEE, v. 22, 1989. Citado na página 37.

DAVISON, A. B. Mp3 players with bookmark capability. 2007. Disponível em: <http://www.andrewdavidson.com/articles/mp3-players-with-bookmarks/>. Citado na página 22.

DEVMEDIA. Padrões de projeto em .NET: Abstract Factory. 2013. Acessado em 23 Novembro de 2013. Disponível em: <http://www.devmedia.com.br/padroes-de-projeto-em-net-abstract-factory/3646>. Citado 2 vezes nas páginas 13 e 37.

DIGIA. Qt Project. 2013. Acessado em 03 de Novembro de 2013. Disponível em: <https://qt-project.org>. Citado na página 39.

DROZDEK, A. Estruturas de Dados e Algoritmos. São Paulo: Cengage Learning, 2002. Citado 4 vezes nas páginas 32, 33, 34 e 35.

FARIAS, S. C. O audiolivro e sua contribuição no processo de disseminação de informações e na inclusão social. Revista Digital de Biblioteconomia e Ciência da Informação, v. 10, n. 1, 2012. Disponível em: <http://www.sbu.unicamp.br/seer/ojs/index.php/rbci/article/view/529>. Citado na página 21.

FREEMAN, E.; FREEMAN, E. Head First Design Patterns. [S.l.]: Sebastopol: O'Reilly, 2005. Citado na página 36.

GUIMARAES, A. de M.; LAGES, N. A. de C. Algoritmos e Estruturas de Dados. Rio de Janeiro: GEN - Grupo Editorial Nacional, 1994. Citado na página 32.

HACKER, S. MP3: The Definitive Guide. [S.l.]: O'Reilly, 2000. Citado 3 vezes nas páginas 22, 28 e 29.

HANSEN, H. C. Fundamentals of acoustics. Occupational Exposure to Noise: Evaluation, Prevention and Control. World Health Organization, Geneva, 2001. Citado na página 25.

IBM; MICROSOFT. Multimidia Programming Interface and Data Specifications 1.0. [S.l.], 1991. Citado na página 29.

ITU. Introduction to ASN.1. 2013. Acessado em 23 Novembro de 2013. Disponível em: <http://www.itu.int/ITU-T/asn1/introduction/index.htm>. Citado na página 41.

JESUS, P. S. Letras e Vozes: O Livro Falado e a preservação da subjetividade. Salvador: [s.n.], 2008. Citado na página 21.

KEFAUVER, A.; PATSCHKE, D. Fundamentals of digital audio. [S.l.]: AR Editions, Inc., 2007. Citado 4 vezes nas páginas 13, 25, 26 e 27.

LAFORE, R. Estruturas de Dados e Algoritmos em Java. Rio de Janeiro: Editora Ciência Moderna, 2004. Citado 2 vezes nas páginas 32 e 34.

LAKATOS, E. M.; MARCONI, M. A. Metodologia do trabalho científico: procedimentos básicos, pesquisa bibliográfica, projeto e relatório, publicações e trabalhos científicos. 7. ed. São Paulo: Atlas S.A, 2012. Citado 3 vezes nas páginas 46, 81 e 83.

LARMAN, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3/e. [S.l.]: Pearson Education India, 2012. Citado 3 vezes nas páginas 36, 37 e 55.

LEWIS, D. R. et al. Comitê multiprofissional em saúde auditiva: Comusa. Braz J Otorhinolaryngol, SciELO Brasil, v. 76, n. 1, 2010. Citado na página 26.

MEYER, B. Object-oriented software construction. [S.l.]: Prentice hall New York, 1988. Citado na página 38.

MIZRAHI, V. V. Treinamento em Linguagem C: Curso Completo. São Paulo: McGraw-Hill, 1990. Citado na página 32.

PALLETA, F. A. C.; WATANABE, E. T. Y.; PENILHA, D. F. Audiolivro: inovações tecnológicas, tendências e divulgação. XV Seminário Nacional de Bibliotecas Universitárias, São Paulo, 2008. Citado na página 21.

PAN, D. Y. Digital audio compression. Digital Technical Journal, v. 5, n. 2, 1993. Disponível em: <http://www.ece.fcu.edu.au/subjects/ee3700/resources/Digita_Audio_Compression.pdf>. Citado na página 28.

PUGH, W. Skip lists: A probabilistic alternative to balanced trees. 1990. Citado 3 vezes nas páginas 13, 33 e 34.

SHAW, M.; GARLAN, D. Software architecture: perspectives on an emerging discipline. Prentice Hall, 1996. Citado na página 36.

TENENBAUM, A. A.; LANGSAM, Y.; AUGENTEIN, M. J. Estruturas de Dados Usando C. São Paulo: MAKRON Books, 1995. Citado na página 35.

XIPH. 2013. Acessado em 28 de Junho de 2013. Disponível em: <http://xiph.org>. Citado na página 29.

Anexos

ANEXO A – Documento de Visão

A.1 Introcução

A.1.1 Objetivo do Documento

Este documento visa descrever, em alto nível, todo o projeto que envolve a especificação de um formato de arquivo aberto para *audiobooks* e os demais produtos relacionados a ele.

A.2 Descricao Geral do Projeto

A inexistência de um formato livre e *open source* para *audiobooks* com suporte a marcação de conteúdo é o grande fator motivador deste projeto. A Audible, uma companhia da Amazon, e a Apple são as grandes líderes do mercado de *audiobooks*. Elas comercializam os produtos em seus respectivos formatos que possuem suporte a marcadores de conteúdo, entretanto eles são protegidos pelas leis propriedade intelectual, tornando impossível o acesso às suas documentações, especificações, códigos fontes, etc. Disponibilizar para a comunidade acadêmica tudo o que for desenvolvido neste trabalho é, com certeza, uma grande contribuição para todos os envolvidos e interessados no ramo de *audiobooks* e pesquisadores em geral.

A metodologia principal de desenvolvimento é o RUP (*Rational Unified Process*), de modo que este documento é o artefato que dá início ao processo de desenvolvimento. Serão utilizados os documentos de Especificação de Caso de Uso e Especificação Suplementar para desenvolver, respectivamente, os requisitos funcionais e os não-funcionais dos produtos.

A.3 Visão Geral dos Produtos

Este projeto, envolve a construção de três produtos: A especificação do formato *RAB* (*RIFF Audiobook*), o Tocador (*player*) de *audiobooks* e o Editor de arquivo.

O *RAB* é um formato de áudio baseado no *RIFF*, que desenvolvido pela Microsoft e a IBM. O Editor de arquivo é produto responsável por criar os arquivos *RAB*. Um arquivo do tipo *WAVE* é inserido como entrada e, após a criação das marcações de conteúdo e

dos metadados do audiobook, um novo arquivo de extensão *.rab* é gerado. O Player de *audiobook* fará uso das marcações de conteúdo presentes no *RAB*.

A.4 Requisitos do Produto

Nesta seção, estão descritos os requisitos iniciais do Tocador e do Editor. O detalhamento de cada um dos requisitos está presente no documento de Especificação de Requisitos e Especificação Suplementar.

1. Requisitos do Editor
    - O usuário deve carregar somente arquivos no formato *WAVE* ou *RAB*.
    - O Editor deve gerar apenas arquivos no formato *RAB*, após a criação dos novos blocos (*chunks*) que definem a estrutura desse formato.
    - O usuário poderá inserir dados apenas na parte referente aos metadados e às marcações de conteúdo, de forma que os demais blocos provenientes do arquivo original não poderão sofrer qualquer alteração.
2. Requisitos do Tocador
    - O usuário deve ser capaz de carregar e executar apenas arquivos do formato WAVE ou RAB.
    - O usuário deve ser capaz de navegar por tempo nos arquivos independente do formato.
    - O Player deve mostrar todos os metadados do *RAB* lido.
    - O usuário deve ser capaz de navegar pelo *RAB* através das marcações de conteúdo do arquivo, em todos os níveis disponíveis.

A.5 Restrições

- O Tocador e o Editor serão desenvolvidos para plataforma Unix.
- A linguagem de programação a ser utilizada é o C/C++.
- A Metodologia principal de desenvolvimento é o RUP.

ANEXO B – Documento de Requisitos

B.1 Introdução

Este artefato descreve os atores que usam os produtos do projeto, bem como os requisitos funcionais de cada um desses produtos. Cada requisito deve ser detalhado neste documento para que seja usado como referência durante as atividades de implementação.

B.2 Atores

Os atores serão o usuário do Editor e do Tocador de *audobooks*.

B.3 Casos de Uso do Editor

Nas Tabelas 4 e 5 são apresentados os casos de uso do Editor.

Tabela 4 – Caso de Uso: Inserir metadados do *audiobook*

Nome	Inserir metadados do *audiobook*
Descrição	Funcionalidade responsável por capturar os metadados inseridos pelo usuário, criar o bloco *META*.
Ator	Usuário final.
Pré-condições	Um arquivo *WAVE* ou *RAB* deve estar carregado;
Fluxo Básico	1. O usuário informa os seguintes dados do *audiobook* para o sistema: Título, Autor, Idioma, Editor, Local de Criação, Número de Páginas, Ano de Criação. 2. O sistema cria o bloco META; 3. Fim do caso de uso.
Fluxo Alternativo	Não há fluxo alternativos.
Pós-condições	O bloco *META* foi criado e está pronto para ser persistido no arquivo.

Tabela 5 – Caso de Uso: Criar Marcações de Conteúdo

Nome	Criar Marcações de Conteúdo
Descrição	Funcionalidade que registra as marcações de conteúdo inseridas pelo usuário, cria o bloco *LGMK* e salva um novo arquivo com extensão *.rab*.
Ator	Usuário final.
Pré-condições	1. Um arquivo *WAVE* ou *RAB* deve estar carregado; 2. O bloco de metadados, devidamente preenchido deve estar criado.
Fluxo Básico	1. O usuário insere o tempo, em segundos, de todas as marcações de primeiro nível; 2. Para cada marcação, o usuário define um nome, encerrando o processo para o nível 1; 3. O usuário insere o tempo, em segundos, de todas as marcações de segundo nível; 4. Para cada marcação, o usuário define um nome, encerrando o processo para o nível 2; 5. O sistema cria o bloco LGMK, salva todos os demais blocos do arquivo original junto com o bloco META e o bloco LGMK; 6. Fim do caso de uso.
Fluxo Alternativo	Não existem fluxos alternativos.
Pós-condições	O bloco *LGMK* foi criado e salvo com o bloco *META*, e um novo arquivo, com extensão RAB foi criado.

B.4 Outros Requisitos

1. O Tocador e o Editor devem carregar apenas arquivos no formato *WAVE* e *RAB*.

   - No Tocador, os arquivos *WAVE* serão lidos e executados como um arquivo de áudio comum. Porém, somente com arquivos RAB, o usuário poderá fazer uso dos marcadores de conteúdo para nevegação;
   - O Editor deve permitir alterações apenas na parte referente aos metadados e às marcações de conteúdo dos arquivos. As demais partes do arquivo não poderão

sofrer alterações.

2. O usuário deve poder navegar através dos marcadores nas funções de *forward* e *rewind*.

3. O usuário deve poder navegar no arquivos por tempo nas funções de *fast forward* e *fast rewind* do Tocador.

    - Ao acionar o comando de *fast foward*, o áudio deve avançar cinco segundos, e o *fast rewind* deve retroceder cinco segundos.

4. O Tocador deve mostrar para o usuário todos os metadados do gravador no *RAB*.

    - Título do *audiobook*, Autor, Idioma, Editor, Local de origem, Número de páginas, Ano de lançamento.

B.5 Diagramas Principais

B.5.1 Diagrama de Caso de Uso

A imagem abaixo apresenta os dois casos de uso do Editor, mostrando o ator de cada um deles:

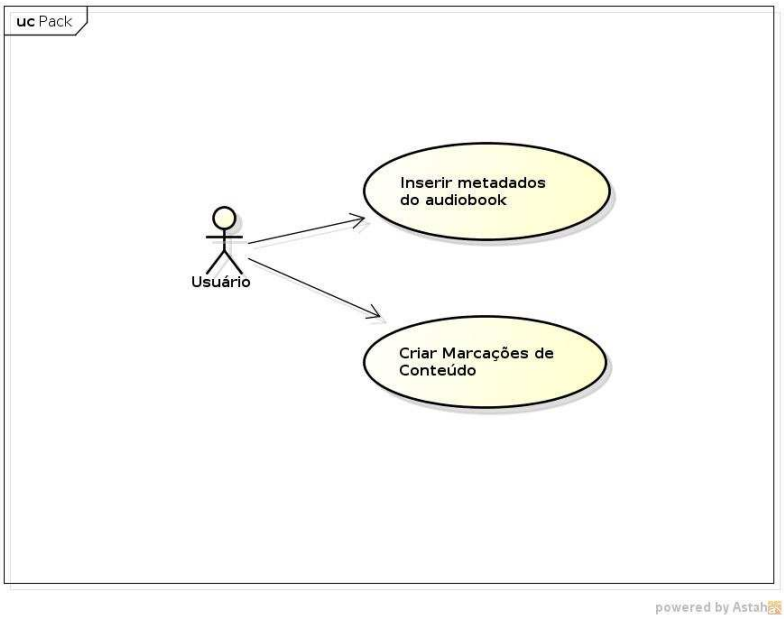

Figura 25 – Diagrama de casos de uso

ANEXO C – Especificação Suplementar

C.1 Introdução

C.1.1 Objetivo do Documento

Este documento espcifica todos os requisitos não-funcionais do Editor e Tocador de *audiobooks*.

C.2 Usabilidade

- As funções de *play* e de *pause* devem estar presentes no mesmo botão. Ele deve funcionar em dois estados, tocando ou pausado.
- Os botões do Tocador deve ser representados por ícones conhecidos dos tocadores comuns.

C.3 Confiabilidade

C.3.1 Disponibilidade

- Após o carregamento do arquivo, o Editor e Tocador, deverão estar disponíveis durante 100% do tempo de uso, não podendo haver quaisquer interrupções.

C.4 Desempenho

- O Editor e o Tocador devem carregar os arquivos em tempo máximo de 3 segundos.
- O Editor deve gerar os arquivos de *audiobook* no tempo máximo de 3 segundos.
- As funções de *fast forward*, *fast rewind*, *forward*, *rewind*, *play* e *pause* do Tocador devem responder no tempo máximo de 0,5 segundo.

C.5 Suportabilidade

- O Tocador e o Editor devem ser executados na plataforma Unix.

- O padrão MVC (*Model-View-Controler*) deve ser implementado no Tocador, para a separação entre as entidades, a interface e as controladoras de caso de uso.

ANEXO D – Pesquisa de Opinião

D.1 Tema

Pesquisa de opinião dos potenciais usuários.

D.2 Objetivos

D.2.1 Objetivo Geral

Verificar a relevância das marcações de conteúdo dos arquivos *RAB* e avaliar os aspectos de usabillidade do Tocador de *audiobooks* na opinião dos usuários.

D.3 Objeto

D.3.1 Problema

Será as marcações lógicas do formato *RAB* trazem reais benefícios para os usuários de *audiobooks*?

D.3.2 Hipótese Básica

Os usuários de *audiobooks*, fazendo uso de *players* e arquivos de audio que não suportam marcações tendem a gastar muito tempo para localizar, de forma precisa, o início uma seção específica, como um capítulo, título ou seção.

D.4 Metodologia

D.4.1 Método de Abordagem

Será utilizado o método indutivo, de modo que os resultados obtidos dos casos particulares serão generalizados, formando regras gerais.

D.4.2 Técnicas

A técnica de coleta de dados de observação direta consiste em um questionário formado perguntas de múltipla escolha e perguntas abertas. Os voluntários deverão seguir

as instruções contidas no cabeçalho do questionário para terem condições de respondê-lo adequadamente.

D.4.3 Delimitação do Universo

Alunos de graduação da Faculdade UnB Gama que cursam do quinto semestre em diante, de ambos os sexos, com conhecimentos básicos em plataformas Linux.

D.5 Instrumento de Pesquisa

D.5.1 Questionário

Questão 1: *Quanto à localização e apresentação dos metadados (Título, Autor, Editora, Idioma, ...) do audiobook carregado no Tocador. O que você achou?*

1. Está ótimo.
2. Bom.
3. Regular.
4. Ruim.
5. Péssimo.

Questão 2: *Quanto à localização e apresentação dos botões do Tocador de Audiobooks. O que você achou?*

1. Está ótimo.
2. Bom.
3. Regular.
4. Ruim.
5. Péssimo.

Questão 3: *Quanto à identificação das funcionalidades do Tocador de Audiobooks. O que você achou?*

1. Foi fácil identificar as funcionalidades.
2. Foi normal a identificação das funcionalidades.
3. Foi difícil identificar as funcionalidades.

4. Não consegui identificar nunhuma das funcionalidades.

Questão 4: *Que nota geral você atribuiria para os aspectos da interface do Tocador de Audiobooks?*

1. 5.
2. 6.
3. 7.
4. 8.
5. 9.
6. 10.

Questão 5: *Qual é a sua opinião sobre funcionalidade de navegação no arquivo através das marcações de conteúdo oferecida pelo Tocador de Audiobooks?*

1. Muito relevante.
2. Relevante.
3. Pouco Relevante.
4. Sem Relevância.

Questão 6: *Qual nota final você atribuiria para o Tocador, levando em conta todos os recursos oferecidos por ele?*

1. 5.
2. 6.
3. 7.
4. 8.
5. 9.
6. 10.

Questão 7: *Quais seriam os recursos que você adicionaria ao tocador?*

Questão 8: *Cite alguma críticas e/ou sugestão.*

ANEXO E – Especificação do Formato

E.1 *RIFF Audiobook* (*RAB*)

O formato *RAB* é baseado no *WAVE* do *Resource Interchange File Format* (*RIFF*). Além de herdar todas as características e as vantagens desse do *RIFF*, ele tem a capacidade de armazenar os metadados e as marcações de conteúdo de um *audiobook*.

A estrutura mínima do *RAB* é formada por quatro blocos de dados: *format*, *data*, *meta* e *lgmk*. Qualquer bloco diferente desses apresentados podem ser ignorados pelos programas. Os blocos *format* e *data* não apresentam qualquer diferença ao descrito na especificação original do *WAVE*. A Figura 26 ilustra o formato.

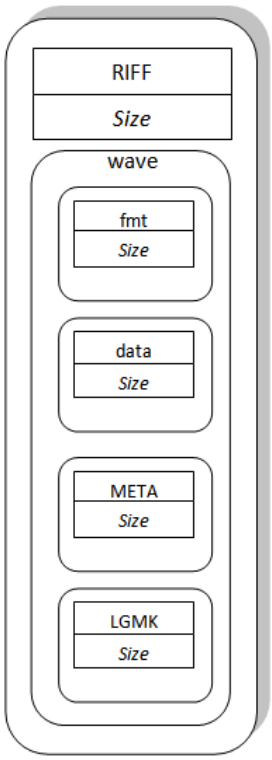

Figura 26 – Estrutura geral do *RAB*

E.1.1 O Bloco META

O bloco *META* armazena todos os metadados do *audiobook*. A figura 27 mostra os campos desse bloco.

Figura 27 – O bloco *META*

Os detalhes de cada campo do bloco *META* estão descritos a seguir:

Nome do Campo: META;

Descrição: Campo identificador do bloco;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: "`META`";

Nome do Campo: *Size*;

Descrição: Representa a soma do tamanho de todos os campos depois dele;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: Qualquer valor inteiro não-sinalizado;

Nome do Campo: Title;

Descrição: Registra o título do *audiobook*;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: Qualquer valor de texto;

Nome do Campo: Author;

Descrição: Registra o autor do *audiobook*;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: Qualquer valor de texto;

Nome do Campo: Language;

Descrição: Registra o idioma do *audiobook*;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: Qualquer valor de texto;

Nome do Campo: Publisher;

Descrição: Registra o editor do *audiobook*;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: Qualquer valor de texto;

Nome do Campo: Year;

Descrição: Registra o ano de criação do *audiobook*;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: Qualquer valor de texto. Preferencialmente os que representam números;

Nome do Campo: Address;

Descrição: Registra o local de origem do *audiobook*;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: Qualquer valor textual;

Nome do Campo: Pages;

Descrição: Registra a quantidade de paginas lo livro impresso;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: Qualquer valor de texto. Preferencialmente os que representam números;

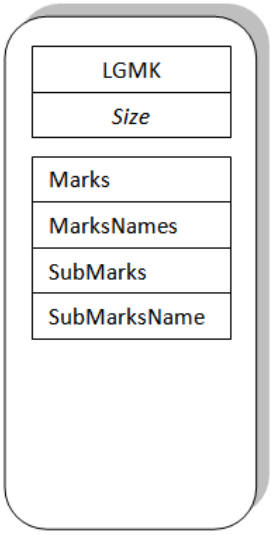

Figura 28 – O bloco *LGMK*

E.1.2 O Bloco LGMK

O bloco *LGMK* contém as marcações lógicas do *audiobook*. A Figura 28 mostra a estrutura desse bloco.

Os detalhes de cada campo do bloco *LGMK* estão descritos abaixo:

Nome do Campo: LGMK;

Descrição: Campo identificador do bloco;

Tipo Correspondente em *C++*: `string`;

Valores Válidos: "`LGMK`";

Nome do Campo: *Size*;

Descrição: Representa a soma do tamanho de todos os campos depois dele;

Tipo Correspondente em *C++*: `uint32_t`;

Valores Válidos: Qualquer valor inteiro não-sinalizado;

Nome do Campo: Marks;

Descrição: Campo responsável por armazenar as marcações lógicas de nível 1. Os valores devem corresponder ao tempo, em segundos, de início da marcação;

Tipo Correspondente em *C++*: `vector<uint32_t>`;

Valores Válidos: Qualquer conjunto de números inteiros não-sinalizados: $\{0, 39, 79, 101, ...\}$

Nome do Campo: MarksNames;

Descrição: Campo que armazena os nomes de cada marcação de nível 1;

Tipo Correspondente em *C++*: `vector<string>`;

Valores Válidos: Qualquer conjunto de valores de texto: {Capítulo 1, Capítulo 2, Capítulo 3, Capítulo 4, Capítulo 5...}.

Nome do Campo: SubMarks;

Descrição: Campo responsável por armazenar as marcações lógicas de nível 2. Os valores devem corresponder ao tempo, em segundos, de início da marcação;

Tipo Correspondente em *C++*: `vector<uint32_t>`;

Valores Válidos: Qualquer conjunto de números inteiros não-sinalizados: $\{0, 10, 24, 50, ...\}$.

Nome do Campo: SubMarksNames;

Descrição: Campo que armazena os nomes de cada marcação de nível 2;

Tipo Correspondente em *C++*: `vector<string>`;

Valores Válidos: Qualquer conjunto de valores de texto: {Seção 1.1, Seção 1.2, Seção 2.1, Seção 2.2, Seção 2.3, Seção 3.1, ...}.

ANEXO F – Projeto e Relatório da Pesquisa de Opinião

O projeto é o primeiro passo para o desenvolvimento de uma pesquisa. Para que ela seja executada com sucesso, é necessário um rigoroso planejamento, de forma que as perguntas principais *o quê? por que? para quê? para quem? onde? como? com quê? quanto? quando? quem? com quanto?* sejam respondidas. Tudo isso irá contribuir para que o conjunto de dados colhidos sejam claramente apresentados e interpretados.

De modo genérico, a estrutura do projeto de pesquisa de acordo com os autores (LAKATOS; MARCONI, 2012), é apresentado da seguinte maneira:

1. Apresentação (*quem?*)
    a) Capa
        - entidades
        - título
        - coordenadores
        - local e data
    b) Relação do pessoal técnico
        - entidade (nome, endereço, telefone)
        - coordenadores (nome, endereço, telefone)
        - pessoal técnico (cargo, endereço, telefone)
2. Objetivos (*para quê? para quem?*)
    a) Tema
    b) Delimitação do Tema
        - especificação
        - limitação geográfica e temporal
    c) Objetivo Geral
    d) Objetivos Específicos
3. Justificativa (*por quê?*)
4. Objeto (*o quê?*)

a) Problema

b) Hipótese Básica

c) Hipóteses Secundárias

d) Variáveis

e) Relação entre Variáveis

5. Metodologia (*como? com quê? quanto? quando?*)

a) Método de Abordagem

b) Métodos de Procedimento

c) Técnicas

- descrição
- como será aplicado
- codificação e tabulação

d) Delimitação do Universo (descrição da população)

e) Tipo de Amostragem

- caracterização
- seleção

f) Tratamento Estatístico

- modelo de experimento
- nível de significância
- variáveis controladas
- medidas
- testes de hipóteses

6. Embasamento Teórico (*como?*)

a) Teoria de Base

b) Revisão da Bibliografia

c) Definição dos Termos

d) Conceitos Operacionais e Indicadores

7. Cronograma (*quando?*)

8. Orçamento (*com quanto?*)

9. Instrumentos de Pesquisa (*como?*)

10. Bibliografia

Depois da coleta de dados, codificação e tabulação, tratamento estatísticos, análise e interpretação, os resultados estão prontos para serem redigidos através do relatório de pesquisa. (LAKATOS; MARCONI, 2012) define a estrutura do relatório de acordo com a *outline* abaixo.

1. Apresentação
    a) Capa
        - entidades
        - título
        - coordenadores
        - local e data
    b) Página de Rosto
        - entidades
        - título
        - coordenadores
        - equipe técnica
        - local e data
2. Sinopse (*abstract*)
3. Sumário
4. Introdução
    a) Objetivo
        - tema
        - delimitação do tema
        - objetivo geral
        - objevos específicos
    b) Justificativa
    c) Objeto
        - problema
        - hipótese básica
        - hipóteses secundárias
        - variáveis
        - relação entre variáveis
5. Revisão da Bibliografia

6. Metodologia
    a) Método de Abordagem
    b) Método de Procedimento
    c) Técnicas
    d) Delimitação do Universo
    e) Tipo de Abordagem
    f) Tratamento Estatístico
7. Embasamento Teórico
    a) Teoria de Base
    b) Definição dos Termos
    c) Conceitos Operacionais e Indicadores
8. Apresentação dos Dados e sua Análise
9. Interpretação dos Resultados
10. Conclusão
11. Recomendações e Sugestões
12. Apêndices
    a) Tabelas
    b) Quadros
    c) Gráficos
    d) Outras Ilustrações
    e) Instrumento(s) de Pesquisa
13. Anexos
14. Bibliografia

ANEXO G – Código Fonte da Biblioteca Estática (*Core*)

```
#ifndef CHUNK_H
#define CHUNK_H

#include <iostream>
#include <string.h>
#include "data.h"

using namespace std;

class Chunk {
        friend ostream& operator<<(ostream& os, const Chunk *chunk)
        {
                chunk->print(os);
                return os;
        }

public:
        virtual ~Chunk() {}

        virtual Data * encode() const = 0;
        virtual uint32_t decode(const Data& data,
                                uint32_t offset = 0) = 0;

        virtual void print(ostream& os) const = 0;
};

#endif
```

```
#ifndef CHUNK_FACTORY_H
#define CHUNK_FACTORY_H

#include "chunk.h"

using namespace std;

class ChunkFactory {
public:
        static Chunk * decode(const Data& data,
                              uint32_t *decoded, uint32_t offset = 0);
};

#endif
```

```
#ifndef DATA_H
#define DATA_H

#include <stdint.h>

class Data {
public:
        Data();
        ~Data();

        void set(uint8_t *bytes, int amount);

        uint32_t size() const;
        const uint8_t * bytes(uint32_t offset = 0) const;


private:
        uint32_t m_size;
        uint8_t *m_bytes;
};

#endif
```

```
#ifndef FORMAT_H
#define FORMAT_H

#include <stdint.h>
#include <string.h>

#include "chunk.h"

using namespace std;

class Format : public Chunk
{
public:
        Format();

        Data * encode() const;
        uint32_t decode(const Data& data, uint32_t offset = 0);

        uint16_t audioFormat() const;
        uint16_t numChannels() const;
        uint32_t sampleRate() const;
        uint16_t bitsPerSample() const;

        void print(ostream& os) const;

        static const string id;

private:
        uint16_t m_audioFormat;
        uint16_t m_numChannels;
        uint32_t m_sampleRate;
        uint32_t m_byteRate;
        uint16_t m_blockAlign;
        uint16_t m_bitsperSample;

        uint32_t size() const;

};
```

```
#endif
```

```
#ifndef GENERIC_H
#define GENERIC_H

#include <iostream>
#include "chunk.h"

using namespace std;

class Generic : public Chunk {
public:
        Generic();
        ~Generic();

        Data * encode() const;
        uint32_t decode(const Data& data, uint32_t offset = 0);

        void print(ostream& os) const;

        string id() const;
        uint32_t size() const;
        const uint8_t * data() const;

private:
        string m_id;
        uint32_t m_size;
        uint8_t *m_data;
};

#endif
```

```
#ifndef LGMK_H
#define LGMK_H

#include <string.h>

#include <stdint.h>

#include <algorithm>
#include <vector>
#include <string>

#include "chunk.h"

class Lgmk : public Chunk
{
public:
        Lgmk();

        Data *encode() const;
        uint32_t decode(const Data& data, uint32_t offset = 0);

        void print(ostream& os) const;

        vector<uint32_t> marks() const;
        vector<string> marksNames() const;

        vector<uint32_t> subMarks() const;
        vector<string> subMarksNames() const;

        void add_mark(uint32_t position);
        void add_markName(string name);

        void add_subMark(uint32_t position);
        void add_subMarkName(string name);

        static const string id;
private:
        vector<uint32_t> m_marks;
        vector<uint32_t> m_subMarks;
```

```
        vector<string> m_marksNames;
        vector<string> m_subMarksNames;

        uint32_t size() const;
};

#endif
```

```
#ifndef META_H
#define META_H

#include <string.h>

#include <string>
#include <vector>

#include "chunk.h"

class Meta : public Chunk {
public:
        Meta();

        Data * encode() const;
        uint32_t decode(const Data& data, uint32_t offset = 0);

        void print(ostream& os) const;

        void setTitle(const string& title);
        void setAuthor(const string& author);
        void setLanguage(const string& language);
        void setPublisher(const string& publisher);
        void setYear(const string& year);
        void setAddress(const string& address);
        void setpages(const string& pages);

        string title() const;
        string author() const;
        string language() const;
        string publisher() const;
        string year() const;
        string address() const;
        string pages () const;

        static const string id;

private:
        enum {TITLE = 0, AUTHOR, LANGUAGE,
```

```
                PUBLISHER, YEAR, ADDRESS, PAGES, COVER};

        vector<string> fields;

        uint32_t size() const;
};

#endif  /* META_H */
```

```
#ifndef SOUND_H
#define SOUND_H

#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

using namespace std;

class Sound
{
public:
        Sound();

        uint8_t *buffer() const;
        int size() const;

        void setBuffer(uint8_t *buf);
        void setSize(int size);

        int m_position;


private:

        uint8_t *m_buffer;
        int m_size;
};


extern void callback(void *userdata, Uint8 *stream, int len);

extern ostream& operator<<(ostream& os, const SDL_AudioSpec& spec);

#endif
```

```
#ifndef WAVE_H
#define WAVE_H

#include <SDL/SDL.h>
#include <string>
#include <vector>

#include "chunk.h"
#include "format.h"
#include "generic.h"

using namespace std;

class Wave : public Chunk {
public:
        ~Wave();

        Data * encode() const;
        uint32_t decode(const Data& data, uint32_t offset = 0);

        void print(ostream& os) const;
        void save(const string& path) const;
        vector<Chunk *> subchunks() const;

        static Wave * load(const string& path);
        static const string id;
        static const uint32_t size;

        void add_chunk(Chunk *chunk);

        SDL_AudioSpec spec() const;
        Generic *data() const;

private:
        vector<Chunk *> m_subchunks;

        Format *format() const;

};
```

```
extern SDL_AudioSpec* Load_Wave(const char *file,
        SDL_AudioSpec *spec, uint8_t **audio_buf,
        uint32_t *audio_len);

#endif
```

```
#include "chunkfactory.h"
#include "generic.h"
#include "meta.h"
#include "format.h"
#include "lgmk.h"

Chunk *
ChunkFactory::decode(const Data& data,
                     uint32_t *decoded, uint32_t offset)
{
        Chunk *chunk = 0;

        const uint8_t *bytes = data.bytes(offset);

        char id[5];
        memcpy(id, bytes, 4);
        id[4] = 0;

        if (memcmp(id, "META", 4) == 0)
        {
                chunk = new Meta();
        } else if(memcmp(id, "fmt ", 4) == 0)
        {
                chunk = new Format();
        } else if(memcmp(id, "LGMK", 4) == 0)
        {
                chunk = new Lgmk();
        }else
        {
                chunk = new Generic();
        }

        if (chunk == 0)
        {
                return 0;
        }

        (*decoded) += chunk->decode(data, offset);
```

```
        return chunk;
}
```

```
#include "data.h"

Data::Data()
{
        m_bytes = 0;
        m_size = 0;
}

Data::~Data()
{
        delete [] m_bytes;
}

void
Data::set(uint8_t *bytes, int amount)
{
        if (m_bytes)
        {
                delete [] m_bytes;
        }

        m_bytes = bytes;
        m_size = amount;
}

uint32_t
Data::size() const
{
        return m_size;
}

const uint8_t *
Data::bytes(uint32_t offset) const
{
        return m_bytes + offset;
}
```

```
#include "format.h"

const string Format::id = "fmt ";

Format::Format()
{
        m_audioFormat = 0;
        m_numChannels = 0;
        m_sampleRate = 0;
        m_byteRate = 0;
        m_blockAlign = 0;
        m_bitsperSample = 0;
}

Data *
Format::encode() const
{
        const int ID_LENGTH = 4;
        const int SIZE_LENGTH = 4;
        const int HEADER_LENGTH = ID_LENGTH + SIZE_LENGTH;

        uint32_t encoded_size = size();

        uint32_t total_size = encoded_size + HEADER_LENGTH;

        uint8_t *bytes = new uint8_t[total_size];


        if (bytes == 0)
                return 0;

        uint32_t offset = 0;

        memcpy(bytes + offset, Format::id.c_str(), ID_LENGTH);
        offset += ID_LENGTH;

        memcpy(bytes + offset, &encoded_size, SIZE_LENGTH);
        offset += SIZE_LENGTH;
```

```
        //m_audioFormat
        uint16_t audioFormatSize = sizeof(m_audioFormat);
        memcpy(bytes + offset, &m_audioFormat, audioFormatSize);
        offset += audioFormatSize;

        //m_numChannels
        uint16_t numChannelsSize = sizeof(m_numChannels);
        memcpy(bytes + offset, &m_numChannels, numChannelsSize);
        offset += numChannelsSize;

        //m_sampleRate
        uint32_t sampleRateSize = sizeof(m_sampleRate);
        memcpy(bytes + offset, &m_sampleRate, sampleRateSize);
        offset += sampleRateSize;

        //m_byteRate
        uint32_t byteRateSize = sizeof(m_byteRate);
        memcpy(bytes + offset, &m_byteRate, byteRateSize);
        offset += byteRateSize;

        //m_blockAlign;
        uint16_t blockAlignSize = sizeof(m_blockAlign);
        memcpy(bytes + offset, &m_blockAlign, blockAlignSize);
        offset += blockAlignSize;

        //m_bitsperSample
        uint16_t bitsperSampleSize = sizeof(m_bitsperSample);
        memcpy(bytes + offset, &m_bitsperSample, bitsperSampleSize);
        offset += bitsperSampleSize;

        Data *data = new Data();
        data->set(bytes, total_size);

        return data;
}

uint32_t
Format::decode(const Data& data, uint32_t offset)
{
```

```
const uint8_t *iterator = data.bytes(offset);
uint32_t decoded = 0;

// Read id
char buffer[5];
memcpy(buffer, iterator + decoded, 4);
buffer[4] = 0;

if (memcmp(buffer, Format::id.c_str(), 4))
        return 0;

decoded += 4;

uint32_t size;
memcpy(&size, iterator + decoded, 4);
decoded += 4;

memcpy(&m_audioFormat, iterator + decoded, 2);
decoded += 2;

memcpy(&m_numChannels, iterator + decoded, 2);
decoded += 2;

memcpy(&m_sampleRate, iterator + decoded, 4);
decoded += 4;

memcpy(&m_byteRate, iterator + decoded, 4);
decoded += 4;

memcpy(&m_blockAlign, iterator + decoded, 2);
decoded += 2;

memcpy(&m_bitsperSample, iterator + decoded, 2);
decoded += 2;

//if(data.size()-16 > 0)
        //decoded += (data.size()-16);

return decoded;
```

```
}

uint16_t
Format::audioFormat() const
{
        return m_audioFormat;
}

uint16_t
Format::numChannels() const
{
        return m_numChannels;
}

uint32_t
Format::sampleRate() const
{
        return m_sampleRate;
}

void
Format::print(ostream& os) const
{
        os << "\tID: " << Format::id << endl;
        os << "\tsize: " << size() << endl;
        os << "\t\taudioFormat: [" << m_audioFormat << "]" << endl;
        os << "\t\tnumChannels: [" << m_numChannels << "]" << endl;
        os << "\t\tsampleRate: [" << m_sampleRate << "]" << endl;
        os << "\t\tbyteRate: [" << m_byteRate << "]" << endl;
        os << "\t\tblockAlign: [" << m_blockAlign << "]" << endl;
        os << "\t\tbitsperSample: [" << m_bitsperSample << "]" << endl;
}


uint32_t
Format::size() const
{
        uint32_t s = 0;
```

```
        s += sizeof(m_audioFormat);
        s += sizeof(m_numChannels);
        s += sizeof(m_sampleRate);
        s += sizeof(m_byteRate);
        s += sizeof(m_blockAlign);
        s += sizeof(m_bitsperSample);

        return s;
}

uint16_t
Format::bitsPerSample() const
{
        return m_bitsperSample;
}
```

```
#include "generic.h"
#include "string.h"

void
Generic::print(ostream& os) const
{
        os << "\tID: " << m_id << endl;
        os << "\tsize: " << m_size << endl;
}

Generic::Generic()
{
        m_size = 0;
        m_data = 0;
        m_id = "UNKW";
}

Generic::~Generic()
{
        delete [] m_data;
}

Data *
Generic::encode() const
{
        const int ID_LENGTH = 4;
        const int SIZE_LENGTH = 4;
        const int HEADER_LENGTH = ID_LENGTH + SIZE_LENGTH;

        uint8_t *bytes = new uint8_t[HEADER_LENGTH + m_size];

        if (bytes == 0)
                return 0;

        memcpy(bytes, m_id.c_str(), ID_LENGTH);
        memcpy(bytes + ID_LENGTH, &m_size, SIZE_LENGTH);
        memcpy(bytes + HEADER_LENGTH, m_data, m_size);

        Data *data = new Data();
```

```
        data->set(bytes, HEADER_LENGTH + m_size);

        return data;
}

uint32_t
Generic::decode(const Data& data, uint32_t offset)
{
        const uint8_t *iterator = data.bytes(offset);
        uint32_t decoded = 0;

        // Read id
        char buffer[5];
        memcpy(buffer, iterator + decoded, 4);
        buffer[4] = 0;

        m_id = string(buffer);
        decoded += 4;

        // Read size
        memcpy(&m_size, iterator + decoded, 4);
        decoded += 4;

        // Read data bytes
        m_data = new uint8_t[m_size];

        if (m_data != 0)
        {
                memcpy(m_data, iterator + decoded, m_size);
        }

        decoded += m_size;

        return decoded;
}

string
Generic::id() const
{
```

```
        return m_id;
}

uint32_t
Generic::size() const
{
        return m_size;
}

const uint8_t *
Generic::data() const
{
        return m_data;
}
```

```
#include "lgmk.h"

const string Lgmk::id = "LGMK";

Lgmk::Lgmk()
{}

Data *
Lgmk::encode() const
{
        const int ID_LENGTH = 4;
        const int SIZE_LENGTH = 4;
        const int HEADER_LENGTH = ID_LENGTH + SIZE_LENGTH;

        uint32_t encoded_size = size();

        uint32_t total_size = encoded_size + HEADER_LENGTH;

        uint8_t *bytes = new uint8_t[total_size];

        if (bytes == 0)
                return 0;

        uint32_t offset = 0;

        memcpy(bytes + offset, Lgmk::id.c_str(), ID_LENGTH);
        offset += ID_LENGTH;

        memcpy(bytes + offset, &encoded_size, SIZE_LENGTH);
        offset += SIZE_LENGTH;

        //m_marks
        uint32_t marksCount = m_marks.size();
        memcpy(bytes + offset, &marksCount, SIZE_LENGTH);
        offset += SIZE_LENGTH;

        for (vector<uint32_t>::const_iterator it =
                m_marks.begin(); it != m_marks.end(); it++)
        {
```

```
            uint32_t mark = *it;
            memcpy(bytes + offset, &mark, 4);
            offset += 4;
    }

    //m_subMarks
    uint32_t subMarksCount = m_subMarks.size();
    memcpy(bytes + offset, &subMarksCount, SIZE_LENGTH);
    offset += SIZE_LENGTH;

    for (vector<uint32_t>::
            const_iterator it = m_subMarks.begin();
            it != m_subMarks.end(); it++)
    {
            uint32_t mark = *it;
            memcpy(bytes + offset, &mark, 4);
            offset += 4;
    }

    //m_marksNames
    uint32_t marksNamesCount = m_marksNames.size();
    memcpy(bytes + offset, &marksNamesCount, SIZE_LENGTH);
    offset += SIZE_LENGTH;

    for(vector<string>::
            const_iterator it = m_marksNames.begin();
            it != m_marksNames.end(); it++)
    {
            uint32_t marksSize = (*it).size();
            memcpy(bytes + offset, &marksSize, SIZE_LENGTH);
            offset += SIZE_LENGTH;

            string markName = *it;
            memcpy(bytes + offset, markName.c_str(), markName.size());
            offset += markName.size();
    }


    //m_marksNames
```

```
        uint32_t subMarksNamesCount = m_subMarksNames.size();
        memcpy(bytes + offset, &subMarksNamesCount, SIZE_LENGTH);
        offset += SIZE_LENGTH;

        for(vector<string>::const_iterator
                it = m_subMarksNames.begin();
                        it != m_subMarksNames.end(); it++)
        {
                uint32_t marksSize = (*it).size();
                memcpy(bytes + offset, &marksSize, SIZE_LENGTH);
                offset += SIZE_LENGTH;

                string markName = *it;
                memcpy(bytes + offset,
                        markName.c_str(), markName.size());

                offset += markName.size();
        }

        Data *data = new Data();
        data->set(bytes, total_size);

        return data;
}

uint32_t
Lgmk::decode(const Data& data, uint32_t offset)
{
        const uint8_t *iterator = data.bytes(offset);
        uint32_t decoded = 0;

        // Read id
        char buffer[5];
        memcpy(buffer, iterator + decoded, 4);
        buffer[4] = 0;

        if (memcmp(buffer, Lgmk::id.c_str(), 4))
                return 0;
```

```
decoded += 4;

uint32_t size;
memcpy(&size, iterator + decoded, 4);
decoded += 4;

uint32_t marks;
memcpy(&marks, iterator + decoded, 4);
decoded += 4;

for (uint32_t i = 0; i < marks; i++)
{
        uint32_t mark;
        memcpy(&mark, iterator + decoded, 4);
        decoded += 4;
        m_marks.push_back(mark);
}

uint32_t subMarks;
memcpy(&subMarks, iterator + decoded, 4);
decoded += 4;

for (uint32_t i = 0; i < subMarks; i++)
{
        uint32_t mark;
        memcpy(&mark, iterator + decoded, 4);
        decoded += 4;
        m_subMarks.push_back(mark);
}

uint32_t marksNames;
memcpy(&marksNames, iterator + decoded, 4);
decoded += 4;

uint32_t temp_size;

cout << "Marks Names: " <<marksNames <<endl;
for(uint32_t i = 0; i < marksNames; i++)
{
```

```
            memcpy(&temp_size, iterator + decoded, 4);
            decoded += 4;

            cout << "temp_size: "<< temp_size <<endl;

            char *temp_char = new char[temp_size + 1];

            if(temp_char != 0)
            {

                    temp_char[temp_size] = 0;
                    memcpy(temp_char,
                            iterator + decoded, temp_size);

                    string mark(temp_char);

                    cout <<"temp_char: {" << mark << "}"<<endl;

                    m_marksNames.push_back(mark);

                    delete [] temp_char;
            }

            decoded += temp_size;
    }


    uint32_t subMarksNames;
    memcpy(&subMarksNames, iterator + decoded, 4);
    decoded += 4;

    uint32_t temp_size2;

    cout << "SubMarks Names: " << subMarksNames <<endl;
    for(uint32_t i = 0; i < subMarksNames; i++)
    {
            memcpy(&temp_size2, iterator + decoded, 4);
            decoded += 4;
```

```
                cout << "temp_size2: "<< temp_size2 <<endl;

                char *temp_char = new char[temp_size2 + 1];

                if(temp_char != 0)
                {
                        temp_char[temp_size2] = 0;
                        memcpy(temp_char,
                                iterator + decoded, temp_size2);

                        string mark(temp_char);

                        cout <<"temp_char: {" << mark << "}"<<endl;

                        m_subMarksNames.push_back(mark);

                        delete [] temp_char;
                }

                decoded += temp_size2;
        }

        return decoded;
}

vector<uint32_t>
Lgmk::marks() const
{
        return m_marks;
}

vector<string>
Lgmk::marksNames() const
{
        return m_marksNames;
}

vector<uint32_t>
Lgmk::subMarks() const
```

```
{
        return m_subMarks;
}

vector<string>
Lgmk::subMarksNames() const
{
        return m_subMarksNames;
}

void
Lgmk::add_mark(uint32_t position)
{
        m_marks.push_back(position);
        sort(m_marks.begin(), m_marks.begin() + m_marks.size());
}

void
Lgmk::add_markName(string name)
{
        m_marksNames.push_back(name);
}

void
Lgmk::add_subMark(uint32_t position)
{
        m_subMarks.push_back(position);
        sort(m_subMarks.begin(),
              m_subMarks.begin() + m_subMarks.size());
}

void
Lgmk::add_subMarkName(string name)
{
        m_subMarksNames.push_back(name);
}

void
Lgmk::print(ostream& os) const
```

```
{
        os << "\tID: " << Lgmk::id << endl;
        os << "\tsize: " << size() << endl;

        os << "\t\tmarks (" << m_marks.size() << "): ";
        for(uint32_t i = 0; i < m_marks.size(); i++)
        {
                if (i)
                        os << ", ";

                os << m_marks[i];
        }

        os << '\n';

        os << "\t\tsubMarks (" << m_subMarks.size() << "): ";
        for(uint32_t j = 0; j < m_subMarks.size(); j++)
        {
                if (j)
                        os << ", ";

                os << m_subMarks[j];

        }

        os << '\n';

        os << "\t\tmarks names (" << m_marksNames.size() << "): ";
        for(uint32_t i = 0; i < m_marksNames.size(); i++)
        {
                if (i)
                        os << ", ";

                os << m_marksNames[i];
        }

        os << '\n';

        os << "\t\tsubMarks names (" << m_subMarksNames.size() << "): ";
```

```
        for(uint32_t j = 0; j < m_subMarksNames.size(); j++)
        {
                if (j)
                        os << ", ";

                os << m_subMarksNames[j];
        }
}

uint32_t
Lgmk::size() const
{
        uint32_t stringMarkSize = 0;
        uint32_t stringSubMarkSize = 0;

        for(uint32_t i = 0; i < m_marksNames.size(); i++)
        {
                stringMarkSize += m_marksNames[i].size();
        }

        for(uint32_t i = 0; i < m_subMarksNames.size(); i++)
        {
                stringSubMarkSize += m_subMarksNames[i].size();
        }

        return (2*sizeof(uint32_t) +
                2*sizeof(uint32_t) * m_marks.size() +
                stringMarkSize) + (2*sizeof(uint32_t) +
                2*sizeof(uint32_t) * m_subMarks.size() +
                stringSubMarkSize);
}
```

```
#include "meta.h"

const string Meta::id = "META";

Meta::Meta()
{
        for (int i = 0; i <= COVER; i++)
        {
                fields.push_back("Unknown");
        }
}

Data *
Meta::encode() const
{
        const int ID_LENGTH = 4;
        const int SIZE_LENGTH = 4;
        const int HEADER_LENGTH = ID_LENGTH + SIZE_LENGTH;
        uint32_t encoded_size = size() + fields.size()*SIZE_LENGTH;

        uint32_t total_size = encoded_size + HEADER_LENGTH;

        uint8_t *bytes = new uint8_t[total_size];

        if (bytes == 0)
                return 0;

        uint32_t offset = 0;

        memcpy(bytes + offset, Meta::id.c_str(), ID_LENGTH);
        offset += ID_LENGTH;

        memcpy(bytes + offset, &encoded_size, SIZE_LENGTH);
        offset += SIZE_LENGTH;

        uint32_t fieldsCount = fields.size();

        for(uint32_t i = 0; i < fieldsCount; i++)
        {
```

```
                int size = fields[i].size();

                memcpy(bytes + offset, &size, SIZE_LENGTH);
                offset += SIZE_LENGTH;

                memcpy(bytes + offset, fields[i].c_str(), size);
                offset += size;
        }

        Data *data = new Data();
        data->set(bytes, total_size);

        return data;
}

uint32_t
Meta::decode(const Data& data, uint32_t offset)
{
        const uint8_t *iterator = data.bytes(offset);
        uint32_t decoded = 0;

        // Read id
        char buffer[5];
        memcpy(buffer, iterator + decoded, 4);
        buffer[4] = 0;

        if (memcmp(buffer, Meta::id.c_str(), 4))
                return 0;

        decoded += 4;

        // Read size
        uint32_t size;
        memcpy(&size, iterator + decoded, 4);
        decoded += 4;

        uint32_t temp_size;
        uint32_t fieldsCount = fields.size();
```

```
        for(uint32_t i = 0; i < fieldsCount; i++)
        {
                memcpy(&temp_size, iterator + decoded, 4);
                decoded += 4;

                char *temp_char = new char[temp_size + 1];

                if(temp_char != 0)
                {
                        temp_char[temp_size] = 0;
                        memcpy(temp_char, iterator + decoded, temp_size);
                        fields[i] = temp_char;

                        delete [] temp_char;
                }

                decoded += temp_size;
        }

        return decoded;
}

void
Meta::print(ostream& os) const
{
        os << "\tID: " << Meta::id << endl;
        os << "\tsize: " << size() << endl;
        os << "\t\ttitle: [" << fields[TITLE] << "]" << endl;
        os << "\t\tauthor: [" << fields[AUTHOR] << "]" << endl;
        os << "\t\tlanguage: [" << fields[LANGUAGE] << "]" << endl;
        os << "\t\tpublisher: [" << fields[PUBLISHER] << "]"<< endl;
        os << "\t\tyear: [" << fields[YEAR] << "]"<< endl;
        os << "\t\taddress: [" << fields[ADDRESS] << "]"<< endl;
        os << "\t\tpages: [" << fields[PAGES] << "]"<< endl;
}

uint32_t
Meta::size() const
{
```

```
        uint32_t total = 0;
        uint32_t fieldsCount = fields.size();

        for (uint32_t i = 0; i < fieldsCount; i++)
        {
                total += fields[i].size();
        }

        return total;
}

void
Meta::setTitle(const string& title)
{
        fields[TITLE] = title;
}

void
Meta::setAuthor(const string& author)
{
        fields[AUTHOR] = author;
}

void
Meta::setLanguage(const string& language)
{
        fields[LANGUAGE] = language;
}

void
Meta::setPublisher(const string& publisher)
{
        fields[PUBLISHER] = publisher;
}

void
Meta::setYear(const string& year)
{
        fields[YEAR] = year;
```

```
}

void
Meta::setAddress(const string& address)
{
        fields[ADDRESS] = address;
}

void
Meta::setpages(const string& pages)
{
        fields[PAGES] = pages;
}

string
Meta::title() const
{
        return fields[TITLE];
}

string
Meta::author() const
{
        return fields[AUTHOR];
}

string
Meta::language() const
{
        return fields[LANGUAGE];
}

string
Meta::publisher() const
{
        return fields[PUBLISHER];
}

string
```

```
Meta::year() const
{
        return fields[YEAR];
}

string
Meta::address() const
{
        return fields[ADDRESS];
}

string
Meta::pages () const
{
        return fields[PAGES];
}
```

```
#include "sound.h"

Sound::Sound()
{
        m_position = -1;
        m_buffer = 0;
        m_size = 0;
}

uint8_t *
Sound::buffer() const
{
        return m_buffer;
}

int
Sound::size() const
{
        return m_size;
}

void
Sound::setBuffer(uint8_t *buf)
{
        m_buffer = buf;
}

void
Sound::setSize(int size)
{
        m_size = size;
}

void
callback(void *userdata, Uint8 *audio, int length)
{
        memset(audio, 0, length);

        Sound *sound = (Sound *) userdata;
```

```
        if (sound->m_position == -1 ||
                sound->m_position >= sound->size())

                return;

        int nextSamplesLength;

        if (sound->m_position + length > sound->size())
                nextSamplesLength =
                        sound->size() - sound->m_position;
        else
                nextSamplesLength = length;

        SDL_MixAudio(audio, sound->buffer() +
                sound->m_position, nextSamplesLength,
                SDL_MIX_MAXVOLUME / 2);

        sound->m_position += nextSamplesLength;
}

ostream&
operator<<(ostream& os, const SDL_AudioSpec& spec)
{
        os << "Frequencia: " << spec.freq << endl;
        os << "Formato: ";

        switch (spec.format) {
        case AUDIO_U8:
                os << "Amostras de 8-bits nao sinalizadas" << endl;
                break;

        case AUDIO_S8:
                os << "Amostras  de 8-bits sinalizadas" << endl;
                break;

        case AUDIO_U16:
                os << "Amostras de 16-bits nao"
                        "sinalizadas (little endian)" << endl;
```

```
            break;

        case AUDIO_U16MSB:
            os << "Amostras de 16-bits "
                  "nao sinalizadas e big endian" << endl;
            break;

        case AUDIO_S16:
            os << "Amostras de 16-bits"
                  "sinalizadas (little endian)" << endl;
            break;

        case AUDIO_S16MSB:
            os << "Amostras de "
                  "16-bits sinalizadas e big endian" << endl;
            break;
        }

        os << "Canais: " << (int) spec.channels << endl;
        os << "Numero de amostras: " << spec.samples << endl;
        os << "Tamanho: " << spec.size << endl;
        os << "Silencio: " << spec.silence << endl;

        return os;
}
```

```
#include <cstdio>
#include <fstream>
#include <string.h>
#include <string>
#include "chunkfactory.h"
#include "generic.h"
#include "wave.h"
#include "format.h"
#include "data.h"

Wave *
Wave::load(const string& path)
{
        // Open file
        ifstream file(path.c_str());

        if (file.is_open() == false)
                return 0;

        // Compute file size
        long begin = file.tellg();
        file.seekg(0, ios::end);
        long end = file.tellg();

        long file_size = end - begin;

        file.seekg(0, ios::beg);

        // Prepare byte array
        uint8_t *bytes = new uint8_t[file_size];

        if (bytes == 0)
        {
                file.close();
                return 0;
        }

        // Read file into byte array
        file.read((char *) bytes, file_size);
```

```
        file.close();

        // Prepara data to decode
        Data data;
        data.set(bytes, file_size);

        // Create a new wave
        Wave *wave = new Wave();

        if (wave == 0)
        {
                return 0;
        }

        // Decode bytes into wave
        if (wave->decode(data))
        {
                return wave;
        } else
        {
                delete wave;
                return 0;
        }
}

const string Wave::id = "RIFF";

void
Wave::print(ostream& os) const
{
        os << "[" << Wave::id << "]" << endl;

        int count = m_subchunks.size();

        for (int i = 0; i < count; i++)
                m_subchunks[i]->print(os);
}

Wave::~Wave()
```

```
{
        int count = m_subchunks.size();

        for (int i = 0; i < count; i++)
                delete m_subchunks[i];
}

uint32_t
Wave::decode(const Data& data, uint32_t offset)
{
        const uint8_t *iterator = data.bytes(offset);
        uint32_t decoded = 0;

        // Check id
        if (memcmp(iterator, Wave::id.c_str(), 4))
                return 0;

        decoded += 4;

        // Read size
        uint32_t size;
        memcpy(&size, iterator + decoded, 4);

        decoded += 4;

        // Check wave format
        if (memcmp(iterator + decoded, "WAVE", 4))
                return 0;

        decoded += 4;

        // Read subchunks
        Chunk *chunk = 0;

        while (decoded + offset < data.size())
        {
                chunk = ChunkFactory::decode(data,
                        &decoded, offset + decoded);
```

```
                m_subchunks.push_back(chunk);
        }

        return decoded;
}

Data *
Wave::encode() const
{
        vector<Data *> encoded;
        uint32_t subchunks_length = 0;

        for (unsigned int i = 0; i < m_subchunks.size(); i++)
        {
                encoded.push_back(m_subchunks[i]->encode());
                subchunks_length += encoded[i]->size();
        }

        const uint32_t ID_LENGTH = 4;
        const uint32_t SIZE_LENGTH = 4;
        const uint32_t FORMAT_LENGTH = 4;
        const uint32_t HEADER_LENGTH = ID_LENGTH +
                        SIZE_LENGTH + FORMAT_LENGTH;

        uint32_t data_size = FORMAT_LENGTH + subchunks_length;
        uint32_t encoded_size = HEADER_LENGTH + subchunks_length;
        // 1046524

        uint8_t *bytes = new uint8_t[encoded_size];

        if (bytes == 0)
                return 0;

        uint32_t offset = 0;

        memcpy(bytes + offset, Wave::id.c_str(), ID_LENGTH);
        offset += ID_LENGTH;

        memcpy(bytes + offset, &data_size, SIZE_LENGTH);
```

```
        offset += SIZE_LENGTH;

        memcpy(bytes + offset, "WAVE", FORMAT_LENGTH);
        offset += FORMAT_LENGTH;

        for (unsigned int i = 0; i < encoded.size(); i++)
        {
                memcpy(bytes + offset,
                        encoded[i]->bytes(), encoded[i]->size());
                offset += encoded[i]->size();
        }

        for (unsigned int i = 0; i < encoded.size(); i++)
        {
                delete encoded[i];
        }

        Data *data = new Data();

        if (data == 0)
        {
                delete [] bytes;
                return 0;
        }

        data->set(bytes, encoded_size);

        return data;
}

void
Wave::save(const string& path) const
{
        ofstream file(path.c_str());

        Data *encoded = encode();

        file.write((char *)encoded->bytes(), encoded->size());
```

```
        file.close();
}

void
Wave::add_chunk(Chunk *chunk)
{
        m_subchunks.push_back(chunk);
}

Format *
Wave::format() const
{
        int chunks = m_subchunks.size();
        Format *format = 0;

        for (int i = 0; i < chunks; i++)
        {
                format = dynamic_cast<Format *>(m_subchunks[i]);

                if (format != 0)
                        return format;
        }

        return format;
}

Generic *
Wave::data() const
{
        int chunks = m_subchunks.size();
        Generic *data = 0;

        for (int i = 0; i < chunks; i++)
        {
                data = dynamic_cast<Generic *>(m_subchunks[i]);

                if (data && data->id() == "data")
                        break;
        }
```

```
        return data;
}

SDL_AudioSpec
Wave::spec() const
{
        Format *fmt = format();
        SDL_AudioSpec spec;

        spec.freq = fmt->sampleRate();
        spec.channels = fmt->numChannels();
        spec.samples = 4096;
        spec.size = 0;

        if (fmt->bitsPerSample() == 8)
                spec.format = AUDIO_U8;
        else
                spec.format = AUDIO_S16;

        return spec;
}

SDL_AudioSpec* Load_Wave(const char *file,
        SDL_AudioSpec *spec, uint8_t **audio_buf,
        uint32_t *audio_len)
{
        Wave *wave = Wave::load(file);
        SDL_AudioSpec s = wave->spec();

        spec->format = s.format;
        spec->freq = s.freq;
        spec->channels = s.channels;
        spec->samples = 4096;
        spec->size = 0;

        Generic *data = wave->data();

        *audio_len = data->size();
```

```
        *audio_buf = (uint8_t *) malloc(*audio_len);

        memcpy(*audio_buf, data->data(), *audio_len);

        return spec;
}

vector<Chunk *>
Wave::subchunks() const
{
        return m_subchunks;
}
```

```
OBJECTS = data.o generic.o sound.o wave.o \
                format.o chunkfactory.o \
                meta.o lgmk.o

INCLUDES = include/data.h include/chunk.h \
                include/generic.h include/sound.h \
                include/wave.h include/format.h \
                include/chunkfactory.h \
                include/meta.h include/lgmk.h

all:
        @make libcore.a

libcore.a: $(INCLUDES) $(OBJECTS)
        ar rcs $@ $(OBJECTS)

%.o: %.cpp
        g++ -c -o $@ -W -Wall -pedantic -ansi -I./include $<

clean:
        @rm -rf *.o *.a *~
```

ANEXO H – Código Fonte do Tocador de *Audiobook*

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Player</class>
 <widget class="QMainWindow" name="Player">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>428</width>
    <height>445</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Player</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <layout class="QFormLayout" name="formLayout">
      <property name="fieldGrowthPolicy">
       <enum>QFormLayout::AllNonFixedFieldsGrow</enum>
      </property>
      <item row="0" column="0">
       <widget class="QLabel" name="staticTitleLabel">
        <property name="text">
         <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;
                 p&gt;&lt;span style=&quot; font-size:10pt;
                 font-weight:600; font-style:italic;&quot;
                 &gt;Title:&lt;/span&gt;&lt;/p&gt;&lt;
                 /body&gt;&lt;/html&gt;</string>
        </property>
        <property name="alignment">
         <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter</set>
```

```
   </property>
  </widget>
 </item>
 <item row="0" column="1">
  <widget class="QLabel" name="titleLabel">
   <property name="text">
    <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;
            TitleLabel&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
   </property>
  </widget>
 </item>
 <item row="1" column="0">
  <widget class="QLabel" name="staticAutorLabel">
   <property name="text">
    <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;
            p&gt;&lt;span style=&quot; font-size:10pt;
            font-weight:600;&quot;&gt;Author:&lt;/span&gt;
            &lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
   </property>
  </widget>
 </item>
 <item row="1" column="1">
  <widget class="QLabel" name="authorLabel">
   <property name="text">
    <string>AuthorLabel</string>
   </property>
  </widget>
 </item>
 <item row="2" column="0">
  <widget class="QLabel" name="staticLanguageLabel">
   <property name="text">
    <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;
            p&gt;&lt;span style=&quot;
            font-size:10pt; font-weight:600;&quot;&gt;
            Language:&lt;/span&gt;&lt;/p&gt;&lt;
            /body&gt;&lt;/html&gt;</string>
   </property>
  </widget>
 </item>
```

```
<item row="2" column="1">
 <widget class="QLabel" name="languageLabel">
  <property name="text">
   <string>LanguageLabel</string>
  </property>
 </widget>
</item>
<item row="3" column="0">
 <widget class="QLabel" name="staticPublisherLabel">
  <property name="text">
   <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;
           p&gt;&lt;span style=&quot; font-size:10pt;
           font-weight:600;&quot;&gt;Publisher:&lt;/span&gt;
           &lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
  </property>
 </widget>
</item>
<item row="3" column="1">
 <widget class="QLabel" name="publisherLabel">
  <property name="text">
   <string>PublisherLabel</string>
  </property>
 </widget>
</item>
<item row="4" column="0">
 <widget class="QLabel" name="staticAddressLabel">
  <property name="text">
   <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;
           p&gt;&lt;span style=&quot; font-size:10pt;
           font-weight:600;&quot;&gt;Address:&lt;/span&gt;
           &lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
  </property>
 </widget>
</item>
<item row="4" column="1">
 <widget class="QLabel" name="addressLabel">
  <property name="text">
   <string>AddressLabel</string>
  </property>
```

```
   </widget>
  </item>
  <item row="5" column="0">
   <widget class="QLabel" name="staticPagesLabel">
    <property name="text">
     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;
            &lt;span style=&quot; font-size:10pt;
            font-weight:600;&quot;&gt;Pages:&lt;/span&gt;&lt;
            /p&gt;&lt;/body&gt;&lt;/html&gt;</string>
    </property>
   </widget>
  </item>
  <item row="5" column="1">
   <widget class="QLabel" name="pagesLabel">
    <property name="text">
     <string>PagesLabel</string>
    </property>
   </widget>
  </item>
  <item row="6" column="0">
   <widget class="QLabel" name="staticYearLabel">
    <property name="text">
     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;
            &lt;span style=&quot; font-size:10pt;
            font-weight:600;&quot;&gt;Year:&lt;/span&gt;
            &lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
    </property>
   </widget>
  </item>
  <item row="6" column="1">
   <widget class="QLabel" name="yearLabel">
    <property name="text">
     <string>YearLabel</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
```

```
 <widget class="QLabel" name="levelLabel">
  <property name="text">
   <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;
           &lt;p align=&quot;center&quot;&gt;
           Level&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
  </property>
 </widget>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
   <widget class="QPushButton" name="fastrewindButton">
    <property name="text">
     <string/>
    </property>
    <property name="icon">
     <iconset resource="resource.qrc">
      <normaloff>:/images/seek-backward.png
           </normaloff>:/images/seek-backward.png</iconset>
    </property>
    <property name="iconSize">
     <size>
      <width>30</width>
      <height>30</height>
     </size>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="rewindButton">
    <property name="text">
     <string/>
    </property>
    <property name="icon">
     <iconset resource="resource.qrc">
      <normaloff>:/images/backward.png
           </normaloff>:/images/backward.png</iconset>
    </property>
    <property name="iconSize">
```

```
    <size>
     <width>46</width>
     <height>46</height>
    </size>
   </property>
  </widget>
 </item>
 <item>
  <widget class="QPushButton" name="playOrPauseButton">
   <property name="text">
    <string/>
   </property>
   <property name="icon">
    <iconset resource="resource.qrc">
     <normaloff>:/images/start.png
              </normaloff>:/images/start.png</iconset>
   </property>
   <property name="iconSize">
    <size>
     <width>69</width>
     <height>69</height>
    </size>
   </property>
  </widget>
 </item>
 <item>
  <widget class="QPushButton" name="forwardButton">
   <property name="text">
    <string/>
   </property>
   <property name="icon">
    <iconset resource="resource.qrc">
     <normaloff>:/images/forward.png
              </normaloff>:/images/forward.png</iconset>
   </property>
   <property name="iconSize">
    <size>
     <width>46</width>
     <height>46</height>
```

```
      </size>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QPushButton" name="fastforwardButton">
     <property name="text">
      <string/>
     </property>
     <property name="icon">
      <iconset resource="resource.qrc">
       <normaloff>:/images/seek-forward.png
              </normaloff>:/images/seek-forward.png</iconset>
     </property>
     <property name="iconSize">
      <size>
       <width>30</width>
       <height>30</height>
      </size>
     </property>
    </widget>
   </item>
  </layout>
 </item>
 <item>
  <layout class="QVBoxLayout" name="verticalLayout_2">
   <item>
    <layout class="QHBoxLayout" name="horizontalLayout_2">
     <item>
      <widget class="QPushButton" name="upLevelButton">
       <property name="text">
        <string/>
       </property>
       <property name="icon">
        <iconset resource="resource.qrc">
         <normaloff>:/images/up.png
                </normaloff>:/images/up.png</iconset>
       </property>
      </widget>
```

```
      </item>
      <item>
       <widget class="QPushButton" name="downLevelButton">
        <property name="text">
         <string/>
        </property>
        <property name="icon">
         <iconset resource="resource.qrc">
          <normaloff>:/images/down.png
                  </normaloff>:/images/down.png</iconset>
        </property>
       </widget>
      </item>
     </layout>
    </item>
    <item>
     <widget class="QLabel" name="markLabel">
      <property name="text">
       <string>&lt;html&gt;&lt;head/&gt;&lt;
               body&gt;&lt;p align=&quot;center&quot;&gt;Mark&lt;
               /p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QLabel" name="subMarkLabel">
      <property name="text">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;
               &lt;p align=&quot;center&quot;&gt;
               SubMark&lt;/p&gt;&lt;
               /body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
    </item>
   </layout>
  </item>
  <item>
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="2">
```

```
      <widget class="QSlider" name="songSlider">
       <property name="orientation">
        <enum>Qt::Horizontal</enum>
       </property>
      </widget>
     </item>
     <item row="1" column="2">
      <widget class="QLCDNumber" name="lcdNumber"/>
     </item>
    </layout>
   </item>
  </layout>
 </widget>
 <widget class="QMenuBar" name="menuBar">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>428</width>
    <height>20</height>
   </rect>
  </property>
 </widget>
 <widget class="QStatusBar" name="statusBar"/>
 <widget class="QToolBar" name="toolBar">
  <property name="windowTitle">
   <string>toolBar</string>
  </property>
  <attribute name="toolBarArea">
   <enum>TopToolBarArea</enum>
  </attribute>
  <attribute name="toolBarBreak">
   <bool>false</bool>
  </attribute>
 </widget>
 <action name="actionOpen">
  <property name="text">
   <string>Open File</string>
  </property>
```

```
 </action>
 <action name="actionSave">
  <property name="text">
   <string>Close</string>
  </property>
 </action>
 <action name="actionSave_As">
  <property name="text">
   <string>Save</string>
  </property>
 </action>
 <action name="actionSave_As_2">
  <property name="text">
   <string>Save As...</string>
  </property>
 </action>
 <action name="actionRecent_Files">
  <property name="text">
   <string>Recent Files</string>
  </property>
 </action>
 <action name="actionCopy">
  <property name="text">
   <string>Copy</string>
  </property>
 </action>
 <action name="actionPaste">
  <property name="text">
   <string>Paste</string>
  </property>
 </action>
 <action name="actionThis_Player">
  <property name="text">
   <string>This Player</string>
  </property>
 </action>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources>
```

```
  <include location="resource.qrc"/>
 </resources>
 <connections/>
</ui>
```

```
#############################################################################
# Automatically generated by qmake (2.01a) sex nov 22 22:12:57 2013
#############################################################################
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += . ../Core/include
LIBS += -lSDL -L../Core -lcore

# Input
HEADERS += player.h soundCTRL.h sounddevice.h ui_meta.h
FORMS += player.ui
SOURCES += main.cpp player.cpp soundCTRL.cpp sounddevice.cpp ui_meta.cpp
RESOURCES += resource.qrc
```

```
#############################################################################
# Makefile for building: Player
# Generated by qmake (2.01a) (Qt 4.8.3) on: qua nov 27 20:23:05 2013
# Project:  Player.pro
# Template: app
# Command: /usr/bin/qmake -o Makefile Player.pro
#############################################################################

####### Compiler, tools and options

CC            = gcc
CXX           = g++
DEFINES       = -DQT_WEBKIT -DQT_NO_DEBUG -DQT_GUI_LIB \
                -DQT_CORE_LIB -DQT_SHARED
CFLAGS        = -m64 -pipe -O2 -Wall -W -D_REENTRANT $(DEFINES)
CXXFLAGS      = -m64 -pipe -O2 -Wall -W -D_REENTRANT $(DEFINES)
INCPATH       = -I/usr/share/qt4/mkspecs/linux-g++-64 -I. \
                -I/usr/include/qt4/QtCore \
                -I/usr/include/qt4/QtGui \
                -I/usr/include/qt4 -I. \
                -I../Core/include -I. -I.
LINK          = g++
LFLAGS        = -m64 -Wl,-O1
LIBS          = $(SUBLIBS)  -L/usr/lib/x86_64-linux-gnu -lSDL \
                -L../Core -lcore -lQtGui -lQtCore -lpthread
AR            = ar cqs
RANLIB        =
QMAKE         = /usr/bin/qmake
TAR           = tar -cf
COMPRESS      = gzip -9f
COPY          = cp -f
SED           = sed
COPY_FILE     = $(COPY)
COPY_DIR      = $(COPY) -r
STRIP         = strip
INSTALL_FILE  = install -m 644 -p
INSTALL_DIR   = $(COPY_DIR)
INSTALL_PROGRAM = install -m 755 -p
DEL_FILE      = rm -f
```

```
SYMLINK        = ln -f -s
DEL_DIR        = rmdir
MOVE           = mv -f
CHK_DIR_EXISTS= test -d
MKDIR          = mkdir -p

####### Output directory

OBJECTS_DIR    = ./

####### Files

SOURCES        = main.cpp \
                 player.cpp \
                 soundCTRL.cpp \
                 sounddevice.cpp \
                 ui_meta.cpp moc_player.cpp \
                 moc_soundCTRL.cpp \
                 moc_ui_meta.cpp \
                 qrc_resource.cpp
OBJECTS        = main.o \
                 player.o \
                 soundCTRL.o \
                 sounddevice.o \
                 ui_meta.o \
                 moc_player.o \
                 moc_soundCTRL.o \
                 moc_ui_meta.o \
                 qrc_resource.o
DIST           = /usr/share/qt4/mkspecs/common/unix.conf \
        /usr/share/qt4/mkspecs/common/linux.conf \
        /usr/share/qt4/mkspecs/common/gcc-base.conf \
        /usr/share/qt4/mkspecs/common/gcc-base-unix.conf \
        /usr/share/qt4/mkspecs/common/g++-base.conf \
        /usr/share/qt4/mkspecs/common/g++-unix.conf \
        /usr/share/qt4/mkspecs/qconfig.pri \
        /usr/share/qt4/mkspecs/modules/qt_webkit_version.pri \
        /usr/share/qt4/mkspecs/features/qt_functions.prf \
        /usr/share/qt4/mkspecs/features/qt_config.prf \
```

```
        /usr/share/qt4/mkspecs/features/exclusive_builds.prf \
        /usr/share/qt4/mkspecs/features/default_pre.prf \
        /usr/share/qt4/mkspecs/features/release.prf \
        /usr/share/qt4/mkspecs/features/default_post.prf \
        /usr/share/qt4/mkspecs/features/unix/gdb_dwarf_index.prf \
        /usr/share/qt4/mkspecs/features/warn_on.prf \
        /usr/share/qt4/mkspecs/features/qt.prf \
        /usr/share/qt4/mkspecs/features/unix/thread.prf \
        /usr/share/qt4/mkspecs/features/moc.prf \
        /usr/share/qt4/mkspecs/features/resources.prf \
        /usr/share/qt4/mkspecs/features/uic.prf \
        /usr/share/qt4/mkspecs/features/yacc.prf \
        /usr/share/qt4/mkspecs/features/lex.prf \
        /usr/share/qt4/mkspecs/features/include_source_dir.prf \
        Player.pro
QMAKE_TARGET  = Player
DESTDIR       =
TARGET        = Player

first: all
####### Implicit rules

.SUFFIXES: .o .c .cpp .cc .cxx .C

.cpp.o:
        $(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"

.cc.o:
        $(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"

.cxx.o:
        $(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"

.C.o:
        $(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"

.c.o:
        $(CC) -c $(CFLAGS) $(INCPATH) -o "$@" "$<"
```

```
####### Build rules

all: Makefile $(TARGET)

$(TARGET): ui_player.h $(OBJECTS)
	$(LINK) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(OBJCOMP) $(LIBS)

Makefile: \
Player.pro /usr/share/qt4/mkspecs/linux-g++-64/qmake.conf \
/usr/share/qt4/mkspecs/common/unix.conf \
	/usr/share/qt4/mkspecs/common/linux.conf \
	/usr/share/qt4/mkspecs/common/gcc-base.conf \
	/usr/share/qt4/mkspecs/common/gcc-base-unix.conf \
	/usr/share/qt4/mkspecs/common/g++-base.conf \
	/usr/share/qt4/mkspecs/common/g++-unix.conf \
	/usr/share/qt4/mkspecs/qconfig.pri \
	/usr/share/qt4/mkspecs/modules/qt_webkit_version.pri \
	/usr/share/qt4/mkspecs/features/qt_functions.prf \
	/usr/share/qt4/mkspecs/features/qt_config.prf \
	/usr/share/qt4/mkspecs/features/exclusive_builds.prf \
	/usr/share/qt4/mkspecs/features/default_pre.prf \
	/usr/share/qt4/mkspecs/features/release.prf \
	/usr/share/qt4/mkspecs/features/default_post.prf \
	/usr/share/qt4/mkspecs/features/unix/gdb_dwarf_index.prf \
	/usr/share/qt4/mkspecs/features/warn_on.prf \
	/usr/share/qt4/mkspecs/features/qt.prf \
	/usr/share/qt4/mkspecs/features/unix/thread.prf \
	/usr/share/qt4/mkspecs/features/moc.prf \
	/usr/share/qt4/mkspecs/features/resources.prf \
	/usr/share/qt4/mkspecs/features/uic.prf \
	/usr/share/qt4/mkspecs/features/yacc.prf \
	/usr/share/qt4/mkspecs/features/lex.prf \
	/usr/share/qt4/mkspecs/features/include_source_dir.prf \
	/usr/lib/x86_64-linux-gnu/libQtGui.prl \
	/usr/lib/x86_64-linux-gnu/libQtCore.prl
	$(QMAKE) -o Makefile Player.pro
/usr/share/qt4/mkspecs/common/unix.conf:
/usr/share/qt4/mkspecs/common/linux.conf:
/usr/share/qt4/mkspecs/common/gcc-base.conf:
```

```
/usr/share/qt4/mkspecs/common/gcc-base-unix.conf:
/usr/share/qt4/mkspecs/common/g++-base.conf:
/usr/share/qt4/mkspecs/common/g++-unix.conf:
/usr/share/qt4/mkspecs/qconfig.pri:
/usr/share/qt4/mkspecs/modules/qt_webkit_version.pri:
/usr/share/qt4/mkspecs/features/qt_functions.prf:
/usr/share/qt4/mkspecs/features/qt_config.prf:
/usr/share/qt4/mkspecs/features/exclusive_builds.prf:
/usr/share/qt4/mkspecs/features/default_pre.prf:
/usr/share/qt4/mkspecs/features/release.prf:
/usr/share/qt4/mkspecs/features/default_post.prf:
/usr/share/qt4/mkspecs/features/unix/gdb_dwarf_index.prf:
/usr/share/qt4/mkspecs/features/warn_on.prf:
/usr/share/qt4/mkspecs/features/qt.prf:
/usr/share/qt4/mkspecs/features/unix/thread.prf:
/usr/share/qt4/mkspecs/features/moc.prf:
/usr/share/qt4/mkspecs/features/resources.prf:
/usr/share/qt4/mkspecs/features/uic.prf:
/usr/share/qt4/mkspecs/features/yacc.prf:
/usr/share/qt4/mkspecs/features/lex.prf:
/usr/share/qt4/mkspecs/features/include_source_dir.prf:
/usr/lib/x86_64-linux-gnu/libQtGui.prl:
/usr/lib/x86_64-linux-gnu/libQtCore.prl:
qmake:  FORCE
        @$(QMAKE) -o Makefile Player.pro

dist:
        @$(CHK_DIR_EXISTS) .tmp/Player1.0.0 || $(MKDIR) \
        .tmp/Player1.0.0
        $(COPY_FILE) --parents $(SOURCES) $(DIST) \
        .tmp/Player1.0.0/ && $(COPY_FILE) \
        --parents player.h soundCTRL.h sounddevice.h \
        ui_meta.h .tmp/Player1.0.0/ && $(COPY_FILE) \
        --parents resource.qrc .tmp/Player1.0.0/ && \
        $(COPY_FILE) --parents main.cpp player.cpp \
        soundCTRL.cpp sounddevice.cpp ui_meta.cpp \
        .tmp/Player1.0.0/ && $(COPY_FILE) --parents \
        player.ui .tmp/Player1.0.0/ && (cd `dirname \
        .tmp/Player1.0.0` && $(TAR) Player1.0.0.tar \
```

```
        Player1.0.0 && $(COMPRESS) Player1.0.0.tar) \
        && $(MOVE) `dirname \
        .tmp/Player1.0.0`/Player1.0.0.tar.gz \
        . && $(DEL_FILE) -r .tmp/Player1.0.0


clean:compiler_clean
        -$(DEL_FILE) $(OBJECTS)
        -$(DEL_FILE) *~ core *.core


####### Sub-libraries

distclean: clean
        -$(DEL_FILE) $(TARGET)
        -$(DEL_FILE) Makefile


check: first

mocclean: compiler_moc_header_clean \
                compiler_moc_source_clean

mocables: compiler_moc_header_make_all \
                compiler_moc_source_make_all

compiler_moc_header_make_all: moc_player.cpp \
                moc_soundCTRL.cpp moc_ui_meta.cpp
compiler_moc_header_clean:
        -$(DEL_FILE) moc_player.cpp moc_soundCTRL.cpp \
                moc_ui_meta.cpp
moc_player.cpp: player.h
        /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) \
        player.h -o moc_player.cpp

moc_soundCTRL.cpp: ui_meta.h \
                ui_player.h \
                soundCTRL.h
        /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) \
```

```
        soundCTRL.h -o moc_soundCTRL.cpp

moc_ui_meta.cpp: ui_player.h \
                 ui_meta.h
        /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) ui_meta.h \
        -o moc_ui_meta.cpp

compiler_rcc_make_all: qrc_resource.cpp
compiler_rcc_clean:
        -$(DEL_FILE) qrc_resource.cpp
qrc_resource.cpp: resource.qrc \
                 images/eject.png \
                 images/stop.png \
                 images/down.png \
                 images/up.png \
                 images/add.png \
                 images/forward.png \
                 images/start.png \
                 images/record.png \
                 images/remove.png \
                 images/seek-backward.png \
                 images/pause.png \
                 images/seek-forward.png \
                 images/backward.png
        /usr/bin/rcc -name resource resource.qrc -o qrc_resource.cpp

compiler_image_collection_make_all: qmake_image_collection.cpp
compiler_image_collection_clean:
        -$(DEL_FILE) qmake_image_collection.cpp
compiler_moc_source_make_all:
compiler_moc_source_clean:
compiler_uic_make_all: ui_player.h
compiler_uic_clean:
        -$(DEL_FILE) ui_player.h
ui_player.h: player.ui
        /usr/bin/uic-qt4 player.ui -o ui_player.h

compiler_yacc_decl_make_all:
compiler_yacc_decl_clean:
```

```
compiler_yacc_impl_make_all:
compiler_yacc_impl_clean:
compiler_lex_make_all:
compiler_lex_clean:
compiler_clean: compiler_moc_header_clean compiler_rcc_clean\
                compiler_uic_clean

####### Compile

main.o: main.cpp sounddevice.h \
                soundCTRL.h \
                ui_meta.h \
                ui_player.h \
                player.h
        $(CXX) -c $(CXXFLAGS) $(INCPATH) -o main.o main.cpp

player.o: player.cpp player.h \
                ui_player.h
        $(CXX) -c $(CXXFLAGS) $(INCPATH) -o player.o player.cpp

soundCTRL.o: soundCTRL.cpp soundCTRL.h \
                ui_meta.h \
                ui_player.h
        $(CXX) -c $(CXXFLAGS) $(INCPATH) \
                -o soundCTRL.o soundCTRL.cpp

sounddevice.o: sounddevice.cpp sounddevice.h \
                soundCTRL.h \
                ui_meta.h \
                ui_player.h
        $(CXX) -c $(CXXFLAGS) $(INCPATH) \
                -o sounddevice.o sounddevice.cpp

ui_meta.o: ui_meta.cpp ui_meta.h \
                ui_player.h
        $(CXX) -c $(CXXFLAGS) $(INCPATH) \
                -o ui_meta.o ui_meta.cpp

moc_player.o: moc_player.cpp
```

```
	$(CXX) -c $(CXXFLAGS) $(INCPATH) \
		-o moc_player.o moc_player.cpp

moc_soundCTRL.o: moc_soundCTRL.cpp
	$(CXX) -c $(CXXFLAGS) $(INCPATH) \
		-o moc_soundCTRL.o moc_soundCTRL.cpp

moc_ui_meta.o: moc_ui_meta.cpp
	$(CXX) -c $(CXXFLAGS) $(INCPATH) \
		-o moc_ui_meta.o moc_ui_meta.cpp

qrc_resource.o: qrc_resource.cpp
	$(CXX) -c $(CXXFLAGS) $(INCPATH) \
		-o qrc_resource.o qrc_resource.cpp

####### Install

install:   FORCE

uninstall:   FORCE

FORCE:
```

```
#include <iostream>

#include <SDL/SDL.h>
#include <QApplication>

#include "sounddevice.h"
//#include "sound.h"
#include "player.h"
#include "ui_meta.h"


using namespace std;

int main(int argc, char *argv[])
{
        if (argc != 2)
        {
                cerr << "Usage: " << argv[0] << " {rab_file.rab}\n";
                return -1;
        }

        char *path = argv[1];

        QApplication app(argc, argv);
        Player w;
        w.setFixedSize(w.size());

        Ui_meta *uim = new Ui_meta(path);
        uim->setupUi(&w);
        uim->setMetaLabels();
        uim->setMarksLabels();
        uim->initLCDDisplay();
        uim->initSlider();
        uim->connections();

        //-------------------------------------------------------------
        SoundCTRL *control = new SoundCTRL();
        control->setView(uim);
```

```
        Sound *sound = new Sound;
        SoundDevice *device = new SoundDevice;

        device->open(sound);
        device->openWAV(path);
        device->audioConverter();
        device->setSound(sound);

        control->setSound(sound);

        //-----------------------------------
        w.show();

        return app.exec();
}
```

```
#ifndef PLAYER_H
#define PLAYER_H

#include <QMainWindow>

namespace Ui {
class Player;
}

class Player : public QMainWindow
{
    Q_OBJECT

public:
    explicit Player(QWidget *parent = 0);
    ~Player();

private:
    Ui::Player *ui;
};

#endif // PLAYER_H
```

```
#include "player.h"
#include "ui_player.h"

Player::Player(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Player)
{
    ui->setupUi(this);
}

Player::~Player()
{
    delete ui;
}
```

```
#ifndef SOUNDCTRL_H
#define SOUNDCTRL_H

#include <QObject>
#include <QIcon>
#include <SDL/SDL.h>
#include <iostream>

//#include <thread>
#include <pthread.h>

#include "sound.h"
#include "ui_meta.h"
#include "lgmk.h"


using namespace std;

class SoundCTRL : public QObject
{
        Q_OBJECT

public slots:
        void playOrPauseCTRL();

        void forward(uint32_t timeInSeconds, Format *format);
        void rewind(uint32_t timeInSeconds, Format *format);
        void set_m_position(uint32_t timeInSeconds, Format *format);

        void fastForward(Format *format);
        void fastRewind(Format *format);



signals:
        void soundPlayed();

        void changeAllMarks();
```

```
        void clock(int time);


public:
        SoundCTRL();

        void setView(Ui_meta *view);
        void setSound(Sound *sound);

        static void callback(void *userdata, uint8_t *stream, int len);

        void changeLabels();
public:
        int playing;
        Ui_meta *view;
        Sound *sound;

private:
        int timeInSeconds;

};


#endif
```

```
#include "soundCTRL.h"

SoundCTRL *g_control = 0;

SoundCTRL::SoundCTRL()
{
        playing = 1;
        view = 0;
        sound = 0;

        timeInSeconds = 0;

        g_control = this;

}

void
SoundCTRL::setSound(Sound *sound)
{
        this->sound = sound;
}

void
SoundCTRL::setView(Ui_meta *view)
{
        this->view = view;
        connect(view, SIGNAL(playSound()), this,
                SLOT(playOrPauseCTRL()));

        connect(view, SIGNAL(pauseSound()), this,
                SLOT(playOrPauseCTRL()));

        connect(this, SIGNAL(soundPlayed()), this->view,
                SLOT(playOrPause()));
        connect(this, SIGNAL(soundPlayed()), this->view,
                SLOT(resetMarkLabels()));
        connect(this, SIGNAL(soundPlayed()), this->view,
                SLOT(resetClockAndSlider()));
```

```
        connect(this, SIGNAL(changeAllMarks()), this->view,
                SLOT(changeMarksLabels()));

        connect(view, SIGNAL(nextMark(uint32_t, Format*)),
                this, SLOT(forward(uint32_t, Format*)));
        connect(view, SIGNAL(prevMark(uint32_t, Format*)),
                this, SLOT(rewind(uint32_t, Format*)));

        connect(view, SIGNAL(forwardTime(Format *)), this,
                SLOT(fastForward(Format *)));
        connect(view, SIGNAL(rewindTime(Format *)), this,
                SLOT(fastRewind(Format *)));

        connect(this, SIGNAL(clock(int)), this->view,
                SLOT(updateClock(int)));

        connect(view, SIGNAL(update_m_position(uint32_t, Format*)),
                this, SLOT(set_m_position(uint32_t, Format*)));

}

void
SoundCTRL::playOrPauseCTRL()
{
        if (sound->m_position == -1)
                sound->m_position = 0;

        playing ^= 1;
        SDL_PauseAudio(playing);
}

void
SoundCTRL::callback(void *userdata, uint8_t *audio, int length)
{
        memset(audio, 0, length);

        Sound *sound = (Sound *) userdata;

        if (sound->m_position == -1)
```

```
                return;

        if (sound->m_position >= sound->size())
        {

                sound->m_position = -1;

                emit g_control->soundPlayed();

                return;
        }
        int nextSamplesLength;

        if (sound->m_position + length > sound->size())
                nextSamplesLength =  sound->size() - sound->m_position;
        else
                nextSamplesLength = length;

        SDL_MixAudio(audio, sound->buffer() +
                sound->m_position, nextSamplesLength,
                SDL_MIX_MAXVOLUME / 2);

        sound->m_position += nextSamplesLength;

        g_control->changeLabels();

}

void
SoundCTRL::fastForward(Format *format)
{
        if(sound->m_position <= 0)
        {
                sound->m_position = 0;
        }
        else
        {
                uint32_t position = 5 * format->numChannels()
                * format->sampleRate() * format->bitsPerSample()/8;
```

```
                sound->m_position += position;
        }
}

void
SoundCTRL::fastRewind(Format *format)
{
        if(sound->m_position <= 0 ||
                sound->m_position <= (int)
                (5 * format->numChannels() *
                format->sampleRate() *  format->bitsPerSample()/8))
        {
                sound->m_position = 0;
        }
        else
        {
                uint32_t position = 5 * format->numChannels()
                * format->sampleRate() * format->bitsPerSample()/8;
                * sound->m_position -= position;
        }
}

void
SoundCTRL::rewind(uint32_t timeInSeconds, Format *format)
{
        uint32_t position = timeInSeconds
        * format->numChannels() * format->sampleRate()
        * format->bitsPerSample()/8;

        sound->m_position = position;
}

void
SoundCTRL::forward(uint32_t timeInSeconds, Format *format)
{
        uint32_t position = timeInSeconds
        * format->numChannels() * format->sampleRate()
        * format->bitsPerSample()/8;
```

```
        sound->m_position = position;
}

void
SoundCTRL::set_m_position(uint32_t timeInSeconds, Format *format)
{
        uint32_t position = timeInSeconds *
                format->numChannels() *
                format->sampleRate() * format->bitsPerSample()/8;
        sound->m_position = position;
}

uint32_t update_mark(const vector<uint32_t>& marks, int timeInSeconds)
{

        for (int i = 1; i < (int) marks.size(); i++)
        {
                if ((int) marks[i] > timeInSeconds)
                        return i - 1;
        }

        return marks.size() - 1;
}

void
SoundCTRL::changeLabels()
{
        timeInSeconds = (8 * sound->m_position)/
                (view->getFormat()->numChannels() *
                view->getFormat()->sampleRate() *
                view->getFormat()->bitsPerSample());

        int nextMark = view->MarkIndex();
        int nextSubMark = view->SubMarkIndex();

        vector<uint32_t> marks = view->marks();
        vector<uint32_t> subMarks = view->subMarks();

        uint32_t now = update_mark(marks, timeInSeconds);
```

```
        if (now != (uint32_t) nextMark)
        {
                view->setMarkIndex(now);
        }

        now = update_mark(subMarks, timeInSeconds);

        if (now != (uint32_t) nextSubMark)
        {
                view->setSubMarkIndex(now);
        }

        emit changeAllMarks();
        emit clock((int) timeInSeconds);
}
```

```
#ifndef SOUNDDEVICE_H
#define SOUNDDEVICE_H

#include <SDL/SDL.h>
#include "soundCTRL.h"
#include "sound.h"

using namespace std;

class SoundDevice
{
public:
        SDL_AudioSpec desired() const;
        SDL_AudioSpec obtained() const;

        SDL_AudioSpec wavSpec() const;
        uint32_t wavLen() const;
        uint8_t *wavBuffer() const;

        void open(Sound *sound);
        void openWAV(char *path);
        void audioConverter();
        void setSound(Sound *sound);
private:
        SDL_AudioSpec m_desired;
        SDL_AudioSpec m_obtained;

        SDL_AudioSpec m_wavSpec;
        uint32_t m_wavLen;
        uint8_t *m_wavBuffer;

        SDL_AudioCVT m_cvt;
};

#endif
```

```
#include "sounddevice.h"

SDL_AudioSpec
SoundDevice::desired() const
{
        return m_desired;
}

SDL_AudioSpec
SoundDevice::obtained() const
{
        return m_obtained;
}

SDL_AudioSpec
SoundDevice::wavSpec() const
{
        return m_wavSpec;
}

uint32_t
SoundDevice::wavLen() const
{
        return m_wavLen;
}

uint8_t *
SoundDevice::wavBuffer() const
{
        return m_wavBuffer;
}

void
SoundDevice::open(Sound *sound)
{
        int rc = SDL_Init(SDL_INIT_AUDIO);

        if (rc != 0) {
                cerr << "Erro na inicializacao "
```

```
                        "do modulo de audio: " <<
                        SDL_GetError() << endl;
                return;
        }

        m_desired.freq = 44100;
        m_desired.format = AUDIO_S16;
        m_desired.channels = 2;
        m_desired.samples = 4096;
        m_desired.callback = SoundCTRL::callback;
        m_desired.userdata = (void *) sound;

        rc = SDL_OpenAudio(&m_desired, &m_obtained);

        if (rc != 0)
        {
                cerr << "Erro na abertura do "
                        "dispositivo de audio: " <<
                        SDL_GetError() << endl;
                SDL_Quit();
                return;
        }

}

void
SoundDevice::openWAV(char *path)
{
        if (SDL_LoadWAV(path, &m_wavSpec,
                &m_wavBuffer, &m_wavLen) == NULL)
        {
                cout << "Falha! " << SDL_GetError() << endl;
                SDL_CloseAudio();
                SDL_Quit();
                return;
        }

        //cout << "SDL_LoadWAV: Ok!" << endl;
}
```

```
void
SoundDevice::audioConverter()
{
        int rc = SDL_BuildAudioCVT(&m_cvt,
        m_wavSpec.format, m_wavSpec.channels,
        m_wavSpec.freq, m_obtained.format,
        m_obtained.channels, m_obtained.freq);

        if (rc != 0)
        {
                cout << "audioConverter(): Falha! "
                << SDL_GetError() << endl;
                SDL_FreeWAV(m_wavBuffer);
                SDL_Quit();

                return;
        }

        m_cvt.len = m_wavLen;
        uint8_t *wavNewBuf = (uint8_t *)
                malloc(m_cvt.len * m_cvt.len_mult);

        if (wavNewBuf == NULL)
        {
                cerr << "audioConverter(): "
                "Sem memoria para um novo buffer!" << endl;
                SDL_FreeWAV(m_wavBuffer);
                SDL_CloseAudio();
                SDL_Quit();

                return;
        }

        memcpy(wavNewBuf, m_wavBuffer, m_wavLen);
        m_cvt.buf = wavNewBuf;

        rc = SDL_ConvertAudio(&m_cvt);
```

```
        if (rc != 0)
        {
                cerr << "audioConverter(): "
                "Erro na conversao do audio!" << endl;
                SDL_FreeWAV(m_wavBuffer);
                free(wavNewBuf);
                SDL_CloseAudio();
                SDL_Quit();

                return;
        }

        SDL_FreeWAV(m_wavBuffer);

        //cout << "audioConverter(): Ok!" << endl;
}

void
SoundDevice::setSound(Sound *sound)
{
        SDL_LockAudio();

        sound->m_position = 0;
        sound->setBuffer(m_cvt.buf);
        sound->setSize(m_cvt.len * m_cvt.len_mult);

        SDL_UnlockAudio();

        //cout << "setSound: Ok!" << endl;
}
```

```
#ifndef UI_META_H
#define UI_META_H

#include <sstream>

#include "ui_player.h"

#include "generic.h"
#include "wave.h"
#include "meta.h"
#include "lgmk.h"


class Ui_meta : public QObject, public Ui_Player
{
        Q_OBJECT

public slots:
        void playOrPause();
        void resetMarkLabels();
        void resetClockAndSlider();

        void fastForward();
        void fastRewind();

        void next();
        void prev();

        void levelUp();
        void levelDown();

        void changeMarksLabels();

        void updateClock(int time);
        void change_song_position(int time);

signals:
        void playSound();
        void pauseSound();
```

```
        void nextMark(uint32_t timeInSeconds, Format *format);
        void prevMark(uint32_t timeInSeconds, Format *format);

        void forwardTime(Format *format);
        void rewindTime(Format *format);

        void remove();

        void update_m_position(uint32_t time, Format* format);

public:
        Ui_meta(const char *path);
        void setMetaLabels();
        void setMarksLabels();
        void initLCDDisplay();
        void initSlider();
        void connections();
        bool playing;

        void rewind();
        void forward();

        void forwardSubMark();
        void rewindSubMark();

        void synchronizeMarks();

        int MarkIndex() const;
        int SubMarkIndex() const;

        void setMarkIndex(int newMarkIndex);
        void setSubMarkIndex(int newSubMarkIndex);

        Lgmk *getLgmk() const;

        Format *getFormat() const;

        vector<uint32_t> marks() const;
```

```
        vector<uint32_t> subMarks() const;
private:
        int markIndex;
        int subMarkIndex;

        vector<uint32_t> m_marks;
        vector<string> m_marksNames;

        vector<uint32_t> m_subMarks;
        vector<string> m_subMarksNames;
        int level;

        Wave *wave;
        Meta *meta;
        Lgmk *lgmk;
        uint32_t duration;
        Format *format;

        QIcon icon;
        stringstream string_stream;
};


#endif
```

```
#include "ui_meta.h"

Ui_meta::Ui_meta(const char *path)
{
        playing = true;

        markIndex = 0;
        subMarkIndex = 0;
        level = 1;

        duration = 0;

        wave = Wave::load(path);

        vector<Chunk *> subchunks = wave->subchunks();

        for(vector<Chunk *>::iterator it =
                subchunks.begin(); it != subchunks.end(); it++)
        {
                lgmk = dynamic_cast<Lgmk *>(*it);

                if(lgmk != 0)
                        break;
        }

        for(vector<Chunk *>::iterator it =
                subchunks.begin(); it != subchunks.end(); it++)
        {
                Generic *data = dynamic_cast<Generic *>(*it);

                if(data != 0 && data->id() == "data")
                {
                        duration = data->size();
                        break;
                }

        }

        for(vector<Chunk *>::iterator it =
```

```
                subchunks.begin(); it != subchunks.end(); it++)
        {
                format = dynamic_cast<Format *>(*it);

                if(format != 0)
                        break;
        }

        duration /= (format->sampleRate()
                * format->numChannels());
                *
        duration /= (format->bitsPerSample()/8);

        if(lgmk == 0)
        {
                lgmk = new Lgmk;

                lgmk->add_mark(0);
                lgmk->add_markName("NONE");

                lgmk->add_mark(duration);
                lgmk->add_markName("NONE");

                lgmk->add_subMark(0);
                lgmk->add_subMarkName("NONE");

                lgmk->add_subMark(duration);
                lgmk->add_subMarkName("NONE");
        }

        m_marks = lgmk->marks();
        m_marksNames = lgmk->marksNames();

        m_subMarks = lgmk->subMarks();
        m_subMarksNames = lgmk->subMarksNames();
}

void
Ui_meta::setMetaLabels()
```

```
{

        vector<Chunk *> subchunks = wave->subchunks();

        for(vector<Chunk *>::iterator it =
                subchunks.begin(); it != subchunks.end(); it++)
        {
                meta = dynamic_cast<Meta *>(*it);

                if(meta != 0)
                        break;
        }

        if(meta == 0)
        {
                meta = new Meta;
        }

        QString title = QString::fromStdString(meta->title());
        QString author = QString::fromStdString(meta->author());
        QString language = QString::fromStdString(meta->language());
        QString publisher = QString::fromStdString(meta->publisher());
        QString year = QString::fromStdString(meta->year());
        QString address = QString::fromStdString(meta->address());
        QString pages = QString::fromStdString(meta->pages());

        titleLabel->setText(title);
        authorLabel->setText(author);
        languageLabel->setText(language);
        publisherLabel->setText(publisher);
        yearLabel->setText(year);
        addressLabel->setText(address);
        pagesLabel->setText(pages);
}

void
Ui_meta::setMarksLabels()
{
        QString firstMark = QString::fromStdString(m_marksNames[0]);
```

```
        markLabel->setText(firstMark);
        markLabel->setAlignment(Qt::AlignCenter);

        QString firstSubMark =
                QString::fromStdString(m_subMarksNames[0]);
        subMarkLabel->setText(firstSubMark);
        subMarkLabel->setAlignment(Qt::AlignCenter);

        QString level = QString::fromStdString("Level 1");
        levelLabel->setText(level);
        levelLabel->setAlignment(Qt::AlignCenter);
}

void
Ui_meta::initLCDDisplay()
{
        lcdNumber->setNumDigits(7);
        lcdNumber->setPalette(Qt::black);
        lcdNumber->display(QString::fromStdString("0.00:00"));
}

void
Ui_meta::initSlider()
{
        songSlider->setRange(0, duration);
}

void
Ui_meta::connections()
{
        connect(playOrPauseButton, SIGNAL(clicked()),
                this, SLOT(playOrPause()));

        connect(forwardButton,SIGNAL(clicked()),
                this, SLOT(next()));
        connect(rewindButton, SIGNAL(clicked()),
                this, SLOT(prev()));

        connect(upLevelButton, SIGNAL(clicked()),
```

```
                this, SLOT(levelUp()));
        connect(downLevelButton, SIGNAL(clicked()),
                this, SLOT(levelDown()));

        connect(fastforwardButton, SIGNAL(pressed()),
                this, SLOT(fastForward()));

        connect(fastrewindButton, SIGNAL(pressed()) ,
                this, SLOT(fastRewind()));

        connect(songSlider, SIGNAL(valueChanged(int)) ,
                this, SLOT(updateClock(int)));
        connect(songSlider, SIGNAL(valueChanged(int)) ,
                this, SLOT(change_song_position(int)));
}

void
Ui_meta::playOrPause()
{
        if(playing)
        {
                icon.addFile(QString::
                fromUtf8(":/images/pause.png"), QSize(),
                                QIcon::Normal, QIcon::Off);

                emit pauseSound();
        }
        else
        {
                icon.addFile(QString::
                fromUtf8(":/images/start.png"), QSize(),
                                QIcon::Normal, QIcon::Off);

                emit playSound();
        }

        playOrPauseButton->setIcon(icon);
        playing = !playing;
}
```

```
void
Ui_meta::resetMarkLabels()
{
        markIndex = 0;
        subMarkIndex = 0;

        QString firstMark = QString::
                fromStdString(m_marksNames[0]);
        markLabel->setText(firstMark);
        markLabel->setAlignment(Qt::AlignCenter);

        QString firstSubMark  = QString::
                fromStdString(m_subMarksNames[0]);

        subMarkLabel->setText(firstSubMark);
        subMarkLabel->setAlignment(Qt::AlignCenter);
}

void
Ui_meta::resetClockAndSlider()
{
        songSlider->setValue(0);
        lcdNumber->setPalette(Qt::black);
        lcdNumber->display(QString::fromStdString("0.00:00"));
}

void
Ui_meta::levelUp()
{
        level = 1;

        QString nivelLabel = QString::fromStdString("Level 1");
        levelLabel->setText(nivelLabel);
}

void
Ui_meta::levelDown()
{
```

```
        level = 2;

        QString nivelLabel = QString::fromStdString("Level 2");
        levelLabel->setText(nivelLabel);
}


void
Ui_meta::fastForward()
{
        emit forwardTime(format);
}


void
Ui_meta::fastRewind()
{
        emit rewindTime(format);
}

void
Ui_meta::next()
{
        if(level == 1)
        {
                forward();
        }else if(level == 2)
        {
                forwardSubMark();
        }

        synchronizeMarks();
}

void
Ui_meta::prev()
{
        if(level == 1)
        {
```

```
                rewind();
        }else if(level == 2)
        {
                rewindSubMark();
        }

        synchronizeMarks();
}

void
Ui_meta::forward()
{
        markIndex++;

        if(markIndex >= (int) m_marks.size())
        {
                resetMarkLabels();

                emit nextMark(duration, format);

        }else
        {
                emit nextMark(m_marks[markIndex], format);
        }
}

void
Ui_meta::rewind()
{
        markIndex--;

        QString mark;

        if(markIndex < 0)
        {
                markIndex = 0;
                subMarkIndex = 0;

                emit prevMark(0, format);
```

```
        }else
        {
                emit prevMark(m_marks[markIndex], format);
        }
}

void
Ui_meta::forwardSubMark()
{
        subMarkIndex++;

        if(subMarkIndex >= (int) m_subMarks.size())
        {
                resetMarkLabels();

                emit nextMark(duration, format);

        }else
        {
                emit nextMark(m_subMarks[subMarkIndex], format);
        }
}

void
Ui_meta::rewindSubMark()
{
        subMarkIndex--;

        QString subMark;

        if(subMarkIndex < 0)
        {
                markIndex = 0;
                subMarkIndex = 0;

                emit prevMark(0, format);

        }else
        {
```

```
                emit prevMark(m_subMarks[subMarkIndex], format);
        }
}


void
Ui_meta::synchronizeMarks()
{
        uint32_t position;
        QString mark;
        QString subMark;

        if(level == 1)
        {
                position = m_marks[markIndex];

                for(uint32_t i = 0; i < m_subMarks.size(); i++)
                {
                        if(position == 0)
                        {
                                subMarkIndex = 0;
                                break;
                        }else if(m_subMarks[i] == position)
                        {
                                subMarkIndex = i;

                                break;
                        }else if(m_subMarks[i] > position)
                        {
                                subMarkIndex = i - 1;
                                break;
                        }
                }

                mark  = QString::fromStdString
                        (m_marksNames[markIndex]);
                subMark = QString::fromStdString
                        (m_subMarksNames[subMarkIndex]);
```

```
                markLabel->setText(mark);
                subMarkLabel->setText(subMark);

        }else if(level == 2)
        {
                position = m_subMarks[subMarkIndex];

                for(uint32_t i = 0; i < m_marks.size(); i++)
                {
                        if(position == 0)
                        {
                                markIndex = 0;
                                break;

                        }else if(m_marks[i] > position)
                        {
                                markIndex = i - 1;


                                break;
                        }else if(m_marks[i] == position )
                        {
                                markIndex = i;
                                break;
                        }
                }

                mark  = QString::fromStdString
                        (m_marksNames[markIndex]);
                subMark = QString::fromStdString
                        (m_subMarksNames[subMarkIndex]);

                markLabel->setText(mark);
                subMarkLabel->setText(subMark);
        }
}

int
Ui_meta::MarkIndex() const
```

```
{
        return markIndex;
}

int
Ui_meta::SubMarkIndex() const
{
        return subMarkIndex;
}

void
Ui_meta::setMarkIndex(int newMarkIndex)
{
        if(markIndex != newMarkIndex)
                markIndex = newMarkIndex;
}

void
Ui_meta::setSubMarkIndex(int newSubMarkIndex)
{
        if(subMarkIndex != newSubMarkIndex)
                subMarkIndex = newSubMarkIndex;
}

Lgmk *
Ui_meta::getLgmk() const
{
        return lgmk;
}

Format *
Ui_meta::getFormat() const
{
        return format;
}

vector<uint32_t>
Ui_meta::marks() const
{
```

```
        return m_marks;
}

vector<uint32_t>
Ui_meta::subMarks() const
{
        return m_subMarks;
}

void
Ui_meta::changeMarksLabels()
{
        QString mark;
        QString subMark;

        mark  = QString::fromStdString
                (m_marksNames[markIndex]);
        subMark = QString::fromStdString
                (m_subMarksNames[subMarkIndex]);

        markLabel->setText(mark);
        subMarkLabel->setText(subMark);
}

void
Ui_meta::updateClock(int time)
{
        int hours = time/3600;
        int minutes = time/60 - 60*hours;
        int seconds = time - 60*minutes - 3600*hours;

        string_stream << hours + '\0' << ".";

        if(minutes < 10)
        {
                string_stream << "0" << minutes + '\0' << ":";
        } else
        {
                string_stream << minutes + '\0' << ":";
```

```
        }

        if(seconds < 10)
        {
                string_stream << "0" << seconds + '\0';
        } else
        {
                string_stream << seconds + '\0';
        }

        lcdNumber->display(QString::
                fromStdString(string_stream.str()));
                songSlider->setValue(time);
}


void
Ui_meta::change_song_position(int time)
{
        emit update_m_position((uint32_t) time, format);
}
```

ANEXO I – Código Fonte do Editor de *Audiobook*

```
#include <SDL/SDL.h>
#include "wave.h"
#include "meta.h"
#include "lgmk.h"
#include "format.h"
#include "sound.h"

#include <iostream>
#include <string>
#include <vector>

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>


const char *
get_filename_ext(const char *filename)
{
        const char *dot = strrchr(filename, '.');

        if(!dot || dot == filename)
                return "";

        return dot + 1;
}

void
assert_file_extension(char *path)
{
        string file_ext(get_filename_ext(path));
        string wav_ext("wav");
        string rab_ext("rab");
```

```
        if(file_ext.compare(wav_ext) && file_ext.compare(rab_ext))
        {
                cout << "ERRO: O formato {" << file_ext<< "}
                                eh incompativel com esse programa!" << endl;

                return;
        }else
        {
                cout << file_ext <<": Extensao Correta!" << endl;
        }
}

void
insert_meta(Meta *meta)
{
        string input;

        cout << "Insira o titulo do audiobook: ";
        getline(cin, input);
        meta->setTitle(input);

        cout << "Insira o nome do autor do audiobook: ";
        getline(cin, input);
        meta->setAuthor(input);

        cout << "Insira o idioma do audiobook: ";
        getline(cin, input);
        meta->setLanguage(input);

        cout << "Insira o editor do audiobook: ";
        getline(cin, input);
        meta->setPublisher(input);

        cout << "Insira o ano de lancamento do audiobook: ";
        getline(cin, input);
        meta->setYear(input);

        cout << "Insira o local de origem do audiobook: ";
```

```
        getline(cin, input);
        meta->setAddress(input);

        cout << "Insira a quantidade de paginas do livro impresso: ";
        getline(cin, input);
        meta->setpages(input);
}

void
insert_level1_marks(Lgmk *lgmk, uint32_t duration)
{
        int count_marks = 1;
        string input;
        duration = 300;

        cout << "A primeira marcacao de NIVEL 1 inicia no tempo "
                        "igual a 0 segundos..." << endl;

        cout << "\tNIVEL 1" << endl;
        while(1)
        {
                cout << "\tMarcacao " << count_marks + 1 << ": "
                "Insira o tempo em segundos. "
                "(Digite ZERO para sair!): ";
                getline(cin, input);

                if(isdigit(*(input.c_str())))
                {
                        if(atoi(input.c_str()) == 0)
                        {
                                cout << "\t\tEdicao de marcacoes "
                                "de NIVEL 1 foi terminado!"
                                << endl;
                                cout << "\t\t"<< count_marks <<
                                " Marcacoes foram inseridas."
                                << endl;
                                lgmk->add_mark(0);
                                break;
                        }else if(atoi(input.c_str()) > (int) duration)
```

```
                        {
                                cout << "\t\tErro: Valor Incorreto!"
                                "O valor inserido esta fora do limite:"
                                "(0 - "<< duration << ")"
                                << endl;

                        }else if(atoi(input.c_str()) < 0)
                        {
                                cout << "\t\tErro: Valor Incorreto!"
                                " Valor negativo" << endl;

                        }else
                        {
                                lgmk->add_mark( atoi(input.c_str()) );
                                count_marks++;
                        }
                }
                else if(isalnum(*(input.c_str())))
                {
                        cout << "\t\tErro: Valor Incorreto! "
                        "Letras e numeros..." << endl;
                }
        }

        int i;
        for(i = 0; i < count_marks; i++)
        {
                cout << "Insira o nome da Marcacao " << i + 1 << ": ";
                getline(cin, input);
                lgmk->add_markName(input);
        }

        cout << "\t\tNIVEL 1.....ok!" << endl;
}

void
insert_level2_marks(Lgmk *lgmk, uint32_t duration)
{
        int count_submarks = 1;
```

```
string input;
duration = 300;

cout << "A primeira marcacao de NIVEL 2 inicia "
        "no tempo igual a 0 segundos..." << endl;

cout << "\tNIVEL 2" << endl;
while(1)
{
        cout << "\tSubmarcacao " << count_submarks + 1
                << ": Insira o tempo em segundos. "
                "(Digite ZERO para sair!): ";
        getline(cin, input);

        if(isdigit(*(input.c_str())))
        {
                if(atoi(input.c_str()) == 0)
                {
                        cout << "\t\tEdicao de marcacoes "
                        "de NIVEL 1 foi terminado!" << endl;
                        cout << "\t\t"<< count_submarks <<
                        " Marcacoes foram inseridas."
                        << endl;
                        lgmk->add_subMark(0);
                        break;
                }else if(atoi(input.c_str()) > (int) duration)
                {
                        cout << "\t\tErro: Valor Incorreto!"
                        "O valor inserido esta fora do "
                        "limite:(0 - "<< duration << ")"
                        << endl;

                }else if(atoi(input.c_str()) < 0)
                {
                        cout << "\t\tErro: Valor "
                        "Incorreto! Valor negativo" << endl;

                }else
                {
```

```
                                lgmk->add_subMark( atoi(input.c_str()) );
                                count_submarks++;
                        }

                }else if(isalnum(*(input.c_str())))
                {
                        cout << "\t\tErro: Valor Incorreto! Letras "
                        "e numeros..." << endl;
                }
        }

        int i;
        for(i = 0; i < count_submarks; i++)
        {
                cout << "Insira o nome da Submarcacao "
                        << i + 1 << ": ";
                getline(cin, input);
                lgmk->add_subMarkName(input);
        }

        cout << "\t\tNIVEL 2 ..... ok!" << endl;
}

int main(int argc, char *argv[])
{
        if (argc != 2)
        {
                cerr << "Usage: " << argv[0] <<
                        " {wave_file.wav}\n";
                return -1;
        }

        char *path = argv[1];

        assert_file_extension(path);

        Wave *wave = Wave::load(path);

        vector<Chunk *> subchunks = wave->subchunks();
```

```
        Format *format = new Format();
        for(vector<Chunk *>::iterator it =
                subchunks.begin(); it != subchunks.end(); it++)
        {
                format = dynamic_cast<Format *>(*it);

                if(format != 0)
                        break;
        }

        uint32_t duration = 0;
        duration /= (format->sampleRate() * format->numChannels());
        duration /= (format->bitsPerSample()/8);

        cout << duration << endl;

        Meta *meta = new Meta();
        Lgmk *lgmk = new Lgmk();

        insert_meta(meta);
        insert_level1_marks(lgmk, duration);
        insert_level2_marks(lgmk, duration);

        wave->add_chunk(meta);
        wave->add_chunk(lgmk);

        cout << "Digite o nome do arquivo de saida: ";
        string filename;
        getline(cin, filename);

        filename.append(".rab");
        cout << filename << endl;
        wave->save(filename);
}
```

```
CC = g++

CFLAGS = -O2 -W -Wall -pedantic -g  -ansi

LIBS = -lSDL -L../Core -lcore
INCLUDES = -I./include -I. -I./test -I../Core/include

HEADERS_DIR = include
SOURCES_DIR = src
OBJECTS_DIR = obj
TEST_DIR    = test
BIN_DIR     = bin
PLUGIN_DIR  = plugins

FILES =

TEST_FILES =

MAIN_FILE = main.cpp

HEADERS = $(addsuffix .h, $(addprefix $(HEADERS_DIR)/, \
                $(basename $(FILES))))
SOURCES = $(addprefix $(SOURCES_DIR)/, $(FILES))
OBJECTS = $(addsuffix .o, $(addprefix $(OBJECTS_DIR)/, \
                $(basename $(FILES))))

TEST_SOURCES = $(addprefix $(TEST_DIR)/, $(TEST_FILES))
TEST_OBJECTS = $(addsuffix .o, $(addprefix $(OBJECTS_DIR)/,\
                $(basename $(TEST_FILES))))

MAIN_SOURCE = $(addprefix $(SOURCES_DIR)/, $(MAIN_FILE))
MAIN_OBJECT = $(addprefix $(OBJECTS_DIR)/, \
                $(addsuffix .o, $(basename $(MAIN_FILE))))

.PHONY: all

all:
        @mkdir -p bin obj
        @make $(BIN_DIR)/editorLc
```

```
$(OBJECTS_DIR)/%.o: $(SOURCES_DIR)/%.cpp $(HEADERS)
        $(CC) $(CFLAGS) $(INCLUDES) -c $< -o $@

#$(OBJECTS_DIR)/%.o: $(TEST_DIR)/%.cpp $(HEADERS)
#       $(CC) $(CFLAGS) $(INCLUDES) -c $< -o $@

$(MAIN_OBJECT): $(MAIN_SOURCE) $(HEADERS)
        $(CC) $(CFLAGS) $(INCLUDES) -c $< -o $@

#$(TEST_DIR)/test: $(OBJECTS) $(TEST_OBJECTS)
#       $(CC) $(CFLAGS) $(INCLUDES) -o
#       $@ $(OBJECTS) $(TEST_OBJECTS) $(LIBS)

$(BIN_DIR)/editorLc: $(OBJECTS) $(MAIN_OBJECT)
        $(CC) $(CFLAGS) $(INCLUDES) -o \
        $@ $(OBJECTS) $(MAIN_OBJECT) $(LIBS)

clean:
        @find . -name '*~' -exec rm -rf {} \;
        @rm -rf $(OBJECTS_DIR)/*.o
        @rm -rf $(BIN_DIR) $(OBJECTS_DIR)

dist-clean: clean
        @rm -rf bin obj
```