**Sumário**

[1 Introdução 3](#_Toc259722067)

[2 Referencial Teórico 4](#_Toc259722068)

[3 Metodologia 5](#_Toc259722069)

[4 Ferramentas 6](#_Toc259722070)

[4.1 .NET Framework 6](#_Toc259722071)

[4.2 Visual studio 2008 6](#_Toc259722072)

[4.3 DirectX 6](#_Toc259722073)

[4.4 SlimDX 6](#_Toc259722074)

[4.5 Subversion 7](#_Toc259722075)

[4.6 UML 7](#_Toc259722076)

[4.7 NUnit 7](#_Toc259722077)

[4.8 TestMatrix 7](#_Toc259722078)

[5 Implementação 8](#_Toc259722079)

[5.1 Songer.SoundInput 9](#_Toc259722080)

[5.1.1 Classe SoundSource 9](#_Toc259722081)

[5.1.2 Classe LineInCapture 10](#_Toc259722082)

[5.1.3 Classe WaveFileCapture 11](#_Toc259722083)

[5.1.4 Classe SoundDetectedEventArgs 11](#_Toc259722084)

[5.2 Songer.SoundAnalysis 12](#_Toc259722085)

[5.2.1 Estrutura ComplexNumber 12](#_Toc259722086)

[5.2.2 Classe CooleyTukeyFFT 12](#_Toc259722087)

[5.3 Songer.MusicalInterpreter 13](#_Toc259722088)

[5.3.1 Classe MusicalNote 14](#_Toc259722089)

[5.3.2 Classe MusicalNoteDictionary 14](#_Toc259722090)

[5.3.3 Classe Chord 14](#_Toc259722091)

[5.3.4 Classe ChordDictionary 15](#_Toc259722092)

[5.3.5 Classe MusicalAnalyzer 15](#_Toc259722093)

[5.4 Songer.Database 15](#_Toc259722094)

[5.5 Songer.Presentation 15](#_Toc259722095)

[6 Experimentos 16](#_Toc259722096)

[7 Conclusão 17](#_Toc259722097)

# Introdução

# Referencial Teórico

# Metodologia

# Ferramentas

Este capitulo apresenta as ferramentas utilizadas na implementação do sistema proposto neste trabalho científico.

## .NET Framework

O .NET Framework é uma plataforma para desenvolvimento de aplicativos, que provê uma grande biblioteca para construção rápida de aplicativos (RAD), abstraindo diversos aspectos do sistema operacional, e uma máquina virtual para execução gerenciada dos mesmos. Permite que se desenvolva em quaisquer linguagens que possuam um compilador específico para a plataforma. Para este trabalho foi escolhida a linguagem C#, que é a mais utilizada na plataforma.

## Visual studio 2008

O Visual Studio é um ambiente integrado de desenvolvimento, que permite a criação de aplicativos para desktop (Windows) e web, tanto em código nativo (com linguagem C++) e para o .NET Framework.

## DirectX

O DirectX é uma coleção de APIs para tratar de tarefas relacionadas à multimídia no sistema operacional Windows, provendo uma abstração do hardware para chamadas de software, utilizado inicialmente para o desenvolvimento de jogos. Neste trabalho, foi utilizada para ter acesso a recursos de áudio, através das APIs do DirectSound.

## SlimDX

A SlimDX é uma biblioteca de código-aberto que disponibiliza acesso às APIs do DirectX através do .NET Framework. Foi criada após a descontinuação do Managed DirectX da Microsoft.

## Subversion

O Subversion é um sistema de controle de versão, utilizado para manter versões atuais e anteriores de arquivos como código-fonte e documentos.

## UML

A UML é uma linguagem de modelagem que permite que o desenvolvedor visualize o desenvolvimento do código de seu trabalho em diagramas padronizados. Para a criação dos diagramas de classe foi utilizado o próprio Visual Studio (ferramenta Class Diagram) e para os demais diagramas foi utilizado o Microsoft Visio.

## NUnit

O NUnit é uma plataforma de testes de unidade, composta de uma biblioteca de classes para desenvolvimento dos testes e aplicativos para execução dos mesmos. É utilizada para a validação de pequenas unidades de código-fonte.

## TestMatrix

É uma ferramenta que se integra ao Visual Studio e permite a execução dos testes de unidade dentro do mesmo, indicando quais testes executaram, se a execução ocorreu com sucesso ou não, e informando todas as linhas de código que a execução percorreu, facilitando a depuração e análise dos testes.

# Implementação

Este capítulo apresenta os conceitos e algoritmos utilizados na implementação do sistema proposto neste trabalho, bem como as classes pelas quais é composto o sistema. Os algoritmos foram desenvolvimentos, tanto o código como os comentários, utilizando somente palavras da língua inglesa, visando à internacionalização do código-fonte desenvolvido. No corpo desde capítulo será explicado o que cada um deles realiza.

Como mencionado no capítulo 3, o sistema foi divido em módulos para facilitar a implementação e futuro entendimento do mesmo. Como o Visual Studio possui o conceito de uma área de trabalho (conhecida como solução) composta de um ou mais projetos, cada modulo foi desenvolvido como um projeto separado dentro de uma única solução chamada Songer. A figura 5.1 apresenta a representação gráfica da solução, mostrando os módulos e seus namespaces[[1]](#footnote-2).

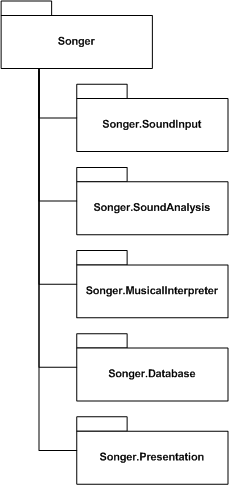


Figura 5.1: Diagrama dos módulos do projeto.

## Songer.SoundInput

Este módulo é responsável pela obtenção de dados referentes a sinais de áudio, para que estes possam ser posteriormente analisados pelo módulo Songer.SoundAnalysis. A figura 5.2 apresenta as classes que o compõe.

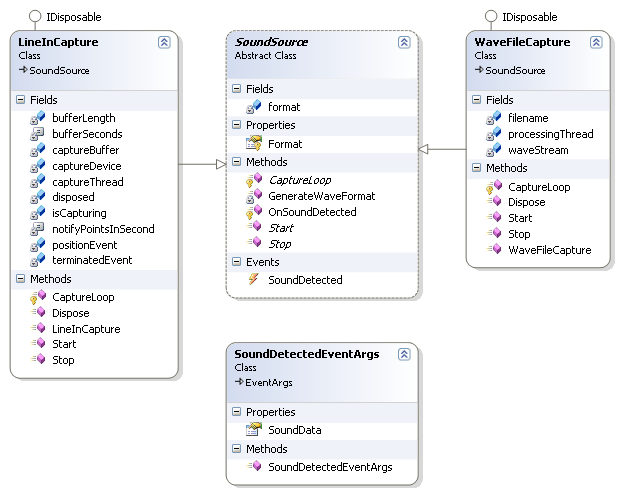


Figura 5.2: Classes do módulo Songer.SoundInput

Este módulo é composto por quatro classes, sendo uma abstrata que serve de modelo para implementação das classes de obtenção de dados, duas implementações de formas de obtenção de dados e uma classe para passar objetos em eventos gerados por estas classes.

### Classe SoundSource

Esta é uma classe abstrata, não sendo utilizada diretamente mas servindo de base às classes WaveFileCapture e LineInCapture. Ela é responsável por criar um modelo de funções que as classes que herdam dela devem implementar, além de implementar algumas funções essenciais, como por exemplo, fornecer o formato padrão de arquivo wave utilizado.

A função GenerateWaveFormat cria um instancia da classe WaveFormat, que contém as informações de formato do arquivo wave com o qual se está trabalhando. Para este trabalho, o formato é o mesmo utilizado em CD-Áudio, com 44100 amostragens de 16 bits por segundo (44100Hz/16-bit), mas com apenas um canal (mono), já que o instrumento utilizado nos testes (guitarra) fornece apenas um único canal de áudio.

A classe SoundSource fornece também um evento SoundDetected, que é disparado sempre que um conjunto de dados de som está disponível para ser processado.

Por fim, a classe fornece três métodos abstratos que todas as classes que herdam desta devem implementar: Start e Stop, responsáveis por iniciar e parar a captura de som; e CaptureLoop, executado como uma thread secundária, onde ocorre a leitura do buffer de som. Vale ressaltar que esta thread adicional é importante para que o sistema continue realizando a captura (principalmente quando de um instrumento musical) continuamente, mesmo quando os outros módulos estão processando as informações que já lhe foram passadas.

### Classe LineInCapture

A primeira classe que herda de SoundSource é a LineInCapture, responsável pela captura de som a partir de um dispositivo de entrada (por padrão, o line-in da placa de som). Para isso, foram utilizadas as APIs do DirectSound, em especial as classe DirectSoundCapture, que realiza a captura do som, e a classe CaptureBuffer, que armazena temporariamente a informação que foi capturada.

No método Start são criadas instancias destas duas classes, para se realizar a captura. Além disso, são criadas também posições de notificação (2 a cada segundo), que fará o papel de semáforo, permitindo que o buffer do som que já foi capturado seja lido rapidamente e repassado para o processamento nas classes de análise de sinal.

Em seguida, a classe inicializa a thread secundário, tendo o método CaptureLoop como código de execução. Nele o buffer de captura é inicializado e fica aguardando uma das mensagens de notificação para realizar a leitura. Quando isto ocorre, ele dispara o evento OnSoundDetected, passando os dados lidos do buffer para as classes inscritas neste evento, e em seguida volta a esperar a próxima notificação.

Por fim, o método Stop elimina a thread, fechando todos os objetos de captura.

### Classe WaveFileCapture

A outra classe que herda de SoundSource é a WaveFileCapture, que é responsável por ler a informação de áudio diretamente de um arquivo wave previamente gravado. A principal utilidade da mesma é para fazer a inserção de informações no banco de dados de forma mais rápida e automatizada, podendo o operador fazer a leitura de diversos arquivos ao invés de precisar “tocar” cada uma das músicas em um instrumento.

O seu construtor permite informar o nome do arquivo que será lido, sendo necessário que seja criado um novo objeto sempre que se quiser ler importar uma nova música ao banco, tendo em vista que após o processamento do arquivo a thread de captura é finalizada automaticamente. Para a leitura, foi utilizada a classe WaveStream (disponível no SlimDX), que facilita a leitura e interpretação de arquivos wave.

Nesta classe o método Start realiza apenas a criação da thread de captura. O buffer de captura e as notificações não são necessárias pois, diferente de uma entrada de linha, todos os dados do áudio já são conhecidos e finitos.

No loop de captura, é realizada a leitura do arquivo wave, enviando para processamento um segundo de áudio por vez, e saindo do loop (encerrando assim a thread) assim que for processado o último segundo de áudio do arquivo.

O método Stop desta classe aborta a thread imediamente, útil quando se quiser cancelar a leitura de um arquivo.

### Classe SoundDetectedEventArgs

A utilidade desta classe é fornecer um conjunto de dados relevantes ao som detectado e que serão disponibilizados à classe que se inscreveu ao evento SoundDetected da classe SoundSource quando o mesmo for disparado.

## Songer.SoundAnalysis

Este módulo é responsável pelo processamento do áudio capturado pelo Songer.SoundInput e geração do espectrograma do mesmo, que será analisado posteriormente pelo módulo Songer.MusicalInterpreter. A figura 5.3 apresenta as classes pertencentes a este módulo.

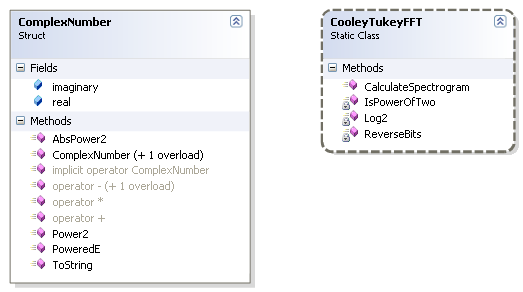


Figura 5.3: Classes do módulo Songer.SoundAnalysis

Este módulo é composto de apenas uma classe e uma estrutura de dados. A estrutura é uma implementação de números complexos e a classe é uma implementação do algoritmo Cooley-Tukey. Ambos são baseados nas implementações do artigo *FFT Guitar Tuner*.

### Estrutura ComplexNumber

Esta estrutura armazena um número complexo, tendo suporte a quase todas as operações básicas (soma, subtração e multiplicação), além de algumas adicionais, como potência de 2 (normal e absoluta) e exponenciação. Esta estrutura é necessário para o calculo da FFT.

### Classe CooleyTukeyFFT

Realiza a extração do espectrograma do som através do algoritmo FFT de Cooley-Tukey. Primeiramente se certifica de que o tamanho do vetor é uma potencia de dois (senão, adiciona itens zerados até que seja uma potencia de dois), em seguida gera um vetor inverso dos dados de som, calculando então a FFT. Com o resultado da FFT, cria-se um vetor to tipo Double[], onde cada item é uma representação de uma freqüência, e o seu valor é a amplitude do som referente àquela freqüência. Por fim, retorna este vetor para a função de origem.

## Songer.MusicalInterpreter

Este é o módulo principal do projeto, responsável pela interpretação do espectrograma (freqüência e amplitude dos sinais) e sua conversão em notas musicais (de acordo com a freqüência fundamental de cada nota). Além disso, este módulo também realiza a conversão do conjunto de notas em um determinado espaço de tempo para o acorde que as possui. A figura 5.4 apresenta as classes que compõem este módulo.

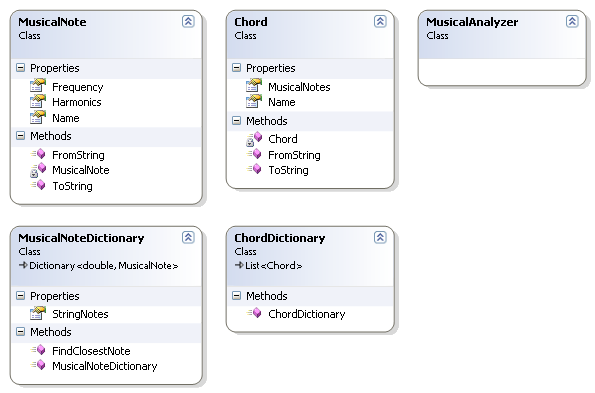


Figura 5.4: Classes do módulo Songer.MusicalInterpreter

O módulo é composto de cindo classes, sendo duas delas utilizadas para representação de notas e acordes (MusicalNote e Chord), outras duas para representar dicionários de notas e acordes (MusicalNoteDictionary e ChordDictionary), e a última responsável pela análise de espectrograma (MusicalAnalyzer).

### Classe MusicalNote

Esta classe representa uma nota musical, tendo o seu nome (no padrão <nota><oitava><sustenido?>), sua freqüência fundamental (que representa a nota), e os harmônicos que esta nota também gera. Para se criar um instancia dessa classe, todas estas informações devem ser passadas em forma de string, no padrão <nome nota>,<freqüência fundamental>,<harmônico 1>,...,<harmônico n>.

### Classe MusicalNoteDictionary

Esta classe representa o conjunto de todas as notas musicais audíveis, da oitava 0 (C0 ou Dó 0) até a oitava 10 (B10 ou Si 10). Ela o faz através da leitura do arquivo Notes.txt, que possui todas as informações de nota no padrão informado no parágrafo anterior, sendo uma nota por linha lida. Ao ler o arquivo, a classe armazena a posição da coleção onde estão armazenadas as informações das notas correspondentes às cordas soltas da guitarra, disponibilizando esta informação através da propriedade StringNotes.

O único método disponível nesta classe, FindClosestNote, é responsável por encontrar a nota mais próxima à freqüência informada. Ela é necessário pois nem todas as freqüências que espectrograma disponibilizou são absolutas em relação às notas musicas, além de a guitarra poder estar um pouco desafinada ou até mesmo a posição do dedo pode causar uma variação na casa de décimos de hertz, imperceptível ao ouvido humano, mas não para o computador, que é mais sensível a estas mudanças de freqüência.

### Classe Chord

Semelhante à classe MusicalNote, esta classe armazena as informações de um acorde, sendo elas o nome do mesmo (por exemplo Gm, para Sol Menor) e as notas que compõem o mesmo (de 3 a 6 notas).

Assim como a classe MusicalNote, esta classe é instanciada a partir de uma string com as informações do acorde, no padrão <nome> <6 5 4 3 2 1>, onde os números representam a posição do dedo no braço da guitarra para a formação de um acorde. A partir da nota da corda solta (como mencionado anteriormente, disponibilizada pela classe MusicalNoteDictionary), é possível somar o número de casas até e saber qual exatamente é a nota musical naquela casa específica.

### Classe ChordDictionary

Esta classe representa um dicionário de acordes, lido a partir do arquivo Chors.txt, contendo uma lista de strings no padrão de acorde informado no parágrafo anterior. Nenhuma propriedade ou método adicional é fornecido, somente a lista de acordes, que é a própria classe.

### Classe MusicalAnalyzer

## Songer.Database

Módulo de banco de dados

## Songer.Presentation

Módulo de apresentação

# Experimentos

# Conclusão

1. Em .NET Framework, Namespace é uma representação de pacote que pode conter outros namespaces, classes e estruturas. [↑](#footnote-ref-2)