ANNA CATARINA B. TAVELLA e DANILO J. S. BELLINI

ORFEU - EDITOR E SEQÜENCIADOR MUSICAL

Projeto de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção da graduação em Engenharia

Área de Graduação: Engenharia Elétrica com Ênfase em Computação e Sistemas Digitais

Orientador: Prof. Dr. João José Neto

São Paulo
2007

ANNA CATARINA B. TAVELLA e DANILO J. S. BELLINI

ORFEU - EDITOR E SEQÜENCIADOR MUSICAL

Projeto de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção da graduação em Engenharia

Área de Graduação: Engenharia Elétrica com Ênfase em Computação e Sistemas Digitais

Orientador: Prof. Dr. João José Neto

São Paulo
2007

RESUMO

Este trabalho consiste na concepção, especificação e implementação de um *software* editor musical que envolve diferentes tipos de notação incluindo a notação em partitura e compatível com formatos padrões de grandes acervos gratuitos. A conversão entre as diversas notações é feita através de algoritmos adaptativos, com o intuito de possibilitar que usuários iniciantes possam tocar suas partituras, e de agilizar o trabalho de transcritores profissionais. O trabalho também inclui um seqüenciador de alta fidelidade, responsável pelo controle de detalhes artísticos não explícitos na notação musical utilizada. O projeto visa a ser atualizado com um conjunto de recursos que possibilitem o acesso a todas as funcionalidades existentes neste *software* aos usuários portadores de deficiência visual.

ABSTRACT

This report consists on the conception, specification and implementation of a musical editor software that involves different kinds of notation including the score notation and is compatible with standards which have a great number of files available for free. The conversion tools are the adaptive algotithms, to enable beginner users to play their own scores and to make professional transcribers work faster. The report also includes a specific module used to control the artistic details that cannot be written at the graphic notation. This project aims to be atualized with a set of resources that give access of all of the software funcionalities to people who have visual handicap.

SUMÁRIO

LISTA DE FIGURAS

Página

LISTA DE TABELAS

Página

1 Objetivo do Projeto

Criar um textitsoftware editor de músicas utilizando diferentes sistemas de notação musical, incluindo a notação padrão (partitura), e compatível com formatos padrão de grande acervo gratuito. O projeto inclui uma conversão entre formas diferentes de notação musical, utilizando algoritmos adaptativos.

Também faz parte do escopo do projeto um seqüenciador de alta fidelidade que possui controle de detalhes artísticos não explícitos na notação gráfica utilizada.

2 Justificativa

Um software de edição de músicas pode focalizar o aspecto de impressão de notação gráfica musical ou o aspecto de seqüenciação de uma música. As alternativas atuais ou focam somente um dos aspectos, devido à ausência de reciprocidade completa entre a notação musical padrão e a sonoridade esperada pelo compositor, ou possuem uma interface que que deixa transparecer indevidamente informações específicas da forma interna de representação da seqüência (números, principalmente).

Representações alternativas da música e suas conversões podem ser utilizadas para facilitar o trabalho de usuários que queiram apenas visualizar o que está presente em uma música e também daqueles que desejam escrever uma música, e os dispensa de um domínio amplo dos conhecimentos musicais.

A compatibilidade com os programas atualmente utilizados é essencial para o aproveitamento do conteúdo público atualmente presente na *Internet*.

3 Funcionamento básico

O *software* Orfeu foi criado com o intuito de auxiliar músicos profissionais e amadores a editar músicas. A entrada de dados do usuário dá-se pelo teclado de computador ou por instrumento MIDI, este último limitado a acordes (o *software* não captura o ritmo), na primeira versão. O Orfeu também possui a opção de importar arquivos do GP4 (Guitar Pro 4). Quando um arquvo é importado, além de ser exibido no editor poderá também ser ouvido através da saída MIDI e/ou salvo no formato próprio de arquivo. Também é possível fazer modificações nas músicas importadas, ou seja, estas podem ser editadas como qualquer outra música criada no Orfeu. Depois de editada, a música pode ser salva em um formato de arquivo próprio ou exportada para MIDI, ou ainda ouvida através da saída MIDI. A figura 1 mostra o esquema de entradas/saídas do *software* Orfeu.

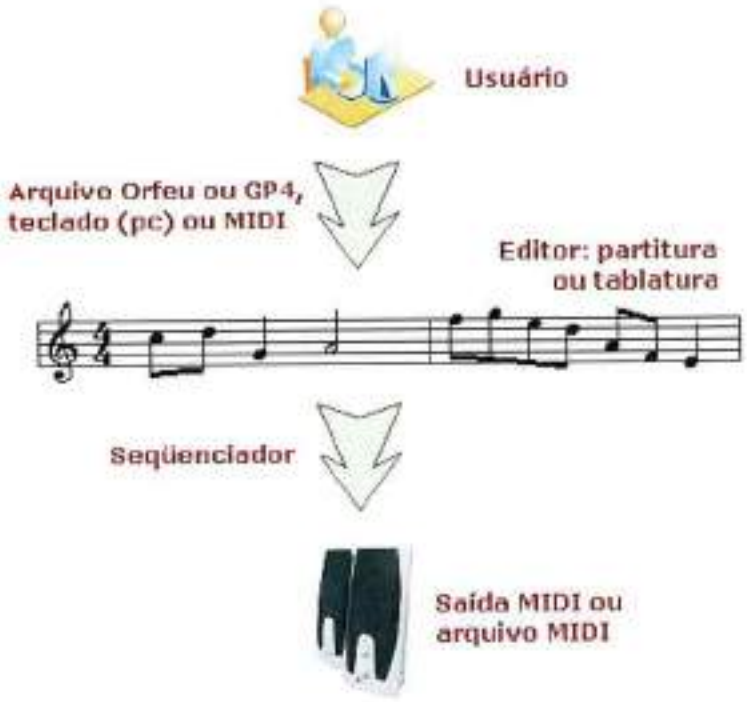

Figura 1: Esquema de entrada/saída

O editor possui dois modos de apresentação: o modo partitura e o modo tablatura. Apenas um modo é exibido por vez e pode ser selecionado pelo usuário, de acordo com sua preferência. As músicas geradas em ambos os editores são armazenadas no mesmo formato de arquivo. No caso de músicas editadas no editor de tablaturas, o arquivo apenas algumas informações a mais, comparado àquele gerado através do editor

de partituras, com o objetivo de indicar a corda em que a nota deverá ser tocada. O número de cordas e a afinação também ser configurados pelo usuário. O Orfeu ainda possui conversão entre os formatos partitura e tablatura. A conversão de tablatura para partitura é simples, pois é unívoca. Entretanto, o contrário não é verdadeiro, é possível que uma nota possua mais de uma opção de corda para ser tocada. Nesse caso, a conversão é adaptativa. A figura 2 mostra o esquema básico de funcionamento do *software* Orfeu.

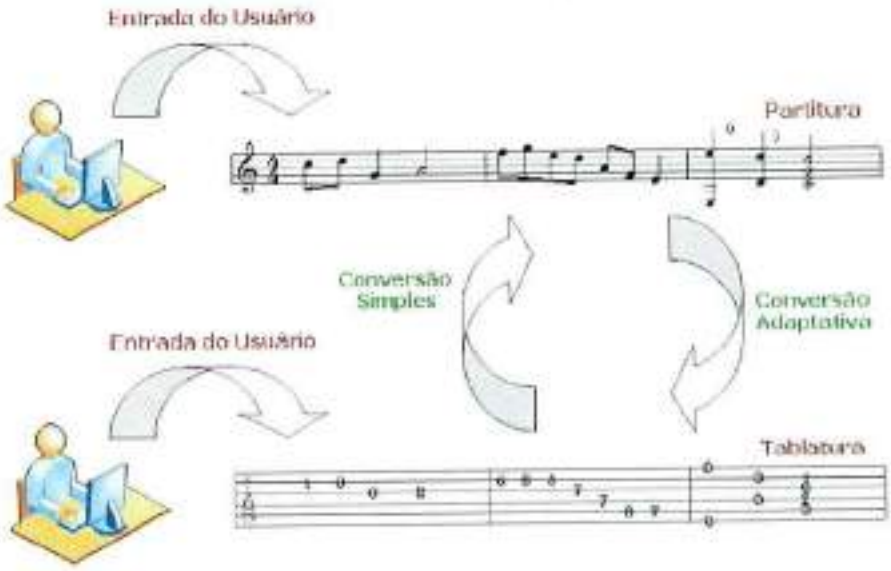

Figura 2: Esquema básico de funcionamento do Orfeu

4 Especificação

Aqui encontram-se especificados os requisitos nos quais o projeto está baseado. Esses requisitos foram elaborados com base nas pesquisas realizadas, que estão descritas adiante.

4.1 Requisitos Funcionais

A seguir são explicitados os requisitos para cumprir todos os objetivos do projeto, de maneira viável, para o cronograma do projeto:

- Inclusão um editor de partituras (notação musical padrão), permitindo que as partituras criadas sejam armazenadas em disco e que esses arquivos gerados sejam posteriormente abertos para edição. Devido à grande complexidade das exceções existentes em tal notação musical, é desnecessário que todos os detalhes sejam colocados no *software*, em sua primeira versão.
- A representação da partitura, mesmo que não atenda a todas as exceções ou detalhes das regras de grafia musical, deve ser inteligível para os músicos, logo, a representação da altura e do ritmo são essenciais e a altura jamais deve ser visualmente ambígua.
- Capacidade de importar ou abrir o maior número possível de arquivos do Guitar Pro. Precisamente, para a versão inicial, o sistema deve ser capaz de interpretar os arquivos do Guitar Pro 4, no mínimo.
- Inclusão de um editor de tablaturas de guitarra, violão ou outro instrumento similar. Esse editor deve ter capacidade de armazenamento e carregamento, assim como o editor de partituras.
- Os editores de partituras e tablaturas deverão possuir equivalência entre si, através de uma interface que gerencia a música. A música pode conter partituras e tablaturas simultaneamente e, portanto, será necessário um formato de armazenamento que contemple músicas com partituras e tablaturas. Não há necessidade de armazenar apenas parte das tablaturas ou das partituras de uma música em disco, apenas o todo.
- O *software* deverá realizar uma conversão de tablatura para partitura e vice-versa, utilizando técnicas adaptativas em pelo menos uma das conversões.

- Inclusão uma personalização de algum recurso sonoro não explícito graficamente na partitura ou tablatura. Exemplos de recursos que podem ser personalizados são glissandos (*bends*, *slides*), trinados, falhas rítmicas, microdinâmicas e reverberação.

- Criação de uma seqüência a partir das informações obtidas pela notação gráfica e dos recursos sonoros personalizados.

- Reprodução da seqüência gerada utilizando um sintetizador externo ao projeto.

- A polifonia e a polirritmia deverão ser suportadas pelo *software*.

- O foco do projeto será a interface com o usuário através do teclado, tornando-a suficientemente completa, ou seja, todos os recursos devem ser acessíveis por aqueles que não possuam ou não utilizam um *mouse*. Este requisito existe visando que o projeto será focalizado para deficientes visuais após o atendimento dos requisitos mínimos.

- Possibilidade de impressão da parte gráfica de uma música, sem a necessidade de uma grande personalização.

- Exportação da seqüência criada para outro formato.

4.2 Requisitos Não-Funcionais

Com base nos requisitos não-fucionais, o *hardware* mínimo para a execução poderá ser verificado ou estimado após a realização do projeto.

- Compatibilidade com o sistema operacional Windows XP, utilizado em conjunto com processadores de 32 *bits* Intel Pentium IV, AMD Sempron e AMD Athlon.

- A máquina hospedeira deverá possuir recursos suficientes para executar concomitantemente algum sistema de síntese sonora por *software* que esteja indiretamente integrado ao projeto.

- O desempenho do *software* deve ser bom o sufiente para reproduzir a maior quantidade possível de vozes sem falha de ritmo. Pelo menos duas vozes deverão ser reproduzidas sem que haja a falha.

- A latência entre o pedido de execução de uma música e o início de sua reprodução deve ser pequena. O valor máximo dessa latência é 1 segundo, pois valores maiores podem deixar o usuário insatisfeito.
- O *software* deverá funcionar apenas como editor na ausência de um sintetizador externo (*hardware* ou *software*) na máquina hospedeira, ou seja, a reprodução das seqüências não ocorrerá nesse caso.

4.3 Funcionalidades desejáveis

Após o atendimento dos requisitos funcionais mínimos, há funcionalidades desejáveis para a maioria dos usuários potenciais:

- Acessibilidade para deficientes visuais.
- As músicas visualmente escritas serem impressas diretamente.
- Processamento de dados de entrada MIDI externa e conversão dos mesmos para a representação interna.
- Importar arquivos de seqüência MIDI.
- Exportar arquivos no formato do Guitar Pro.
- Importar e exportar arquivos de outros *softwares* como o Sibelius, o Finale, o Encore, o Lilypond ou o PowerTab.
- Trabalhar com partituras transpostas e transpor partituras.
- Análise de harmonia e ritmo de uma música, permitindo que na importação de arquivos MIDI e na entrada por instrumento MIDI essas informações não precisem ser detalhadas pelo usuário.

4.4 Decisões para atendimento dos requisitos

O padrão MIDI deve ser adotado. A funcionalidade do *software* divide-se nas seguintes partes.

- Blocos do sistema (características das representações utilizadas como notação musical):

  - Partitura.
  - Tablatura.

- Seqüenciação (saída MIDI):
  - Geração da seqüência a partir da música editada.
  - Integração com o sintetizador.

- Edição (interface com o usuário):
  - Teclado (de computador).
  - Mouse (opcional).
  - Instrumento MIDI (opcional) - por exemplo, uma guitarra ou um teclado.

- Manipulação de arquivos:
  - Importação/Exportação (conversão de dados).
  - Abrir/Salvar arquivos próprios.

- Manipulação e interpretação dos dados.
  - Conversão de partitura para tablatura.
  - Conversão de tablatura para partitura.
  - Transposição e análise da harmonia e ritmo (opcional).

5 Projeto do núcleo

5.1 Estrutura de dados da representação musical

A seguir encontra-se uma estrutura de dados simplificada e geral para uma música, sem seus detalhes visuais ou sonoros. Tais detalhes estão descritos nos itens posteriores. Encontra-se a seguir o diagrama de classes (figura 3) e uma breve explicação de cada classe.

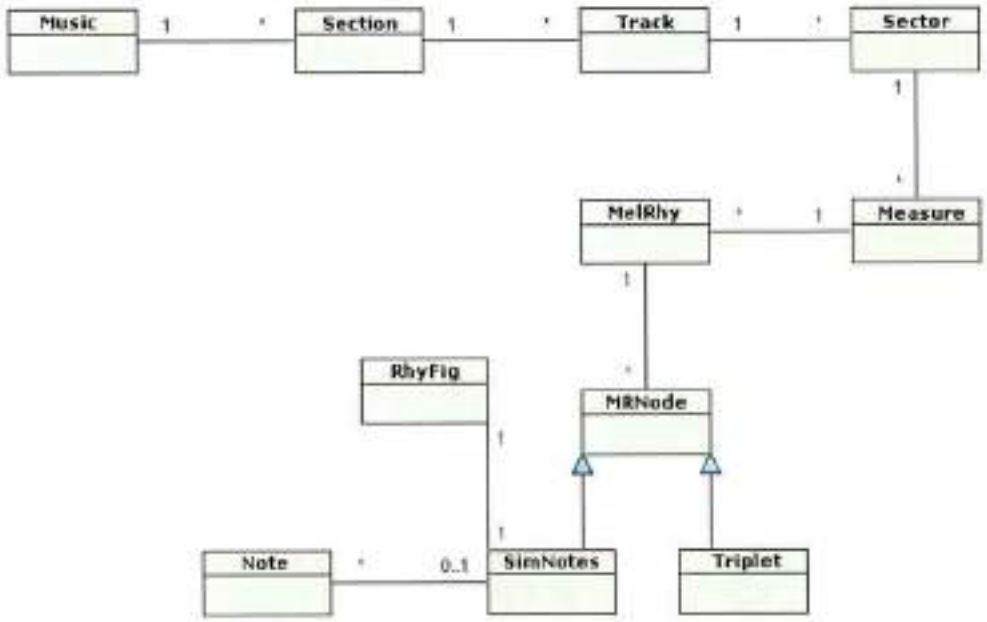

Figura 3: Estrutura de dados de uma música

- Criação de uma estrutura de dados para uma música:
  - `Note`: nota, contendo apenas a informação de altura, podendo ser obtida na forma tonal (com enarmonias, podendo ser notas diferentes, e.g. dó# e réb são notas diferentes) ou sonora (ignorando a tonalidade, e.g. dó# e réb são a mesma nota).
  - `RhyFig`: figura rítmica, contém informações sobre a duração relativa de uma nota, segundo as regras de grafia musical; não permite o uso de quiálteras.

- `SimNotes`: Conjunto qualquer de notas (não necessariamente ordenado), podendo ser nulo (pausa) ou ter várias notas (acorde). Esta é uma lista ligada de `Note`. Permite repetição de notas.
- `Tuplet`: quiáltera, contém apenas informações de relatividade entre o ritmo alterado e o ritmo não alterado.
- `MelRhy`: conjunto ordenado de `SimNotes` incluindo a possibilidade de formar quiálteras, formando um bloco de melodias ritmicamente iguais (ou um conjunto de `SimNotes` seguidos). Uma melodia é um caso particular simples de um `MelRhy`. Sua representação é uma floresta em que todas as folhas são `SimNotes` e os demais nós são `Tuplet`.
  - `MRNode`: estrutura de dados essencial para a representação da floresta como uma lista ligada heterogênea.
    - Pode indicar um `SimNote`, um `Tuplet`, um final de `Tuplet` (requerido quando representado como lista ligada heterogênea) ou um final de `Measure` (idem).
- `Measure`: compasso, conjunto de `MelRhy`, possibilitando o caso mais simples de polirritmia, aquela interna ao compasso. Esta é uma lista ligada de `MelRhy`. Apesar de não necessitar ser ordenado, é interessante que ele seja, pois permite exclusão por indexação e sua conseqüente exibição ao usuário (como é feito por outros programas).
- `Sector`: criação do conceito de parte de uma música (pentagrama ou tablatura). É um conjunto ordenado (representado por lista ligada) de compassos (`Measure`).
- `Track`: união dos "setores" de uma música em um grupo de mesmo timbre. É um conjunto ordenado de `Sector`.
- `Section`: união das "trilhas" de uma música em um grupo do mesmo naipe de timbres. É um conjunto ordenado de `Track`.
- `Music`: música, como um conjunto de seções utilizadas na mesma.

A `MRNode` possui nós que representam o final das estruturas (como o fim de compasso, por exemplo). Em particular o nó final não é musicalmente necessário, mas esses nós representam todas as posições acessíveis pelo usuário (com um cursor, por exemplo). Em outras palavras, essa estrutura já prevê que a música deverá ser editável, mas não prevê como.

Por padrão, as estruturas que são representáveis por vetores ou listas ligadas iniciam com um elemento. Apenas o `SimNotes` inicia-se vazio (que constitui uma pausa). O `Measure`, por exemplo, é iniciado com um `SimNotes` vazio. Desta forma o `MelRhy` inicia-se com 2 elementos (a pausa e a finalização do `MelRhy`).

5.2 Matemática musical

Existe a necessidade de uma nova classe para a matemática musical, que é responsável pela medida do intervalo entre duas notas. O intervalo pode ser medido de duas formas: na forma de uma relação melódica/harmônica entre duas notas ou por semitons. No segundo caso, um intervalo é apenas um número inteiro e não necessita de uma estrutura de dados específica; já no primeiro caso, o intervalo é uma composição de três partes: um número, uma orientação e uma qualificação. A nova classe está descrita a seguir:

- `Interval`: Intervalo, contém a informação do intervalo entre duas notas na forma tonal (e.g. quarta aumentada e quinta diminuta, apesar de terem o mesmo som, são considerados intervalos diferentes).

Para inserir o intervalo à matemática musical deve-se inserir métodos às classes da estrutura de dados da música que os utilizem. Neste caso os métodos são:

- `Note`:
  - Transposição simples de uma nota, perdendo a informação da tonalidade (transposição em semitons).
  - Obtenção do intervalo entre duas notas, tanto na forma tonal como na forma sonora (dado em semitons).
  - Transposição de uma nota, de forma tonal.
- `MelRhy`:
  - Cálculo da duração total do `MelRhy`, em múltiplos de um `RhyFig`.
- `Measure`:
  - Método de verificação se todos os `MelRhy` estão com a duração total do compasso exatamente preenchida.

6 Pesquisas Realizadas

6.1 Pesquisa com os possíveis usuários envolvendo enquetes e perguntas simples

Diversas comunidades do Orkut foram utilizadas para a realização desta pesquisa, totalizando 40 mil membros. O total de pessoas que responderam foi de cerca de 200. Além disso, foram consultados alunos músicos da Escola Politécnica e do Conservatório Beethoven. O total de pessoas que responderam pessoalmente à pesquisa foi cerca de 30.

Foi observada grande valorização da parte de seqüenciação do programa, mostrando que os atuais usuários dos outros *softwares* editores (e.g. Encore, Guitar Pro, Sibelius, Finale) continuam insatisfeitos com as possibilidades dos programas. Muitos usuários pouco experientes na área de música computacional associam a deficiência do programa à síntese e não à seqüenciação (muitos editores famosos incluem um sintetizador por esse motivo), o que dificulta a compreensão do que realmente é reivindicado.

Nota-se que a importância dada ao visual existe principalmente quando a questão é escrever música no computador. A impressão da notação gráfica da música é considerada essencial para muitos, mas para outros essa impressão é desnecessária. A conversão para o formato de arquivo de um *software* de impressão (e.g. Lilypond) não aparentou ser problema para os usuários (não houve críticas à idéia).

A utilização de mais de uma possibilidade de inserção dos dados no programa mostrou-se bastante interessante para os usuários. A existência da notação padrão foi considerada essencial por praticamente todos os usuários. A entrada de dados com um instrumento MIDI também foi considerada adequada para fornecer ao programa as notas tocadas pelo usuário, mesmo não detectando o ritmo do que foi tocado, já que esse recurso costuma estar disponível nas ferramentas, exigindo do usuário o fornecimento de uma grande quantidade de dados, que representam a informação em questão.

Foi considerado interessante que o *software* seja multiplataforma, porém foi observado que é prioritário que se disponha do programa em funcionamento sobre a plataforma do Windows XP. Foi nitidamente considerado essencial que o *software* apresente um bom desempenho. Imaginando uma síntese de trilhas simultâneas, é essencial que a máquina, executando o seqüenciador, não perca o ritmo da música que

estiver sendo executada, mesmo que estejam sendo usadas ao mesmo tempo todas as trilhas disponíveis (16).

6.2 Entrevistas com músicos profissionais e estudantes de música

Ao entrevistar músicos profissionais e alunos de graduação em música, vários pontos diferentes foram observados. Foi observado, por exemplo, que há, por parte dos compositores, um interesse de que sejam incluídas, na ferramenta, notações gráficas modernas (em alguns casos, imprecisas quanto à execução, dificultando grandemente a sua exata seqüenciação). Muitas dessas notações seriam alternativas muito pessoais, cuja presença poderia não ser de interesse geral em um *software* feito para o grande público, por tratar-se de uma abrangência excessiva. No entanto, várias delas mostram-se essenciais, por promoverem maior precisão em elementos originalmente pouco precisos.

Aqui a seqüenciação novamente tem destaque. Foi observado que os ornamentos do tipo glissando raramente são bem executados em seqüências, e que a disponibilidade de uma interface especial para tais ornamentos se mostra interessante. Para um compositor, mostra-se importante, dessa maneira, que seja possível obter, com a ferramenta, uma reprodução fiel da sonoridade por ele imaginada.

Certos fatos com relação às possibilidades composicionais de obras modernas foram discutidos, e a conclusão foi que a reivindicação dos compositores é que a facilidade de uso para recursos que ainda não são encontrados nos *softwares* existentes, obrigando-os ao uso de artifícios às vezes desconfortáveis para representar apropriadamente aquilo que desejam. Por exemplo, por falta de opções, na ausência de tais recursos, o compositor acaba, com freqüência, sendo forçado a baixar o nível de abstração usada na representação de certos detalhes de suas composições, descrevendo-as diretamente em arquivos MIDI, com a ajuda de um *software* seqüenciador.

Há um grande interesse de um particular recurso - o de possibilitar a representação confortável de rítmicas complexas - a ponto de atrair para a utilização da ferramenta aqui proposta grande parte dos usuários de outros *softwares*, por se tratar de um recurso bastante conhecido e usado atualmente por compositores modernos e, em certos casos, usados até mesmo na música popular, e, apesar disso, não contemplado ou não disponibilizado de forma confortável nas ferramentas atualmente em uso freqüente.

6.3 Pesquisa de componentes de *software* e de partes reutilizáveis

Foi encontrado um *software* implementado em Delphi contendo apenas uma interface entre um teclado MIDI ou um arquivo MIDI e o sintetizador padrão do computador, permitindo que uma pessoa utilize o teclado MIDI com os timbres do sintetizador padrão do computador. Esse *software* utiliza um componente próprio a partir do qual é possível basear-se para a criação de uma interface mais genérica de conexão entre o *software* e a entrada MIDI, saída MIDI, entrada de arquivos MIDI e saída de arquivos MIDI.

Diversos outros *softwares* (cerca de dez) foram analisados, porém poucos disponibilizavam o código fonte, e a maioria não apresentava o componente a ser usado como interface com os dispositivos de entrada e saída MIDI. Entre esses *softwares*, o DGuitar, implementado em Java, é um visualizador de tablaturas geradas pelo *software* Guitar Pro, porém possui um código-fonte extremamente difícil de entender e apresenta poucos recursos disponíveis para seus usuários.

Foi encontrado um documento, baseado na engenharia reversa do formato de arquivo GP4, no qual é explicitado o significado de cada um dos elementos encontrados em arquivos desse formato. Esse documento, embora tenha auxiliado na importação de arquivos do Guitar Pro 4, contém alguns erros.

6.4 Pesquisa sobre as possibilidades de linguagens de implementação

Dentre as diversas possibilidades de linguagens para a implementação do *software* aqui proposto, foi preciso selecionar uma. Para isso, foram utilizados os seguintes critérios:

(I). IDE amigável, com *debugger*.

(II). Facilidade para a criação de interfaces gráficas.

(III). Existência de recursos de integração com dispositivos MIDI.

(IV). Existência de recursos simples para manipulação direta de arquivos.

(V). Multiplataforma (funcionamento no Microsoft Windows considerado essencial).

(VI). Desempenho.

(VII). Orientação a objetos e recursos adicionais (e.g. sobrecarga de operadores, propriedades, multiparadigma, classes prontas de estruturas de dados básicas como pilhas e filas, etc.).

(VIII). Conhecimento prévio de certas funcionalidades da linguagem.

(IX). Disponibilidade de recursos gráficos (normalmente utilizando objetos do tipo `Canvas`).

Dentre as possibilidades de linguagens, a comparação se passou entre:

- C++ (Utilizando Qt ou GTK, com a IDE KDevelop)
- C#.NET
- Delphi/Kylix
- Java

Essas foram as escolhidas, por serem linguagens que possuem a orientação a objetos, recursos gráficos, são multiplataforma, etc. Os pesos dados a cada critério foram escolhidos com base na escala de 1 a 5, em que 5 indica que o critério é importantíssimo ou essencial, e 1 indica que a importância é pequena, tendo em vista os outros critérios. Os pesos foram estimados de acordo com sua importância estimada no desenvolvimento. Utilizando tabela 1 a seguir, elegeu-se a linguagem a ser utilizada:

Tabela 1: Critérios utilizados na decisão da linguagem de programação

Critério	**Peso**	**C++(Qt)**	**C++(GIK)**	**C#.NET**	**Delphi/Kylix**	**Java**
I	3	7	7	7	9	8
II	2	7	4	9	10	8
III	5	6	6	7	5	8
IV	5	10	10	8	9	8
V	2	10	10	7	4	10
VI	5	8	9	5	10	4
VII	3	9	9	5	8	2
VIII	3	6	6	5	7	6
IX	5	9	9	8	10	10
Total	**33**	**265**	**264**	**223**	**270**	**234**

Os valores foram estimados com base (i) na bibliografia listada ao final deste relatório, (ii) no conhecimento prévio do grupo sobre cada linguagem e (iii) no conhecimento de alguns alunos da Escola Politécnica que desenvolvem programas nessas

linguagens e que foram consultados. Algumas justificativas dadas para cada nota, em cada critério:

(I). A interface KDevelop do C++ possui *debugger* e está comparável às demais, porém incorpora alguns erros, o que cria dificuldades aos seus usuários, enquanto que a IDE utilizada pelo Java (considerando o uso do Netbeans ou do Eclipse) mostra-se de maior qualidade em certos aspectos, juntamente com a interface do Delphi. O total de dados pedidos pela IDE de cada possibilidade foi considerado no critério, e a qualidade das mensagens de erro do *debugger* também.

(II). Para o caso do Qt, há um anexo ao KDevelop e um *software* próprio (Qt Designer) para auxiliar o desenvolvimento de uma GUI, ambos de utilização mais complexa que os equivalentes do C#.NET, do Java e do Delphi/Kylix. O número de informações que se necessita conhecer, a separação entre o desenvolvimento estético da GUI e o desenvolvimento do seu código subjacente foi considerada neste critério.

(III). A existência nativa de componentes voltados ao desenvolvimento da interface gráfica foi considerada importante, porém é necessário que estejam disponíveis exemplos utilizando tais componentes ou componentes externos, para que possam ser utilizados no projeto. Desta forma, a disponibilidade de exemplos, encontrados durante a pesquisa de componentes, assim como a existência de partes reutilizáveis, acessíveis e disponíveis para uso, teve grande influência nas notas dadas neste critério.

(IV). Como a presença de componentes nativos voltados ao desenvolvimento da interface gráfica (GUI) está presente em todas as linguagens, foi considerada a quantidade de dados pedida por cada linguagem para as mesmas informações, e como essa informação pode ser tratada, dividindo-a em *tokens*.

(V). Neste item foi identificado como de alta importância o número de alterações necessárias no código para que se possa portá-lo para outro sistema, bem como o número de sistemas em que é possível que o projeto funcione. Aqui é considerado que já existe um requisito quanto a isso, a exigência de que, no mínimo, o projeto funcione sobre o sistema operacional Windows da Microsoft.

(VI). Pelo fato de C++ e Delphi não serem linguagens com código-objeto interpretado, o desempenho mostra-se melhor que o das linguagens interpretadas. Há a possibilidade de gerar programas executáveis a partir dos arquivos do C#.NET, porém tais códigos não se mostram tão eficientes quanto os do C++ ou do Delphi.

(VII). O fato de uma linguagem obrigar a existência de um método principal dentro de uma classe, em lugar de uma função que gere as suas primeiras instâncias, foi considerado como um fator estético e lógico negativo, apesar de pouco importante neste item. Recursos adicionais como sobrecarga de operadores e propriedades, que permitem códigos mais fáceis de serem compreendidos, foram considerados importantes. Linguagens que permitem a mistura de paradigmas mostram-se mais adequadas ao projeto por possuírem estruturas básicas otimizadas, maior desempenho e simplificação de partes atípicas de código, para a orientação a objetos. A existência de estruturas de dados básicas não foi considerada, pois todas as linguagens em estudo apresentam tais estruturas.

(VIII). O conhecimento próprio de cada linguagem, por parte dos desenvolvedores, foi considerado essencial para evitar atrasos no cronograma. O conhecimento de como utilizar os recursos gráficos (item a seguir), da maneira de criar a interface gráfica, e a manipulação de arquivos mostraram-se os mais influentes nesta estimativa.

(IX). Java e Delphi/Kylix comportam-se quase da mesma forma quanto aos objetos `Canvas`, segundo o que foi visto em exemplos na pesquisa de partes reutilizáveis. C#.NET, conforme descrito em seu arquivo de ajuda, procura fazer o mesmo, porém de uma forma diferente, e exigindo o fornecimento de mais dados para as mesmas informações. C++, com GTK e com Qt, tem esse aspecto bastante documentado, apesar apresentar menor clareza que os objetos usados em Java e em Delphi/Kylix.

7 Computação Gráfica

7.1 Pesquisa sobre computação gráfica

A necessidade de criação de uma interface com o usuário para este projeto exige conhecimentos apurados de computação gráfica. Representar uma partitura com todos os seus possíveis detalhes e suas personalizações em uma tela possui complexidade similar à construção de um *software* de edição de imagens vetoriais. Tal complexidade inviabiliza a criação do *software*, logo alguma restrição deve ser feita, e por esse motivo o requisito sobre a exibição das partituras não deve exigir um excesso de detalhes.

Entre as possibilidades, estão:

- Gráficos matriciais
  - Imagens fixas: sem ampliações ou reduções (*zoom*).
  - Várias imagens: ampliações ou reduções de escala fixa.
  - Imagem fixa com *stretch* (redimensionamento de imagens matriciais com estimativas de cor): ampliações ou reduções personalizadas com distorção.
  - Várias imagens com *stretch*: ampliações ou reduções personalizadas com distorção menos acentuada nas proximidades de algum conjunto de imagens.
- Gráficos vetoriais
  - Impressão
    - Pré-renderização: *Sprites* (jargão de desenvolvimento de jogos) ou imagens rasterizadas. Os *sprites*, normalmente, contêm animação envolvida mas não há necessidade de animação neste projeto.
    - Reconstrução completa da tela a cada atualização.
  - Origem dos dados
    - Fonte com estrutura própria definida para o sistema.
    - Fonte típica do Windows.
    - Primitivas de desenho.

Essas possibilidades ainda podem entrar junto com o conceito de camadas (*layers*), tipicamente usado em programas de tratamento matricial de imagens e em jogos, normalmente para separar os fundos dos *sprites*. Essa técnica pode ser usada tanto em gráficos matriciais como em gráficos vetoriais.

Quanto à movimentação da tela por barras de rolagem existem pelo menos 3 possibilidades que podem ser estudadas:

- Tela virtual completa (técnica usada em diversos jogos).
- Conjunto de telas virtuais (técnica usada no Google Maps, que consiste em várias telas menores que a tela visível e justapostas lado a lado, renderizadas individualmente).
- Tentativa de reconstrução de todos os elementos, dentro ou fora da tela (técnica usada em alguns jogos).

Para a impressão, existem várias possibilidades de objetos hóspedes de um `Canvas` que podem ser utilizados, entre eles os objetos `Form`, `Image` e `PaintBox`, todos da API do Windows. Dos citados, o que apresenta melhor desempenho é o `PaintBox`.

Um ensaio de interface foi realizado procurando testar essas diversas soluções gráficas que podem ser usadas no projeto, buscando conhecer a melhor. Neste ensaio foram utilizados:

- Criação das estruturas de dados:
  - *Sprite*: figura bidimensional pré-renderizada (rasterizada), que no *software* em questão não é necessário que contenha animação.
    - *Sprites* devem conter um método de impressão em um `Canvas` de parâmetro, recebendo coordenadas relativas e utilizando-as.
  - *Layer*: camada.
    - Método virtual de impressão
    - Deve ter herdeiros:
      - `Foreground`: conjunto de *sprites*. A impressão aqui deve chamar a impressão dos *sprites*, passando as coordenadas relativas.
      - Cursor: Análogo ao *Foreground*, mas possui outra ordem de impressão.

- `LinesBackground`: fundo como um conjunto de linhas horizontais. A impressão aqui é diretamente feita.
- `ImgBackground`: fundo como uma imagem colocada lado a lado verticalmente, com uma cor de fundo.

– `Screen`: tela, conjunto de *layers*. Os *layers* `ImgBackground` são impressos primeiro, depois os layers `LinesBackground`. Os seguintes itens deverão ser observados.

* Método de inserção de um novo *layer* na ordem correta.
* Construtor deve inserir o primeiro `ImgBackground`.
* Apenas uma instância de `Screen` deve ser criada, tendo como parâmetro um objeto `Canvas` que será o destino da impressão da tela sempre.
* Método de impressão que chama a impressão dos *layers*, passando as coordenadas relativas (*rebuild*).
* Método de atualização sem nova leitura dos dados originais (*refresh*).
* Método de atualização das dimensões (*resize*), ligada ao evento de modificação do tamanho do objeto que possui o `Canvas` de impressão.

- Inserção de *scrolling*.
  - Barras de rolagem na interface gráfica.
  - Métodos de atualização da posição relativa na `Screen`

Ainda não há um esclarecimento dos resultados, porém o uso de técnicas de otimização nos gráficos mostra-se necessário.

7.2 Atributos visuais refletidos na estrutura de dados

As informações aqui colocadas são algumas idéias tidas até o momento da forma com que os gráficos seriam utilizados pela interface gráfica.

- *Sector*:
  - É adicionado `LinesBackground` como (neste caso) uma tablatura vazia, atualizando-a em sua construção e pelo redimensionamento da tela.
  - É adicionado `Foreground`.

  - Método para gerar um *sprite* contendo a falsa clave de uma tablatura, colocando-a em seu *layer* `Foreground`.

- `SimNotes`:

  - Um *sprite* das notas ou da pausa em tablatura (apenas os números).

- `Measure`:

  - Ordenação de `SimNotes`, essencial quando há simultaneidade de vozes (`MelRhy`). A ordenação deve ser feita sempre na inserção de uma nota, fornecendo um número de ordem para todos os `SimNotes`, número esse relativo ao compasso. Números iguais representarão notas tocadas simultaneamente.
  - Método de impressão de todas as suas notas, com coordenadas relativas. Apenas faz a chamada do método de impressão de cada `SimNotes`, passando as coordenadas internas ao compasso corretamente.
  - É adicionado um método para gerar *sprite* de sua fórmula de compasso.

- `BarLine`: nova estrutura para armazenar a barra de finalização de um compasso.

  - Criação de método para gerar um *sprite* de saída com uma barra de final de compasso.

- `Measure` passa a ter um `BarLine`

- O método de impressão do `Measure` passa a receber como parâmetro dois *flags*, um que indica se a fórmula de compasso deve ser impressa e outro que indica se a barra de compasso deve ser impressa.

- Classe `SimNotes` recebe método que gera *sprites* de ritmo, simplificado.

- `Note` passa a ter atributos e métodos específicos de tablatura (corda, casa, etc.), para permitir a personalização.

- Generalização na criação dos *sprites* e fundos relativos a tablatura.

  - Linhas de tablatura em `Sector`.
  - Falsa clave de tablatura em `Sector`.
  - *Sprites* dos números nos `SimNotes`.

- Ferramenta visual que permita tal personalização (janela), proibindo casos degenerados como a ausência de cordas. As cordas sempre devem estar afinadas em uma das 12 notas da escala cromática (temperada).

- `SimNotes`:

  - Atributos gráficos (como inversão de haste e deslocamento horizontal).
  - Um *sprite* das notas ou da pausa em partitura (bandeirolas sempre desunidas e sempre para cima).

7.3 Graphics32

Após o estudo entre as possibilidades de utilização das diferentes ferramentas básicas, um estudo foi feito a fim de verificar qual seria a melhor alternativa para a confeccção da parte gráfica do projeto. Qualquer das alternativas pesquisadas consumiria um intervalo de tempo considerável, tornando difícil o término do projeto no prazo estipulado, caso fosse necessário construir toda a biblioteca gráfica.

Entretanto, foi constatada a existência da biblioteca Graphics32, desenvolvida principalmente para Delphi e Kylix, que é otimizada para imagens de 32 *bits*. Para o projeto do editor musical, a *performance* é um fator relevante, pois o editor precisa permitir um grande número de símbolos musicais e executar a música, através da saída MIDI.

A biblioteca Graphics32 fornece uma forma fácil de trabalhar com *layers*, implementa o recurso da transparência, pois as duas extensões de arquivo com que trabalha (bmp e jpg) não permitem que a imagem tenha fundo transparente - o que é necessário para o projeto em questão. A possibilidade de trabalhar com *layers* também auxilia na confecção da interface gráfica segundo o modelo adotado.

A biblioteca Graphics32 mostrou-se poderosa o suficiente para concluir a pesquisa de computação gráfica e anulou o que foi pensado sobre os atributos visuais refletidos na estrutura de dados. Isso foi extremamente vantajoso, pois a interface gráfica não ficou dependente da estrutura de dados, o que torna os módulos do *software* bem definidos.

7.4 Autômatos Adaptativos

A seguir será feita a análise dos autômatos adaptativos como ferramenta complementar para o desenvolvimento do projeto.

Autômatos adaptativos são dispositivos mais poderosos que os autômatos de pilha estruturados pela sua capacidade adaptativa. As transições de um autômato adaptativo podem ser de dois tipos:

- Transições normais, que são similares às transições de um autômato de pilha estruturado.
- Transições contendo ações adaptativas, responsáveis pela mudança de configuração do autômato.

O autômato inicia-se com uma configuração, e, dependendo das transições executadas, é modificado até que chegue a uma configuração final, quando a cadeia de entrada é inteiramente consumida ou até que não haja nenhuma transição possível. Se a cadeia tiver sido inteiramente consumida e o autômato terminar em um estado de aceitação com a pilha vazia, então significa que a cadeia foi aceita. Caso contrário, a cadeia não foi aceita.

As ações adaptativas podem ser de três tipos:

- Inspeção: verifica o valor de uma variável.
- Inserção: adiciona uma transição ao autômato. Neste caso, se a transição a ser adicionada ocasionar a mudança para um estado não presente no autômato, o mesmo é criado.
- Remoção: remove uma transição do autômato. Neste caso, se a transição a ser removida tornar um estado inalcançável, o mesmo é removido.

As ações adaptativas podem ocorrer em dois momentos:

- Antes da transição: são as ações adaptativas anteriores. Devem ser executadas antes de ocorrer a mudança de estado. Neste caso, se a ação remover a transição a ser executada, o autômato permanece no estado presente e uma nova transição deve ser executada.

- Depois da transição: são as ações adaptativas posteriores. Devem ser executadas depois da mudança de estado. Neste caso, se a transição for removida pela ação adaptativa o autômato terá mudado de estado.

8 MIDI - Musical Instrument Digital Interface

8.1 Resumo da arquitetura do MIDI e sua conexão com o projeto

O MIDI é um padrão de transmissão serial de dados, que permite a troca de informações entre instrumentos e equipamentos de aplicação musical. A figura 2 mostra um cabo MIDI fêmea. Resumidamente o padrão MIDI envolve um seqüenciador e um sintetizador e a comunicação através do mesmo pode ser observada através das figuras 4 e 5.

Figura 4: Cabo MIDI fêmea

Figura 5: Arquitetura: comunicação MIDI

Tanto o seqüenciador quanto o sintetizador podem ser *hardware* ou *software*. É comum encontrar placas de som com sintetizadores MIDI por *hardware*, porém como nem todo usuário possui esse recurso. O MS Windows possui um sintetizador MIDI por *software*, possibilitando que qualquer usuário do mesmo possa utilizar o computador para síntese de sons a partir de seqüências MIDI.

O MIDI não precisa ser utilizado apenas para enviar dados a um sintetizador, esse é apenas o caso mais comum. O MIDI também pode ser utilizado por um seqüenciador para controlar um outro seqüenciador (por exemplo, um pedal MIDI em um teclado MIDI), e também pode ser utilizado em um dispositivo de armazenamento (por exemplo, um *software tracker*, isto é, um *software* de "mixagem" de sons armazenados em seqüências MIDI ao invés de formas de ondas). O mais importante a ser notado é que o MIDI é um protocolo unidirecional. A figura 6 mostra exemplos de comunicação MIDI.

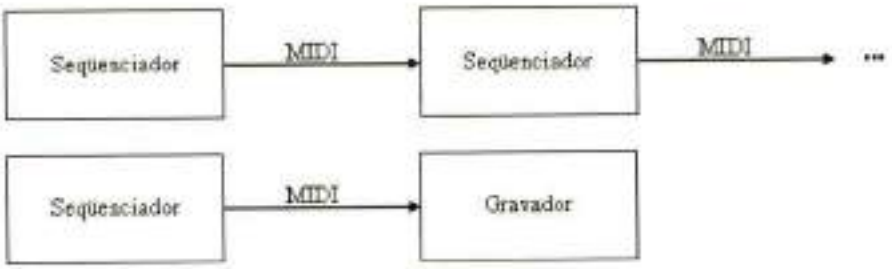

Figura 6: Outros exemplos da comunicação MIDI

8.2 Mensagens MIDI

O protocolo é assíncrono e funciona da seguinte forma: dado um seqüenciador (como um teclado MIDI), enviará uma mensagem no instante em que a tecla é pressionada, e o objetivo é que o sintetizador faça com que a nota relativa a essa tecla soe sem que o usuário perceba que houve um atraso na geração do som, ou seja, a latência deve ser mínima. Essas mensagens são, portanto, tratadas na hora em que são recebidas, e são de 3 tipos: Eventos MIDI, eventos `SysEx` e meta-eventos.

Os eventos `SysEx` (*System Exclusive*) são particulares de cada dispositivo MIDI e não serão utilizados nesse projeto. Os meta-eventos são mensagens existentes apenas em arquivos MIDI, para incluir informações que não são transferidas pelo protocolo (e.g. informações ritmicas, final de uma trilha, nome de uma trilha, nome do autor da música, letra da música, etc.). Apenas os eventos MIDI não podem ser grandes, tendo no máximo o tamanho de 3 *bytes*, sendo 1 *byte* de *status* e 2 *bytes* de parâmetros. O primeiro *byte* é decomposto em seus 2 *nibbles*, sendo que o primeiro indica o tipo de evento MIDI sendo enviado e o segundo indica o canal MIDI tratado naquele instante. A tabela 2 ilustra esse caso.

Tabela 2: Mensagem MIDI

Status		Parâmetros	
Tipo	Canal	Parâmetro 1	Parâmetro 2
7654	3210	76543210	76543210

Os parâmetros sempre devem ser menores que 128, ou seja, seu primeiro *bit* deve ser sempre 0. Dessa forma quando o *byte* de *status* se repetir, será necessário apenas o reenvio dos parâmetros (pois o *byte* de *status* sempre é maior que 127. Basta verificar o primeiro *bit* para descobrir se é uma mensagem nova ou se são parâmetros novos para o *status* antigo). Os tipos de evento MIDI são mostrados através da tabela

3, a seguir. Os eventos contidos nessa tabela estão descritos posteriormente.

Tabela 3: Eventos MIDI

Tipo Evento	Código	Parâmetro 1	Parâmetro 2	Evento
Note Off	0x8	número da nota(*)	dinâmica	(I)
Note On	0x9	número da nota	dinâmica	(II)
Note Aftertouch	0xA	número da nota	valor	(III)
Controller	0xB	número do controle	valor	(IV)
Program Change	0xC	número do programa	*não utilizado*	(V)
Channel Aftertouch	0xD	valor	*não utilizado*	(VI)
Pitch Bend	0xE	valor (LSB)	valor (MSB)	(VII)

(I). Faz a nota parar de soar. Neste caso a dinâmica se refere à "velocidade com a qual a tecla foi solta. Isso pode influenciar efeitos como o do cravo.

(II). Faz com que a nota comece a soar.

(III). Altera a condição da nota (após soar). Isso pode ser uma mudança de vibrato ou de dinâmica, por exemplo.

(IV). Modifica parâmetros do som, como o panorama, a reverberação, pedal de *sustain*, etc. Existe uma tabela relacionando o números de controle de cada um desses parâmetros.

(V). Modifica o timbre selecionado.

(VI). Aplica a alteração da condição da nota a todas as notas do canal que estiverem soando.

(VII). Se aplica a todas as notas do canal atual e serve para fazer um glissando à nota com alta precisão: de 0 a 16383. A faixa do glissando depende da configuração do sintetizador. É preciso lembrar que os 2 parâmetros possuem 7 *bits*.

(*) A nota de valor 60 é o dó central, 61 é o dó# central e assim por diante, de 0 a 127.

Além disso, existem no formato de arquivo MIDI infomações de duração das notas, de trilhas e cabeçalhos que inexistem no protocolo MIDI. Essas mensagens, na forma de meta-eventos, têm duração arbitrária e que não serão explicitadas neste item.

8.3 API do Windows e componentes MIDI

A API do Windows mostrou-se bastante completa, porém bastante complicada para que pessoas inexperientes a utilizem diretamente. Foi necessário ver o funcionamento da mesma, pois nenhum componente de MIDI I/O respondia satisfatoriamente ao programa. Para o *software* em questão, é preciso algo que possua as seguintes características:

- Gratuito
- Sem a exigência da abertura do código segundo normas como a da GPL, para não comprometer o futuro do projeto (isso evitaria o uso futuro de componentes do projeto, por exemplo).
- Para uso com o Delphi.
- Interface com a saída MIDI existente no sistema operacional.
- (Opcional) Compatibilidade com o Kylix, para possibilitar a existência de uma versão para Linux do projeto.
- (Opcional) Interface com a entrada MIDI.

Como nenhum pacote de componentes de *software* encontrado atendeu adequadamente a esses requisitos, a busca se deu por componentes que tivessem a maior parte desses itens para que pudesse ser usado na criação de um novo pacote de componente interno ao projeto. O pacote encontrado foi o `RtMIDI`, para C++ e os componentes internos do projeto PianoEx, para Delphi. O `RtMIDI` utiliza *strings* como mensagens, o que pode causar problemas com o caractere nulo, e não há um temporizador (ou seja, somente pode ser utilizado de interface com o I/O MIDI, não fornecendo nenhum serviço adicional). O PianoEx, na versão encontrada, possui com um código ruim, com várias informações irrelevantes. Apresentou erro na execução de certas músicas. Seu temporizador é baseado em mensagens (cujo problema será visto a seguir). Apesar de serem inadequados, esses dois *softwares* auxiliaram, junto com o MSDN, a criação de um novo pacote de componentes para a interface com o MIDI.

(I). Arquitetura do Windows.

O Windows tem uma arquitetura baseada em mensagens, ou seja, uma aplicação se comunica com outra e essas se comunicam com o sistema operacional através

de mensagens. Simplificadamente cada aplicação tem sua fila de mensagens e quando é dado a ela o uso do processador, essas mensagens são lidas. Um exemplo de mensagem é o clique de um *mouse*: uma mensagem é enviada à aplicação que monitorava o *mouse* avisando que houve um clique em determinada posição, para quea mesma possa então executar a ação desse clique. Essa arquitetura deve ser evitada em aplicações que exigem tempo real ou são muito sensíveis a variações de latência, pois uma mensagem leva um tempo aleatório para ser tratada, já que ela está em uma fila e não é possível saber em que momento a aplicação verificará as mensagens dessa fila.

Para aplicações que exigem maior velocidade, existe uma alternativa, o uso de rotinas de *callback*. Essas rotinas são chamadas diretamente pelo sistema operacional quando ocorre um determinado evento. Essa funcionalidade é essencial em certas aplicações de tempo crítico.

Adicionalmente diversas questões sobre o funcionamento do sistema operacional em questão precisaram ser vistas para auxiliar o desenvolvimento, como a padronização dos nomes, alguns arquivos importantes, conceito de `Handle`, `HWND` e `HMIDIOUT`, etc., mas nada foi tão relevante quanto esse item abordado acima.

(II). Temporizador Multimídia (*Multimedia Timer*).

No caso do projeto em questão, a utilização do objeto usual de temporização, incluso no Delphi, não é viável, pois ele trabalha com mensagens e o projeto utiliza MIDI, um recurso multimídia que exige a menor variação de latência possível (60ms é considerado um valor extremamente alto de latência). Isso torna necessário o uso de um temporizador que utilize o *callback* ao invés de mensagens.

Dessa forma, um temporizador multimídia precisou ser construído. A API do Windows possui um recurso de criação de temporizadores desse tipo, porém não é possível parar um temporizador e continuar depois, ou seja, o componente no Delphi deve criar um novo temporizador do Windows a cada chamada.

Pelo menos uma versão do Windows não possui um *flag* na criação do *timer* que proíbe que o mesmo dispare outra chamada à função *callback* (`TimerProc`) após a destruição do *timer* (*flag* `TIME_KILL_SYNCHRONOUS`), o que impõe que todo objeto `Timer` deve ser desativado antes do início da destruição dos componentes do programa, a fim de evitar problemas com o acesso da memória que já foi liberada. Adicionalmente o `Timer` não deve ser ativado durante a

criação antes do carregamento do programa, para evitar que haja algum acesso à memória não alocada.

(III). Saída MIDI.

O Windows fornece rotinas básicas para o acesso aos dispositivos de saída MIDI (que quase sempre são sintetizadores) que permite:

- Obter o número de dispositivos de saída MIDI existentes.
- Obter informações específicas de cada um dos dispositivos.
- Abrir e fechar uma conexão com o dispositivo.
- Desativar todas as notas que foram enviadas.
- Enviar uma mensagem de 32 *bits* (o *byte* menos significativo será enviado primeiro).
- Interceptar mensagens sendo enviadas ao dispositivo (por *callback*).
- Ler e alterar o volume sonoro.
- Envio de mensagens longas.

O `RtMIDI` fundiu os métodos de envio de mensagens longas e curtas em um só, e inseriu tratamento de erros, porém não está livre ainda das mensagens MIDI, sendo ainda necessário o conhecimento dos *opcodes* do MIDI e da sintaxe do mesmo.

Todo esse esquema é vinculado a um `HMIDIOUT`, tipo de `Handle` específico para esse dispositivo. Para colocar prioridade em tempo real, é necessário que componente de execução faça isso com o `Handle` (`HWND`) da janela (visual ou não) da execução do MIDI, usando um temporizador multimídia.

9 Pequenos programas para testar tecnologias

9.1 `MIDIOutTest`

Trata-se de um programa simples que implementa um pequeno teclado na tela do computador a ser acionado com o auxílio de um mouse, funcionando como se fosse um teclado de um piano. Utiliza a primeira porta MIDI do computador, no timbre inicial (geralmente o timbre de piano). O objetivo desse programa é assegurar o funcionamento do componente `TMIDIOut` de maneira simples e interativa. A figura 7 mostra a interface construída para possibilitar o teste.

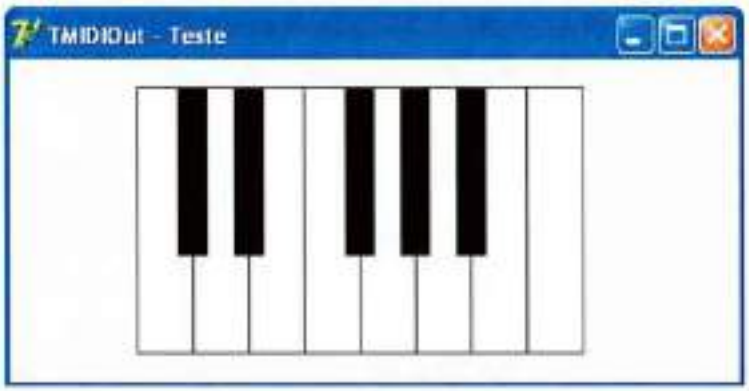

Figura 7: Interface construída para testes com a interface MIDI

Além do teste do `TMIDIOutTest`, houve outro teste, incluindo além desse componente o `TMMTimer`, ou seja, é um teste que inclui variação visual e sonora simultaneamente. A figura 8 mostra a paleta de componentes reutilizáveis do Orfeu.

Figura 8: Paleta de componentes reutilizáveis do Orfeu

9.2 Teste com arquivos XML

Para a manipulação de arquivos XML, foi feito um programa simples, em que um arquivo XML é aberto e são removidos os espaços em branco do mesmo. A figura 9 mostra a tela de execução e o resultado. Após o teste, foram incorporadas as funções de abrir/salvar arquivo da interface com o usuário.

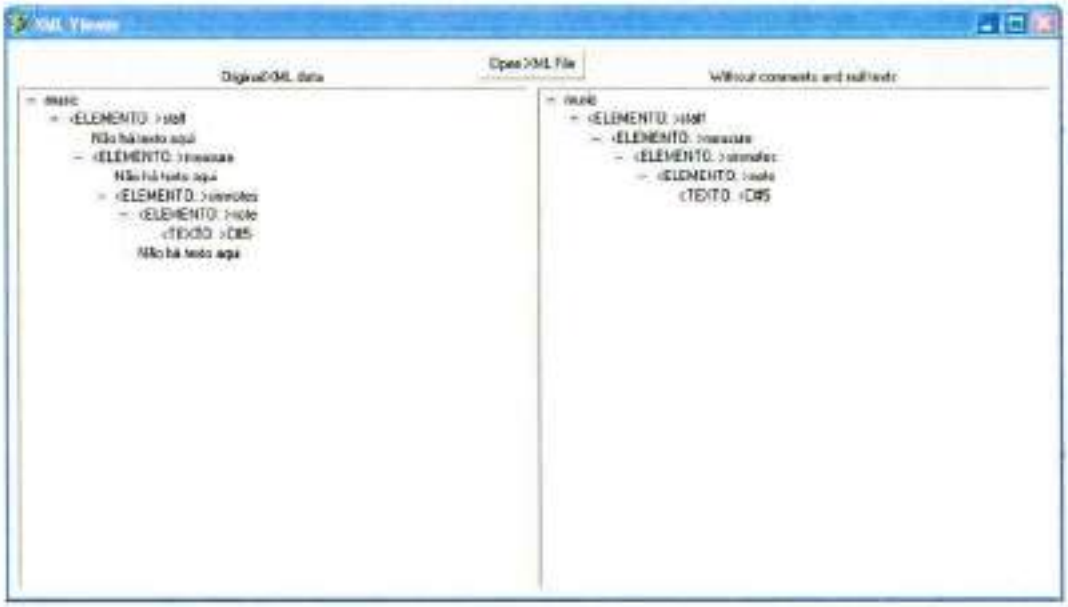

Figura 9: Teste de abertura de arquivo XML

10 Detalhamento

10.1 Componentes (interface MIDI)

Um pacote de componentes do projeto, responsável pela interface MIDI, foi iniciado em `OrfeuComponents.dpk`, registrando seus componentes na aba `Orfeu` do Delphi. Esse pacote é de uso exclusivo para o Windows, porém há intenções de torná-lo compatível com o Linux no futuro.

(I). `MMTimer.pas` - Componente `TMMTimer`

É um componente que implementa um temporizador multimídia a partir da API do Windows. Sua funcionalidade é bastante similar ao do `Timer` do Delphi para o usuário. É um herdeiro direto de `TComponent`, sem métodos adicionais, e que publica as seguintes propriedades:

- `Enabled`: Define se o temporizador está ativado ou não. Deve ser mudado para falso antes de iniciar a destruição dos componentes do programa e deve ser iniciado como falso.
- `Interval`: Período do temporizador em que é feita a chamada do procedimento associado ao evento `OnTimer`.
- `Resolution`: Define a qualidade do temporizador. Quanto menor, melhor é a qualidade.
- `OnTimer`: Evento que associa ao mesmo um procedimento que será executado pelo temporizador periodicamente.

(II). `MIDI.pas`

Este arquivo não inclui componente registrado. Seu conteúdo inclui:

- Constantes dos tipos de eventos MIDI: `MIDI_ETYPE_*` (e.g. `MIDI_ETYPE_NOTEOFF`).
- Constantes dos números de controle do evento de controle MIDI: `mc*` (e.g `mcSustain`, `mcPanMSB`).
- Classe abstrata para a entrada e saída MIDI:
  - Classe `TMIDIIO`
    Esta classe publica a propriedades:

* `Connected` (somente leitura): indica se o componente está ou não conectado ao dispositivo. As classes herdeiras podem editar o valor de `_Connected`, que é o local de armazenamento dessa propriedade (variável protegida, propriedade pública somente de leitura).
* `PortNumber`: número de identificação do dispositivo com o qual ele se conecta; varia de 0 até `número_de_dispositivos` - 1, separadamente para a entrada MIDI e para a saída MIDI. O valor inicial de`PortNumber` é fixo (`DEFAULT_PORT_NUMBER` = 0) e sempre os objetos `TMIDIIO` são iniciados desconectados. Ao alterar o `PortNumber` quando o componente estiver conectado, ele automaticamente reconectará à nova porta MIDI. Na destruição, o objeto MIDI é desconectado caso ele ainda haja conexão, antes de prosseguir, logo, isso não precisa ser verificado. Adicionalmente essa classe abstrata impõe às classes herdeiras os seguintes métodos:
  - `OpenPort`: abertura em `PortNumber` a conexão com o dispositivo.
  - `ClosePort`: fechamento da conexão com o dispositivo em `PortNumber`.
  - `GetPortCount`: obtém o número de dispositivos MIDI (de entrada ou de saída) existentes no sistema.
  - `GetPortName`: obtém um nome para o dispositivo MIDI apontado pelo `PortNumber`. Esse nome serve apenas para o usuário ter alguma informação sobre o dispositivo ao selecioná-lo em uma lista.

(III). `MIDIOut.pas` - Componente `TMIDIOut`

Este componente implementa a interface com a saída MIDI de um computador. Essa interface permite que o usuário não tenha conhecimento da API do Windows, e também permite, até certo ponto, que o mesmo desconheça algumas informações sobre o protocolo MIDI. Trata-se de uma classe herdeira do componente `TMIDIIO`, porém neste caso é registrado, implementando as rotinas abstratas e, além delas, implementando as seguintes rotinas:

- `SendMessage` : São 3 variantes de um procedimento de envio de mensagens curtas. O usuário deve evitar o uso dessas rotinas, pois as rotinas

a seguir substituem estas (os nomes indicam o tipo de mensagem que é enviado, e os parâmetros já levam o nome de seu conteúdo):

- `NoteOn`;
- `NoteOff`;
- `NoteAftertouch`;
- `ControlChange`;
- `ProgramChange`;
- `ChannelAftertouch`;
- `PitchBend` - Esta rotina já separa o sinal nos seus 14 *bits*.

(IV). Componente `TMIDIIn`

Assim como o componente `TMIDIOut`, o `TMIDIIn` herda o básico de um dispositivo de entrada e saída da classe `TMIDIIO`. Praticamente não apresenta recursos adicionais por não necessitar dos mesmos. Todo o seu funcionamento é análogo ao `TMIDIOut`, a menos do fato de que o `TMIDIIn` recebe eventos e é capaz de executar rotinas dessa forma.

Na API do Windows foi verificado que a entrada MIDI deve ter sua varredura por notas iniciada e terminada, além da conexão efetuada com o dispositivo. Esse fato gerou alguns problemas de desenvolvimento.

A função de *callback* do evento MIDI recebido gera um evento `OnReceive` que pode ser tratado diretamente ao usar o componente. Esse evento já trás consigo a mensagem MIDI recebida descompactada em 4 partes: tipo do efeito, canal, parâmetro MIDI 1, parâmetro MIDI 2.

10.2 O editor de partituras

O primeiro teste foi através de uma interface muito simples em que alguns símbolos musicais são colocados na tela, em diferentes posições e em diferentes *layers*. O objetivo deste teste era colocar diferentes *layers* na tela, a exploração do recurso de transparência, do vetor de *bitmaps* e a familiarização com alguns dos outros componentes básicos do Delphi.

Após o teste com a inserção de diversos *layers* em uma mesma imagem, verificou-se que era vantajoso utilizar um *layer* para cada símbolo musical. Essa vantagem existe pelo fato de a biblioteca utilizada Graphics32 suportar um número de

layers grande o suficiente que garante o funcionamento desejável do aplicativo. Devido a isso, a inserção de notas e símbolos e sua localização é facilitada.

O formato das imagens dispostas na interface gráfica é o *bitmap* pelo fato de a biblioteca Graphics32 possuir várias facilidades com relação a imagens nesse formato além de facilitar a inserção de novos símbolos. Os *bitmaps* utilizados não são redimensionados ao serem são apresentados na tela para evitar operações com os mesmos e, com isso, melhorar o desempenho do aplicativo.

Dessa forma, cada símbolo que deve ser disposto no pentagrama é disposto em um *layer*, seja o símbolo uma nota, um acidente, o símbolo da clave ou o indicativo de *staccato*. Apenas os *layers* de fundo, que representam o pentagrama são opacas. Todos os outros *layers* possuem o fundo transparente.

Foi preferível utilizar um *bitmap* para cada nota ao invés de utilizar figuras básicas como bandeirolas e hastes para facilitar a inserção e remoção de notas em um `SimNotes`. O `SimNotes` é um conjunto de notas simultâneas. Pode ser um acorde (três ou mais notas), bicorde (duas notas), uma nota ou uma pausa. As operações de inserção e remoção não comprometem o desempenho. Como as imagens das figuras rítmicas são carregadas previamente (e não no momento em que cada figura é instanciada), é esperado um aumento de *performance* em relação ao carregamento das imagens durante a instanciação.

O desenvolvimento da interface foi feito baseado na estrutura de dados, com implementação *bottom up*. Primeiramente foi feita a inserção de uma única nota no local correto do pentagrama, com a figura rítmica fixada. No pentagrama, foi fixada a clave de sol e a escala de dó maior, embora ainda não sejam mostradas na tela . Nessa etapa, não foi levada em conta a duração do compasso nem a possibilidade de acordes ou bicordes. A duração da nota (figura rítmica) também foi ficada. Nesse instante foi decidido o tamanho do pentagrama e o dos *bitmaps*, assim como outras variáveis gráficas importantes. A figura 10 mostra duas notas impressas.

Figura 10: Impressão de notas simples

Após a impressão da nota na posição correta, foi impressa uma nota simples, contendo acidente, que deve também ser impresso. A figura 11 mostra um exemplo de nota com acidente. A figura 12 mostra todos os acidentes, com seus respectivos nomes e o bequadro, símbolo utilizado para anular acidentes.

Figura 11: Impressão de notas com acidente

Figura 12: Acidentes e bequadro

Uma regra importante da notação musical é o fato que caso a nota ultrapasse os limites do pentagrama, linhas auxiliares deverão ser desenhadas. A figura 13 mostra exemplos de situação que requer a impressão de linhas auxiliares.

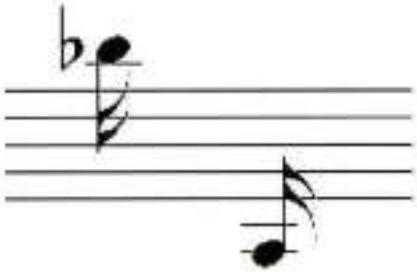

Figura 13: Impressão de notas com linhas auxiliares

Após a impressão correta de uma nota na interface, o passo seguinte consistiu no controle da figura rítmica correspondente, pois até as etapas anteriores a figura rítmica estava fixa. A figura rítmica representa o tempo relativo de duração da nota. A figura 14 apresenta notas de diferentes figuras rítmicas. As figuras rítmicas suportadas pelo Orfeu incluem desde a quartifusa até a máxima e podem estar dispostas com a haste para cima ou para baixo. A figura 15 apresenta todas as figuras rítmicas.

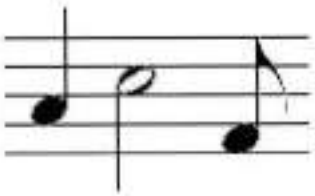

Figura 14: Impressão de notas com duração distinta

Figura 15: As figuras rítmicas

O *software* Orfeu tem suporte para acordes e bicordes. A etapa seguinte consistiu na inserção de várias notas em um `SimNotes`. A figura 16 apresenta exemplos de `SimNotes`.

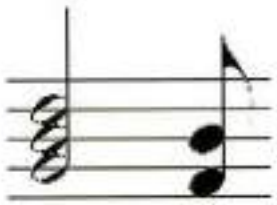

Figura 16: Impressão de acorde e bicorde

Uma característica importante da notação musical em partitura é que na impressão de `SimNotes`, caso haja notas com intervalo de segunda, uma delas deverá ter a cabeça disposta para a esquerda, enquanto que a outra estará disposta para a direita. Isso ocorre para que não haja sobreposição das mesmas. A figura 17 mostra exemplos desse caso particular.

Na edição de músicas, é comum que haja intervalos de tempo em que nenhuma nota é tocada. Esses intervalos são chamados de pausas. O símbolo da pausa

Figura 17: Impressão de acordes que possuem notas em intervalo de segunda

está diretamente ligado ao seu tempo de duração. A figura 18 mostra um fragmento de música em que há `SimNotes` e pausas. A primeira pausa corresponde a uma pausa de colcheia e a segunda, de semínima. A figura 19 as pausas das diferentes figuras rítmicas.

Figura 18: Exemplos de pausas com notas

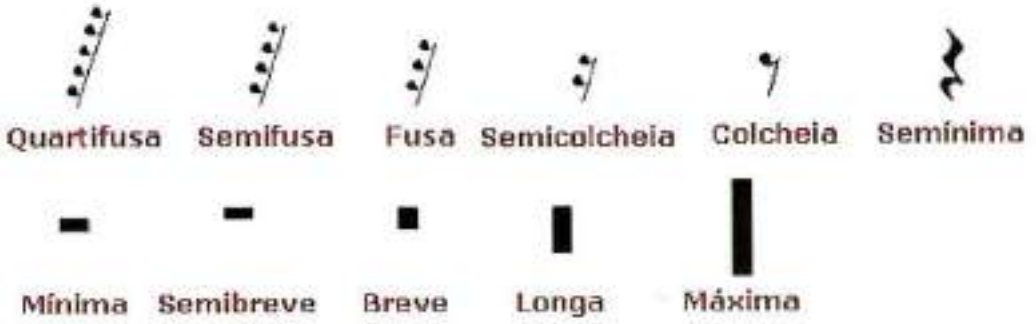

Figura 19: Pausas

Um recurso existente de edição musical é quando ocorre a presença de uma nota de uma certa duração e de mais metade dessa duração. São as chamadas "notas pontuadas". Cada ponto representa a duração de metade do tempo da nota, bicorde ou acorde imediatamente anterior. Ou seja, caso a semínima da figura 20 esteja acompanhada de dois pontos, essa nota terá a duração da semínima somada à duração de colcheia (primeiro ponto) somada à duração de semicolcheia (segundo ponto). O *software* não limita a quantidade de pontos.

Figura 20: Semínima pontuada

Após o tratamento dado a esses pontos, foi necessário incluir e verificação da união da haste, pois dependendo da distância entre as notas do acorde ou bicorde, as mesmas podem ficar separadas. Isso ocorre devido à utilização da figura da semínima para as notas mais graves (quando a haste está disposta para cima) ou mais agudas (quando a haste está disposta para baixo), se a nota tiver duração inferior à da semínima. Caso contrário, a figura da própria nota é utilizada. A figura 21 mostra um exemplo de grande separação de notas em um bicorde.

Figura 21: Grande separação de notas em um bicorde

Com relação à impressão de acordes e bicordes foi encontrado ainda um outro problema: a presença de acidentes em notas próximas. Nesse caso, um dos acidentes estará disposto mais à esquerda que outro, para que não haja sobreposição. Em caso acidentes de notas que estejam em intervalo segunda ou terça, ocorre esse deslocamento. No caso de haver dois intervalos de segunda consecutivos, haverá três locais em que os acidentes serão dispostos. A figura 22 apresenta algumas dessas possibilidades.

Um ornamento suportado pelo Orfeu é o *staccato*. O símbolo que indica essse ornamento, é um ponto abaixo da nota mais grave do `SimNotes`. A figura 23 mostra um exemplo.

Com a impressão completa e correta de acordes e bicordes, a etapa seguinte

Figura 22: Exemplos de acidentes deslocados

Figura 23: Exemplos de bicorde com staccato

foi definida como a impressão de múltiplos acordes, formando um compasso. O compasso corresponde à classe `Measure` do diagrama de classes da figura 3. A união de bandeirolas não estará presente nesta versão do *software*. Apesar de parecer ser apenas a impressão de vários `SimNotes`, um após o outro, é necessário imprimir os números de qualidade e quantidade no início do compasso, caso sejam diferentes do anterior e uma barra vertical que indica seu término. A figura 24 mostra um exemplo a impressão de um compasso completo.

Figura 24: Impressão de um compasso completo

Ao imprimir um compasso, foi verificado que havia um problema a ser resolvido: a impressão dos acidentes quando já houvesse sido impresso um acidente no mesmo compasso. O problema será mostrado com um exemplo. Supondo que a primeira nota do compasso fosse um Lá bemol, essa nota seria impressa juntamente com seu acidente e o mesmo seria válido para todas as notas Lá da mesma oitava, durante aquele compasso. Se a próxima nota tiver esse mesmo acidente, a acidente não deveria ser impresso. Caso a nota possuísse um acidente diferente, seria impresso

naturalmente. E no caso de a nota a ser impressa não possuir acidente? Nesse caso, é utilizado o bequadro para anular o acidente. A figura 25 mostra um exemplo de utilização do bequadro.

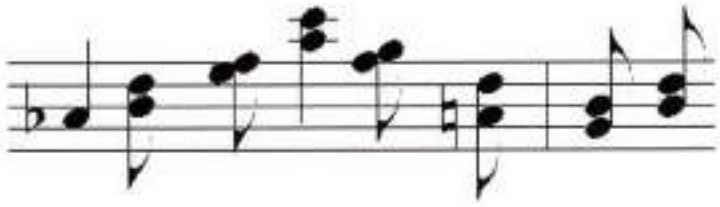

Figura 25: Exemplo de uso do bequadro

A próxima estrutura a ser impressa na tela é a pauta, que corresponde à classe `Staff` do diagrama da figura 3. Além dos compassos, a pauta precisa conter a clave inicial e a escala musical utilizada. A figura 26 apresenta a clave de sol na escala de dó maior.

Figura 26: Impressão parcial da pauta

Quando a impressão da pauta foi realizada, surgiu um certo problema que se resume em como imprimir a música. Continuar sempre na horizontal até o final ou desenhar novo pentagrama e continuar abaixo são as opções (novo sistema). A escolha feita foi fixar um limite horizontal e criar novo sistema. Isso porque a impressão da música será facilitada. A visualização também é favorecida, pelo fato de a divisão em sistemas possibilitar um maior número de compassos visualizados, pela sua disposição na tela. A figura 27 apresenta o mesmo compasso sendo inserido diversas vezes. Nessa figura, estão presentes três sistemas.

Para a impressão da música em vários sistemas foi necessário aumentar o tamanho da figura de acordo com o tamanho da música, para que seu tamanho não seja limitado e ao mesmo tempo não haja um espaço de edição muito grande e inutilizado. O *scrolling* do editor apenas está presente na vertical, pois como já foi dito anteriormente, a largura é fixa para facilitar a impressão.

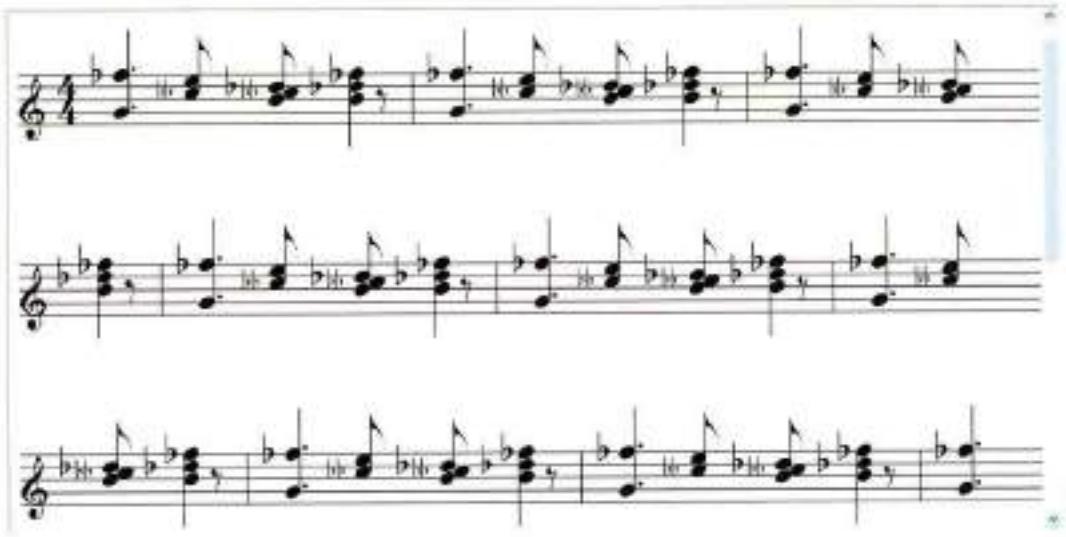

Figura 27: Impressão de vários compassos iguais em três linhas

10.3 O Cursor

O cursor é um dos elementos mais importantes da interface gráfica, pois é através do mesmo que todo o controle de alterações é feito. Há apenas uma instância desse iterador no programa que faz a varredura de todos os componentes da trilha (`Staff`). Através desse componente, é possível recuperar o compasso selecionado, o `SimNotes` e a nota correspondente. Esse componente percorre a trilha, executando as seguintes ações:

- Movimento para cima: no modo partitura, movimenta-se para cima no pentagrama a distância equivalente à metade da distância entre duas linhas (intervalo de segunda), selecionando a respectiva nota.
- Movimento para baixo: mesma movimentação do caso anterior, no sentido contrário.
- Movimento para a direita: cursor seleciona o próximo `SimNotes`.
- Movimento para a esquerda: cursor seleciona o `SimNotes` anterior.
- Primeiro acorde do compasso: cursor retorna até o primeiro `SimNotes` do compasso.
- Último acorde do compasso: cursor avança até o último `SimNotes` do compasso.

- Primeiro acorde: cursor retorna ao primeiro `SimNotes` da música.
- Último acorde: cursor avança ao último `SimNotes` da música.

É importante ressaltar que ao movimentar o cursor, caso haja mudança de seleção do `SimNotes`, o compasso é automaticamente selecionado. Ao criar um novo `SimNotes` ou compasso, o cursor automaticamente é movimentado para selecionar o novo `SimNotes`, pois acredita-se que será o primeiro `SimNotes` a ser editado pelo usuário. Outro fato importante é que quando o modo *player* é acionado, o cursor não se move até que se tenha retornado ao modo de edição.

O cursor possui como atributos ponteiros para o acorde e o compasso selecionado, porém a nota é uma cópia da existente no `SimNotes`. Portanto, ao movimentar o cursor para cima ou para baixo, é feita uma verificação: se a nota existir no acorde, o acidente é atualizado; caso não exista, o acidente será sempre natural. Para aplicações futuras em que haverá escala musical definida, o acidente deverá corresponder ao acidente definido na escala. Essas ações são necessárias para eliminar a infuência de acidentes das notas editadas anteriormente.

No modo tablatura, as funções do cursor de para cima e para baixo apenas mudam-no de corda. A inserção das notas dá-se apenas ao pressionar as teclas numéricas do teclado (no caso de entrada por teclado de computador) ou pela entrada através de instrumento MIDI. No caso da inserção por instrumento MIDI, todas as notas são dispostas na mesma corda. Para encontrar a digitação, o algoritmo de conversão partitura -> tablatura deverá ser utilizado.

10.4 A atualização da tela

O método `UpdateScreen` é utilizado para fazer qualquer tipo de alteração na tela, seja pela edição, pela abertura de arquivos ou pela mudança do modo de visualização (de tablatura para partitura e vice-versa). Há um *flag* chamado `madechange`, que é habilitado toda vez que é feita alguma alteração na tela. É desabilitado quando é lido. Outro *flag* chamado `FileHasChanges` é utilizado para verificar se alguma alteração foi feita pelo usuário. Em caso afirmativo o usuário é questionado se deseja salvar o arquivo.

No modo partitura, a tela é apenas modificada na medida em que há necessidade. Ou seja, se a alteração é feita no último `SimNotes`, apenas o mesmo será

reimpresso. No caso de não ser o último `SimNotes`, porém este fizer parte do último compasso, o compasso inteiro é reimpresso. No caso de ser outro compasso que não o último, a trilha inteira é reimpressa. Isso ocorre para o modo partitura, pois a simples inserção de uma nota ou acidente pode ocasionar o deslocamento de todos os `SimNotes` posteriores.

Por exemplo, caso seja colocado um acidente em uma nota que anteriormente não possuía, supondo que nenhuma outra nota do acorde ou bicorde possua acidente, todo o acorde ou bicorde será deslocado para a direita, bem como seus sucessores. Para contornar esse problema poderia ser adotada uma distância regular entre as notas, porém isso acarretaria perdas consideráveis na estética, principalmente pela possibilidade de ocorrência de acidentes próximos, em que um deslocamento maior ainda é requerido.

No modo tablatura, toda a tela é atualizada a cada alteração. Nesse caso é utilizado o *flag* `madechanges` nesse caso. É importante que a tela inteira seja modificada todas as vezes pois é preciso reimprimir tudo no caso do redimensionamento da mesma. No caso da partitura, não é possível redimensionar a tela. Seja no modo partitura ou tablatura, o cursor é reimpresso qualquer que seja a modificação na tela.

Outro *flag* importante é o de inicialização, que indica quando a tela deverá ser completamente apagada antes da inclusão de qualquer modificação. É muito importante na sua inicialização (modo partitura) e útil para a abertura de arquivos, já que nesse modo a tela só é atualizada caso necessário.

No caso de ser aberta alguma tela auxiliar, como por exemplo para a mudança de fórmula de compasso, provoca a atualização de toda a tela, quando a mesma é fechada, ou seja, quando o *software* volta à tela principal. É preciso que ocorra essa reimpressão porque há casos em que as telas auxiliares modificam a trilha toda.

10.5 A estrutura interna de arquivos

O XML, de acordo com [15], (*eXtensible Language Markup*) é uma linguagem de marcação definida por extensível por permitir que o usuário crie suas próprias *tags*. Seu principal propósito é permitir o compartilhamento de estruturas de dados de diferentes sistemas de informação, principalmente na *internet*. É também baseado em padrões internacionais e largamente utilizado nos sistemas atualmente.

Devido a essas vantagens, foi feita a opção de utilizar o XML para compor a estrutura interna do arquivo. Adicionalmente, o XML permite que posteriormente novos *tags* sejam adicionados à estrutura sem grandes mudanças na estrutura, caso seja necessário, para a implantação de melhorias. A seguir, na figura 28, encontra-se um exemplo de arquivo. A extensão escolhida para o formato de arquivo é .orfeu.

```
<music name="Teste" author="Danilo">
  <staff name="Guitar">
    <tabstrings>
      <note>G7</note>
      <note>B2</note>
      <note>D#4</note>
    </tabstrings>
    <measure quantity="4" quality="4">
      <simnotes rhythm="16">
        <!- All octaves must be in American format -->
        <note>C#5</note>
      </simnotes>
      <simnotes rhythm="64">
        <note>B4</note>
      </simnotes>
    </measure>
  </staff>
  <staff name="Guitar 2">
    <measure quantity="4" quality="4">
      <simnotes rhythm="8">
        <!- All octaves must be in American format -->
        <note>F#4</note>
        <note tabstring="3">A#4</note>
      </simnotes>
      <simnotes rhythm="128">
         <note>B4</note>
      </simnotes>
    </measure>
  </staff>
</music>
```

Figura 28: Exemplo de arquivo .orfeu

Como é possível observar, a música é identificada pelo nome e autor. Após sua identificação, é definida uma pauta. A pauta, por sua vez, contém a afinação de cada uma das cordas. Essa informação é importante no editor de tablaturas. Essa definição não é obrigatória, mas caso exista, é necessário que seja o primeiro atributo da pauta. É utilizada apenas no modo tablatura.

Depois da afinação das cordas, as estruturas seguintes são os compassos. Na

definição do compasso, há os atributos qualidade e quantidade próprios dos mesmos. Abaixo do compasso, tem-se os `SimNotes`, onde o ritmo é definido. por último, encontram-se as notas, necessariamente no formato americano (constituído das letras A a G).

Um arquivo útil para o usuário é o `tuning.ini`, que armazena as afinações gravadas pelo usuário na biblioteca. Adiante, na parte referente aos *menus* é explicado como salvar uma afinação. A afinação *default* é a *Default Guitar EBGDAE*, que não está nesse arquivo por ser o padrão. A seguir (figura 29), encontra-se um exemplo de arquivo `tuning.ini`.

```
$[$Half step down EbBbGbDbAbEb$]$
Note0=D\#5
Note1=A\#4
Note2=F\#4
Note3=C\#4
Note4=G\#3
Note5=D\#3
$[$Dropped D EBGDAD$]$
Note0=E5
Note1=B4
Note2=G4
Note3=D4
Note4=A3
Note5=D3
$[$Bass 4 strings GDAE$]$
Note0=G3
Note1=D3
Note2=A2
Note3=E2
```

Figura 29: Exemplo de arquivo tuning.ini

10.6 Interface com o usuário

O principal meio de entrada do usuário é o teclado do computador. Foram criados atalhos para tornar a edição mais fácil e rápida. Alguns atalhos já existentes em outros editores foram mantidos, pelo fato de o usuário já estar acostumado aos mesmos, porém alguns foram melhorados. A tabela 4 mostra os atalhos e as ações correspondentes.

A interface com o usuário possui basicamente dois modos, tanto para o editor

Tabela 4: Tabela de atalhos do teclado

Atalho	Ação
Esc	Pára a execução sonora
Space	Inicia/pára a execução sonora
Backspace	Se for pausa, apaga a pausa; caso contrário apaga somente a nota; anda com o cursor para trás
Enter	Se a nota não existir, coloca-a; caso contrário, apaga-a
End	Cursor retorna para o último `SimNotes` da música
Ctrl+End	Cursor retorna para o último `SimNotes` do compasso
Home	Cursor vai para o primeiro `SimNotes` da música
Ctrl+Home	Cursor vai para o primeiro `SimNotes` do compasso
Left Arrow	Cursor retorna para o `SimNotes` anterior
Up Arrow	Cursor sobe
ShiftUp Arrow	Aumenta o acidente (mais aguda) no editor de partitura; no de tablatura, passa a nota selecionada para a corda de cima
Right Arrow	Cursor vai para o `SimNotes` seguinte
Down Arrow	Cursor desce
Shift+Down Arrow	Diminui o acidente (mais grave) (partitura); passa a nota selecionada para a corda de baixo (tablatura)
Ctrl+Shift+Ins	Insere novo compasso antes
Ctrl+Ins	Insere novo compasso após
Shift+Ins	Insere novo `SimNotes` depois
Ins	Insere novo `SimNotes` antes
Ctrl+Shift+Del	Apaga o compasso
Ctrl+Del	Se o compasso estiver vazio, apaga-o; caso contrário, apaga a nota
Shift+Del	Apaga o `SimNotes`
Del	Se o acorde for uma pausa, apaga-o; caso contrário, apaga a nota
2-9	no editor de partituras, insere a nota com esse intervalo ascendentemente
Shift+(2-9)	no editor de partituras, insere a nota com esse intervalo descendentemente
A-G	no editor de partituras, insere a nota com esse nome ascendentemente ou na posição cursor
Shift+(A-G)	no editor de partituras, insere a nota com esse nome descendentemente ou na posição cursor
R	Transforma o acorde apontado em pausa
"=/+"	Dobra a duração da nota
"-/_"	Divide por 2 a duração da nota
Ctrl+"."	Coloca ou tira *staccato*
Shift+"./>"	Aumenta um ponto na nota
"./>"	Retira os pontos se a nota houver; caso contrário, insere um
X	Inverte a direção da haste

de tablaturas quanto para o de partituras. O primeiro é o de edição, em que notas, `SimNotes` e compassos são inseridos. O segundo é o *player*, no qual o usuário é capaz de ouvir a melodia editada. É importante salientar que o modo *player* possui um cursor para cada pauta e não interfere no cursor do modo de edição.

10.7 O editor de tablatura

A tablatura é uma notação musical largamente utilizada, principalmente para instrumentos de cordas, pela fácil associação entre as notas a serem tocadas e as cordas em que se encontram. A figura 30 apresenta uma tablatura para instrumento de 6 cordas, como por exemplo a guitarra.

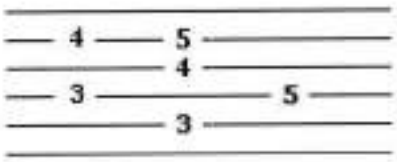

Figura 30: Editor de tablatura

No editor de tablatura, o número representa o intervalo entre a nota a ser tocada e a afinação da corda. Nesse editor é possível colocar qualquer número de um ou dois dígitos. Para que possam ser aceitos tanto números de um dígito quanto de dois, foi utilizado um *timer* para esperar a entrada do segundo dígito. No caso de o usuário não entrar com o segundo dígito dentro do tempo, o número é entendido como sendo de apenas um dígito.

No âmbito geral, o editor de tablatura é mais simples que o editor de tablaturas. O seu desenvolvimento tornou-se ainda mais rápido pelo fato de grande parte dos problemas decorrentes da edição já terem sido tratados durante o desenvolvimento do editor de partituras, os atalhos já terem sido definidos e até mesmo porque foi necessário apenas adicionar alguns atributos e métodos em determinadas classes.

Uma observação importante a er feita, no entanto, é que o editor de partitura pode ser redimensionado, enquanto que essa operação não é permitida na primeira versão do *software*, para o editor de partituras. Outra diferença é que quando é utilizado este editor, a tela toda é atualizada, independente da modificação.

10.8 A tela principal e os *menus*

A tela principal é composta do local em que é apresentado o editor de partituras ou tablaturas e de *menus*, para a realização das funções que serão explicitadas a seguir. É importante salientar que os editores de tablatura e partitura não são dispostos na tela simultaneamente e cada trilha é mostrada por vez. Nesta primeira versão, a trilha corresponde à pauta.

No *menu* Arquivo (`File`) têm-se as funções básicas de manipulação de arquivos:

- Novo
- Abrir
- Salvar
- Salvar como ...
- Importar arquivo do Guitar Pro
- Exportar para MIDI
- Sair

No *menu* Editar (`Edit`) têm-se as seguintes funções:

- Conversão adaptativa de partitura para tablatura (`Find fingering`)
- Transposição (`Transpose`)
- Edição da fórmula de compasso (`Time signature`)
- Propriedades (`Music properties`)
- Configuração

A transposição é feita por intervalo. A figura 31 ilustra a tela de transposição. A mudança da fórmula de compasso pode ser realizada através da tela mostrada através da figura 32, ao ser selecionada essa opção no *menu*. O item "Propriedades" mostra algumas propriedades da música, como o nome e o andamento. No item "Configuração" está incluída a configuração de entrada MIDI e também a de saída.

No *menu* Trilhas (`Tracks`) têm-se as funções relativas às trilhas, que são:

Figura 31: Exemplo de transposição

Figura 32: Edição da fórmula de compasso

- Nova trilha
- Apagar trilha
- Editar trilha
- Selecionar trilha
- Ver como tablatura

A função "Selecionar trilha" altera a trilha mostrada no editor. Uma janela é aberta com as possíveis trilhas a serem selecionadas, que pode ser vista através da

figura 33. A função "Ver como tablatura" altera o modo de edição partitura/tablatura. Ao criar uma nova trilha (ou editar alguma já existente), uma janela similar à da figura 34 é disposta, para que o usuário possa configurá-la.

Figura 33: Seleção de trilha

Figura 34: Criação/edição de trilha

Através da figura 34 pode-se observar que é possível configurar os detalhes MIDI: volume, panorama, (que é a distribuição de quanto para a esquerda ou para a direita está o som), porta e canal. No canto inferior esquerdo tem-se a uma lista de intrumentos para que o usuário possa escolher o timbre. É possível selecionar a notação de nomes/oitavas adotadas (padrão norte-americano ou europeu/brasileiro). Há

algumas afinações que são padrões, definidas na biblioteca, porém o usuário pode definir outra afinação e salvá-la para uso posterior. A configuração das trilhas é bastante flexível e permite ao usuário adequá-la às suas necessidades.

No *menu* (`Tracks`) têm-se as seguintes funções:

- Lista de atalhos (`Key shortcut list`)
- (`About`)

A lista de atalhos apresenta todos os atalhos do teclado e o último item fornece informações sobre os autores, data e versão.

10.9 Implementação da estrutura musical

O diagrama de classes da figura 3 apresenta a estrutura completa de uma música. Entretanto, para a primeira versão, essa estrutura foi simplificada para que as funcionalidades mais importantes estivessem disponíveis nesta versão. A figura 35 mostra o diagrama de classes utilizado na implementação. Os atributos e métodos de cada classe estão descritos a seguir.

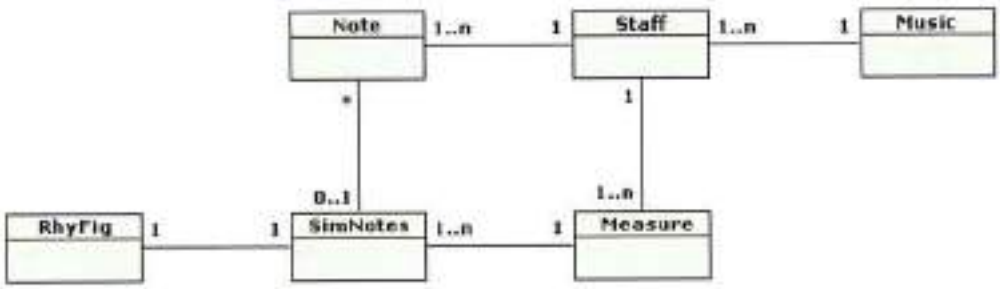

Figura 35: Diagrama de classe da primeira versão

Ao comparar os diagramas de classe das figuras 3 e 35 é possível observar que houve mudança de cardinalidade dos relacionamentos. Isso se deve à forma com que o editor foi implementado, que não permite que haja uma trilha (`Staff`) sem compassos (`Measure`) ou um compasso sem (`SimNotes`). No caso do compasso, sempre haverá um `SimNotes`, mesmo que seja uma pausa (`SimNotes` vazio). E no caso da trilha, esta deve conter pelo menos um compasso que contém uma pausa.

No caso do `SimNotes`, é possível que este não possua nota (*Note*) alguma, o que caracteriza uma pausa. Entretanto, o relacionamento que há entre a trilha e a nota (responsável pela afinação das cordas) também não pode possuir cardinalidade 0, pois é suposto que um intrumento de cordas tenha pelo menos uma corda. O editor de tablaturas garante que haja pelo menos uma corda.

A seguir estão explicitados os atributos e métodos das classes contidas no diagrama da figura 35, relacionados com a estrutura musical.

(I). `Note`

- Atributos
  - Nota propriamente dita
  - Oitava
  - Acidente
  - Corda (específico para tablatura)
  - Número MIDI
- Métodos
  - Obter o nome completo: nota, oitava e acidente
  - Transpor um intervalo - parâmetro: intervalo
  - Transpor semitons - parâmetro: inteiro
  - Encontrar o intervalo - parâmetro: nota
  - Leitura da nota do arquivo XML
  - Copia nota
  - Aumenta acidente (nota mais aguda)
  - Diminui acidente (nota mais grave)

(II). Figura rítmica

- Atributos
  - Tipo de figura
  - Número de pontos
- Métodos
  - Dobra o tempo
  - Reduz o tempo pela metade

(III). *SimNotes*

- Atributos
  - Lista de notas
  - Quantidade de notas
  - Figura rítmica
  - Haste (para cima ou para baixo)
  - *Staccato*
- Métodos
  - Insere nota
  - Apaga nota
  - Apaga todas as notas
  - Carrega do arquivo XML
  - Grava em arquivo XML
  - Verifica se é pausa
  - Imprime `SimNotes` na partitura
  - Atualiza `SimNotes` na partitura
  - Imprime `SimNotes` na tablatura
  - Atualiza `SimNotes` na tablatura

(IV). Compasso

- Atributos
  - Lista de `SimNotes`
  - Quantidade de `SimNotes`
  - Quantidade
  - Qualidade
- Métodos
  - Insere `SimNotes`
  - Insere `SimNotes` vazio
  - Apaga nota
  - Está vazio
  - Carrega de arquivo XML
  - Grava em arquivo XML

- Imprime compasso na partitura
- Atualiza compasso na partitura
- Imprime compasso na tablatura
- Atualiza compasso na tablatura

(V). Pauta ou trilha

- Atributos
  - Nome
  - Lista de compassos
  - Quantidade de compassos
  - Volume
  - Panorama
  - Canal
  - Instrumento
  - Lista de notas (afinação das cordas, para a tablatura)
  - Quantidade de cordas
- Métodos
  - Insere trilha
  - Insere trilha vazia
  - Apaga trilha
  - Carrega do arquivo XML
  - Grava em arquivo XML
  - Insere corda
  - Apaga corda
  - Apaga todas as cordas
  - Está na afinação *default*
  - Imprime trilha na partitura
  - Atualiza trilha na partitura
  - Imprime trilha na tablatura
  - Atualiza trilha na tablatura

(VI). Música

- Atributos

  - Nome
  - Autor
  - Lista de trilhas
  - Quantidade de trilhas
- Métodos
  - Insere trilha
  - Insere trilha vazia
  - Apaga trilha
  - Apaga todas as trilhas
  - Carrega do arquivo XML
  - Grava em arquivo XML
  - Troca de trilha

10.10 Execução sonora

A execução sonora do Orfeu possui uma liberdade bastante grande quanto à polifonia e à polirritmia. A música pode ser armazenada com total assincronia entre as trilhas, porém enquanto houver proporcionalidade entre as figuras rítmicas básicas, a execução sonora é capaz de executar o som da maneira desejada.

A classe que implementa a saída MIDI é a `TMusicPlayer`, herdeira da `TThread`, cujo construtor `Play` já a executa e em sua parada chama um método de parada sonora e atualização da tela (evento `OnTerminate`). Esse `thread`, ao terminar, já é destruído da memória. Logo o ponteiro para ele é apenas utilizado na chamada do método `Terminate` (herdado), que solicita a parada.

Durante a criação através do construtor `Play`, o objeto sendo construído basicamente executa:

(I). Cria um *thread* inicialmente suspenso

(II). Inicializa os dados (entre as inicializações, cria um cursor para cada trilha)

(III). Encontra o tempo que teria passado até a posição do cursor, considerando que no instante inicial tem-se T = 0. Esse resultado é o valor relativo do instante inicial e deve ser negativo ou zero, pois o instante inicial da partitura está ou junto ou antes da posição inicial

(IV). Inicializa os cursores em posições para que a próxima `SimNotes` esteja depois do início da execução

(V). Inicia a saída sonora

(VI). Inicia o *thread* que estava suspenso

Durante o construtor não foi utilizada mudança na prioridade de execução do *thread*, pois essa mudança retira excessivamente o controle do usuário sobre o *thread* principal.

O núcleo da execução está no método `Execute`. Esse método é executado no sexto passo do construtor e deve ficar em um laço que verifica se o *flag* `Terminated` foi alterado. Sua lógica consiste basicamente em:

(I). Definir o instrumento, volume e panorama de cada trilha no canal adequado

(II). Obter o valor absoluto do instante inicial para cada trilha

(III). Executar as primeiras notas de cada trilha

(IV). Repetir bloco de execução (figura 36) até `Terminated`

```
TickCount := Instante atual
Para cada Cursor faça
  Se esse Cursor não terminou
  SIM: Se instante início próxima nota apontada <= TickCount
    Pare de tocar o SimNotes
    Se a trilha terminou
    SIM: Termine o Cursor
      Se todos os cursores terminaram, Terminate
  Se esse Cursor não terminou
  SIM: Se instante início próxima nota apontada <= TickCount
    SIM: Vai para a próxima nota e a toca
      Atualiza instante final da nota e inicial da próxima
```

Figura 36: Bloco de execução

Todo o tempo é calculado em *ticks*, dado em milisegundos. Para encontrar o tempo em *ticks* de uma figura e de compasso genérico, há métodos para obter esse valor.

Essa é uma técnica de *wait for flag* na execução sonora. A opção de usar *um wait for flag* decorre do fato que a precisão de obter o intante atual é suficiente e mesmo que haja alguma variação no atraso da execução, a pulsação sempre é mantida. As falhas de ritmo são menores que 10ms, imperceptíveis ao ouvido humano. A menor percepção é a de variações de atraso de 1/20kHz = 50ms utilizando o fato de que a maior freqüência perceptível é de 20kHz. Dessa forma é garantido que este *thread* funciona e que não foi necessário usar o *timer* multimídia `TMMTimer` para a execução sonora.

10.11 Entrada MIDI

O teclado MIDI foi utilizado como elemento opcional para entrada de dados no aplicativo. Essa entrada não é suficiente para fazer toda a transcrição do conteúdo musical, exigindo que o usuário use o teclado do computador simultaneamente.

O efeito do funcionamento do teclado MIDI é habilitado desde a construção até a destruição da janela principal, garantindo que quando algo for pressionado a estrutura de dados é atualizada. Foi tomado o cuidado de evitar que seja feita alguma mudança no modo de execução, pois possibilitaria que *threads* simultâneos pudessem ler o mesmo dado.

Ao pressionar uma nota, acorde ou bicorde, as notas da posição atual do cursor são substituídas pelas notas executadas. Como as mensagens MIDI enviam nota por nota, significa que há um critério para selecionar quando as notas devem ser apagadas. Por padrão, quando uma nota é pressionada e nenhuma outra nota estava sendo pressionada antes, o `SimNotes` é apagado. Dessa forma é possível colocar o acorde nota por nota, bastando não soltar uma delas enquanto outra é colocada. A tela necessita ser atualizada quando alguma nota é pressionada.

11 Algoritmo baseado em autômato adaptativo

11.1 Conversão de partitura em tablatura

A estrutura de dados da partitura e da partitura é basicamente a mesma. Uma música é dividida de sucessivamente até que se chegue aos compassos e finalmente às notas. Para oeste algoritmo, a informação de como os dados foram estruturados é irrelevante: o que importa é apenas a maneira de representação das notas. As notas em partitura são identificadas por um conjunto de três atributos: nome, acidente e oitava, enquanto as notas em tablatura são identificadas por dois atributos: corda e posição. Porém, apesar de aparentemente as tablaturas serem mais simples de representar, elas exigem que haja o conhecimento prévio do número de cordas do instrumento e da afinação de cada corda, dentro da pauta em análise.

Fixando-se uma corda, pode-se facilmente converter a nota da partitura em uma nota equivalente de tablatura. Na tablatura cada número denota a diferença, em semitons, entre a nota representada e a nota da afinação da nota correspondente, sendo esse número necessariamente positivo para que seja possível a execução no instrumento (números negativos evidenciam condições impossíveis para o instrumento dado na corda escolhida). Se for adotada a numeração usual de composição dodecafônica para as notas, disposta na tabela 5 a seguir:

Tabela 5: Numeração usual de composição dodecafônica para as notas

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	1	2	3	4	5	6	7	8	9	10	11

E se somarmos o número representado na tablatura (`TNote.Number`), ao número da oitava (`TNote.NumOct`) multiplicado por 12, será obtida a distância em semitons com relação à nota C0. Para obter a distância entre duas notas quaisquer, basta calcular a distância entre as duas notas, subtraindo-se da distância da nota mais aguda em relação a C0, a distância da nota mais grave em relação a C0 (o inverso daria um número negativo). É fácil concluir que existe um número inteiro para cada nota que possa ser tocada em uma corda com uma afinação qualquer, porém esse número pode ser negativo ou excessivamente grande. A faixa de valores permitida será sempre de 0 a 99. Valores fora dessa faixa serão considerados inexistentes pelo algoritmo.

Resumidamente, na tablatura, em cada pauta (`TStaff`) são representadas as cordas com suas respectivas afinações, e as notas (`TNote`) possuem características

específicas para trabalhar com tablaturas, são essas o método `tabFretNumber`, responsável por obter o número que constará na tablatura (posição) e o `tabString`, um número inteiro que indica a corda em que a nota está colocada. Esse número sempre deve estar na faixa de 0 a (número de cordas - 1), mesmo que force a ficar sobreposto a outra nota na mesma corda ou seja um número fora da faixa de 0 a 99 (características alheias ao algoritmo em questão).

É dado o nome de "digitação" ao conjunto de posições encontradas pelo algoritmo. O algoritmo não é capaz de converter de uma forma ideal qualquer partitura em tablatura. O único critério utilizado foi: Em uma melodia, as distâncias entre de notas adjacentes devem ser as menores possíveis.

A partir desse critério é possível criar diversos algoritmos de conversão, porém o uso de um algoritmo baseado em autômatos adaptativos mostrou-se adequado para efetuar tal conversão. O algoritmo instancia um único autômato, e efetua o processamento da entrada do autômato para as várias fitas de entrada, como será mostrado a seguir.

A música é dividida em partes para que seja possível a conversão. O autômato trabalha apenas com trechos puramente melódicos e contínuos, ou seja, cada pausa ou cada acorde é tratado como separação entre trechos melódicos, e cada trecho melódico é tratado pelo autômato de forma isolada. Cada um desses trechos melódicos corresponde a uma das instâncias do autômato.

As pausas não necessitam de nenhum tratamento, visto que não há notas a serem colocadas em seus espaços. Os acordes e bicordes não são tratados por autômatos adaptativos, e sim por alguma lógica que tenta apenas dispor os mesmos de uma forma possível, usando como critério básico que cada corda deve ter uma ou nenhuma nota com posição na faixa de 0 a 99, inicialmente.

Trabalhando com cada melodia, pode-se obter para cada corda uma seqüência de números que indicam como a melodia seria tocada no instrumento utilizando apenas essa corda, por exemplo, para um baixo com afinação padrão (GDAE), pode-se ter a melodia representada da seguinte forma, contida na tabela 6 (notas da melodia de exemplo: início de Greensleaves):

Cada uma das linhas de cada melodia é considerada uma fita utilizada pelo autômato. O autômato lê apenas um valor por vez e se adapta à melodia mais adequada. O autômato tem sempre um estado representando cada corda do instrumento e um estado de troca de corda, que indica que para nota seguinte não é adequada aquela

Tabela 6: Notas de uma melodia exemplo

Afinação / Melodia	A3	C4	D4	E4	F4	E4	F4
G3	2	5	7	9	10	9	7
D3	7	10	12	14	15	14	12
A2	12	15	17	19	20	19	17
E2	17	20	22	24	25	24	22

corda em que foi colocada a nota anterior. O autômato fica então da seguinte forma (figura 37):

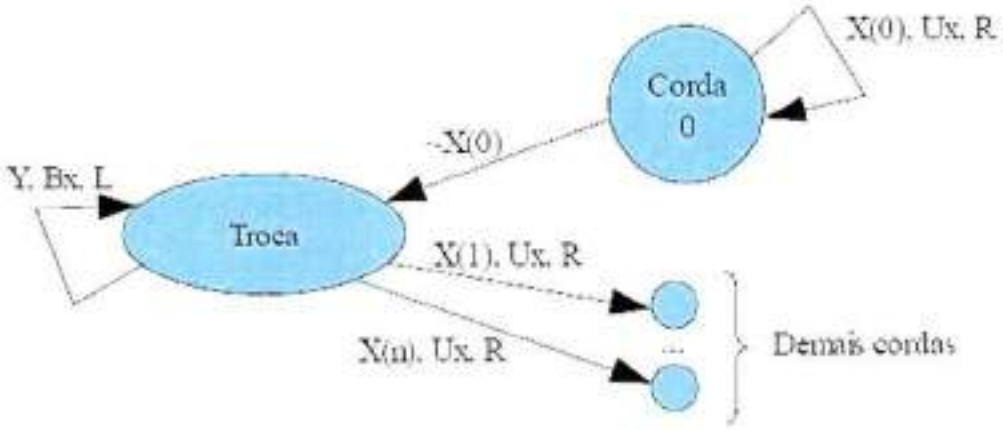

Figura 37: Autômato adaptativo

X(i) e ~X(i) representam conjuntos complementares de possibilidades de entrada utilizando a i-ésima fita. Esse conjunto é modificado a partir das ações adaptativas, como se houvesse a remoção das transições do Y corrente e a inserção das transições de um novo X, efetuado de maneira mais direta. A condição Y é o produtório (considerando o produto como conector lógico"e") de ~X(i) para todo i possível, ou seja, é a saída padrão que ocorre quando nenhuma outra transição está habilitada.

Ux e Bx representam ações adaptativas que atualizam as transições conforme será visto a seguir. Todas as ações adaptativas são realizadas após a transição.

L e R representam o movimento do ponteiro das fitas para a esquerda e para a direita, respectivamente, baseando-se no fato de que a primeira nota da parte analisada da música está sempre na extremidade esquerda da fita e é sempre a primeira a ser apontada (as fitas nunca são consumidas, há apenas um deslocamento do ponteiro sobre elas, que inicia sobre a nota inicial de cada parte melódica). A transição de um estado

de uma corda para o estado de troca não movimenta o ponteiro das fitas.

Cada ação com R atribui o número da corda como saída, mas cada ação com L indica que essa saída era incorreta, retirando esse número da saída. Isso pode ser feito imaginando-se uma fita adicional "de saída", com números que indicam a fita escolhida para aquela nota. A tabela 7 apresenta os números na fita de saída.

Tabela 7: A saída indica a fita mais adequada

Afinação / Melodia	A3	C4	D4	E4	F4	E4	F4
G3 (Fita 0)	2	5	7	9	10	9	7
D3 (Fita 1)	7	10	12	14	15	14	12
A2 (Fita 2)	12	15	17	19	20	19	17
E2 (Fita 3)	17	20	22	24	25	24	22
Saída	2	1	1	0	0	0	1

Todo estado do autômato que representa uma corda possui transições exatamente como apresentado na figura 37, porém não foram representadas na mesma as voltas, por serem redundantes e para evitar poluir a imagem. O estado inicial será a corda zero, e no caso de haver mais de uma transição quando no estado de troca, o próximo estado será o da corda de menor valor, ainda não visitada. Para saber qual é essa corda, basta saber que esse valor deve ser maior que a saída anteriormente utilizada, e que os valores de saída iniciais devem ser sempre -1 (inicialização da fita de saída), assim como os valores impostos ao se transitar para a esquerda na fita. Na realidade da implementação, esses -1 não precisam ser implementados e como só os valores mais altos da saída são lidos, essa fita pode ser implementada como uma pilha (*outputTape*). Isso ficará mais claro com as ações adaptativas.

11.2 Aceitação e rejeição da cadeia

Cada símbolo de entrada é um número de 0 a 99 ou algum símbolo que indique que a entrada é inválida, por fornecer um valor fora da faixa explicitada ou por não ser uma nota única (podendo ser uma pausa, um bicorde ou um acorde).

A aceitação ocorrerá quando estivermos no estado de alguma corda e esse estado receber a entrada correspondente a uma posição na fita em que nenhuma das cordas terá um valor válido, ou ainda, se no término da fita (final da música), transitando para um estado ainda não explicitado na figura acima, de aceitação da cadeia. Essa transição tem prioridade sobre a transição habilitada com ~X(i).

Caso o autômato encontre um acorde ou bicorde, é chamada uma rotina de tratamento de acordes/bicordes, pois os mesmos não são tratados pelo autômato adaptativo da versão inicial. O mesmo ocorre com as pausas.

A rejeição da cadeia ocorrerá se chegar ao estado de troca, tentando chegar a uma posição anterior à inicial (única transição para a esquerda ocorrendo quando estamos na primeira nota). Isto significa que todas as possibilidades de conversão foram feitas para os dados critérios (implícitos nas ações adaptativas), e que o processo deverá ser reiniciado para critérios menos rigorosos.

As transições de aceitação e rejeição não alteram a fita de saída e não efetuam nenhuma ação adaptativa.

Para simplificar a visualização, o autômato, para 3 cordas, é semelhante à figura 38 (os textos sobre as transições não foram colocados para não poluir a imagem). Os estados 0, 1 e 2 correspondem às cordas e T é o estado de troca. O estado mais à direita é o estado de aceitação.

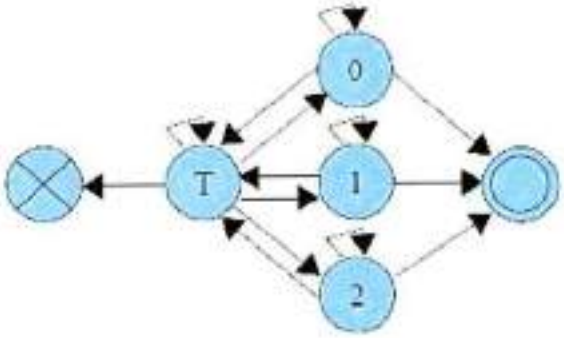

Figura 38: Visualização do autômato para 3 cordas

11.3 Ações adaptativas e algoritmo

Todo o núcleo do funcionamento desse algoritmo está nas ações adaptativas. Essas ações visam a obedecer ao critério previamente estabelecido de que as distâncias entre as posições devem ser as menores possíveis. Distâncias de 3 posições serão consideradas sempre possíveis de serem executadas (pois supondo que o músico tenha 4 dedos, todas as notas são possíveis de serem mapeadas em um dedo a partir do número da posição), e portanto esse será o número inicial. Se o autômato conseguir achar uma digitação para a parte com esse número como faixa, essa será a utilizada,

senão ele tenta com 4 posições e assim em diante. O algoritmo encontra-se na figura 39, visto em alto nível, em uma pseudo-linguagem:

```
Repetir
  Faixa de distâncias := 3
  Enquanto (Executa o autômato) não retorna uma aceitação
    Faixa de distâncias := Faixa de distâncias + 1
  Impõe a fita de saída: cordas utilizadas para a parte
Até acabar a música
```

Figura 39: Algoritmo em pseudo-linguagem

Essa faixa de distâncias é importante para o autômato porque controla as ações adaptativas que restringem a faixa de valores que limita os valores de todos os X vistos na figura 38.

Para imaginar o que faz a ação Ux (`Update` X), seja uma lista com apenas os últimos valores usados da melodia, por exemplo, se há uma melodia com 12 notas e já foi escolhida a digitação de 11 notas, não seria adequado retornar ao início devido ao aumento da faixa de distâncias na décima segunda nota. Ou seja, há uma janela de notas que será considerada para a verificação da faixa de distâncias, e o tamanho dessa janela, nesta primeira versão, é ficado como 7, valor estimado durante a elaboração do algoritmo, por ser um valor pouco provável para a faixa de distâncias, e imaginando o que aconteceria com todas as notas na mesma corda se esse valor fosse menor, a conclusão é que o valor não deve ser muito maior que 7 nem muito menor. Melhores ajustes para esse valor (quantidade de notas na janela) podem fazer parte de projetos futuros de otimização.

Explicitando o esperado a partir do que fora dito, a ação Ux atualiza a função X indicando a faixa de valores permitidos para a posição da nota na corda. Essa faixa pode ser facilmente calculada tendo todos os valores da janela de posições usadas, bastando obter o valor máximo e mínimo dentro da janela, é obtido o valor máximo e mínimo de X como sendo:

```
MinX = MinJanela - FaixaDistâncias + (MaxJanela - MinJanela)
MaxX = MaxJanela + FaixaDistâncias - (MaxJanela - MinJanela)
```

A função X(i) nada mais é que uma função que retorna verdadeiro quando a posição da nota na i-ésima corda estiver dentro da faixa [MinX;MaxX], e falso caso contrário.

Para armazenar os valores da janela há uma pilha de valores inteiros, que são justamente o resultado da conversão. Essa pilha (*fretStack*) conterá todos os valores descobertos, porém apenas os da janela (ou do topo) são utilizados para atualizar o MinX e o MaxX.

A ação Ux, portanto, apenas insere o novo valor na pilha *fretStack* e atualiza os valores de MinX e MaxX, conforme visto acima. Isso é equivalente à retirada de transições que estavam na faixa de MinX a MaxX e que não estão mais, e a inserção de transições que a partir de agora estão nessa faixa.

A ação Bx (*Backup* X) realiza basicamente o mesmo Ux, porém ao invés de inserir um valor na pilha `fretStack` ela desempilha um valor (se possível). Quando a pilha já estiver vazia, essa ação cria uma transição para o estado de rejeição, com prioridade sobre as demais e sem restrição para transitar (já habilitada, por cadeia vazia). Na realidade isso é feito apenas ativando um *flag* que indica que a ação Bx ativou a transição de rejeição.

Adicionalmente, Bx deve armazenar o valor da última transição feita (em `lastTransition`), para evitar *loops*, ou seja, se da última vez o X(i) foi usado então a última transição feita foi para o estado da corda 1, esse valor deve ser armazenado e o Ux deve apagá-lo (com -1). Dessa forma a varredura feita quando estiver no estado de transição será feita de *lastTransition* + 1 em diante, o que significa que aos poucos certas transições são apagadas por Bx durante a execução e recolocadas por Ux.

11.4 Tratamento de notas múltiplas

Como já foi dito, o tratamento de notas múltiplas é feito de forma direta, sem utilizar o autômato adaptativo. Essa lógica (`findMultiFingering`) consiste simplificadamente em ordenar as cordas do instrumento da mais aguda até a mais grave, e aproveitar o fato dessa ordenação já existir na `SimNotes` naturalmente. A partir daí é verificado se a nota mais grave é possível de ser tocada na corda mais grave, caso contrário, outras cordas são selecionadas até que seja encontrada uma que possibilite tocar a nota. Em seguida é verificada a segunda nota mais grave, e assim em diante, até completar todo o acorde.

12 Importar arquivo do Guitar Pro 4

12.1 Formato de arquivos do Guitar Pro

O formato de arquivo do Guitar pro é dividido em 3 partes: cabeçalho, corpo (compassos, trilhas e depois *beats*, equivalentes aos *SimNotes*) e diagramas de acorde, conforme pode ser visto em [14], porém alguns detalhes foram omitidos. Infelizmente o formato desse arquivo é totalmente sequencial e obriga a leitura de todas as informações, mesmo irrelevantes, até que se chegue nas informações que serão importadas. O texto a seguir explica o formato do arquivo do Guitar Pro e como é realizada a conversão para a estrutura de dados do Orfeu. Toda a conversão é realizada através do método do `FormMain` que é chamado no *menu* de importação.

As *strings* que iniciam por um inteiro têm um *byte* com o número de *bytes* que acompanha, por exemplo (em hexadecimal):

```
05 00 00 00 04 41 41 41 41
```

Essa é uma *string* iniciada por inteiro. Os primeiros 4 *bytes* são um número inteiro (como o LSB vem antes do MSB, o número representado é 5), depois tem-se um "*character string*" como é dito pelo texto do site já citado, ou seja, tem-se um *byte* que indica o tamanho do texto (4) e depois dele, o texto ("AAAA"). Para *strings* maiores que 255 caracteres, não há uma explicação do que acontece, logo o sistema apenas captura o texto que tem dimensão indicada pelo quinto *byte* e ignora o restante do espaço, saltando corretamente de acordo com o inteiro para chegar à próxima parte.

Todo número inteiro ocupa 4 *bytes* no texto a seguir, e sempre que for dito "*string*" o texto se refere a um *byte* contendo o tamanho (pode ser zero) seguido pelo conteúdo do mesmo. *Strings* iniciadas por inteiros podem ser vazias, mas como é necessário que haja o inteiro e a *string*, essas são apresentadas como:

```
01 00 00 00 00
```

o que significa que há um inteiro com o valor 1 indicando que toda a *string* possui apenas 1 *byte*, e a *string* em si é o *byte* nulo, indicando que o tamanho da *string* é nulo. Apesar de parecer irrelevante armazenar duas vezes o tamanho, isso é feito para assegurar o armazenamento de *strings* longas (maiores que 255 caracteres), embora não seja algo relevante para o projeto.

A numeração dos *bits* de um *byte* é feita do *bit* menos significativo (0) ao *bit*

mais significativo (7).

O arquivo do Guitar Pro define uma linguagem. Pode ser escrita em alto nível como um conjunto de partes como está dividido o texto a seguir. Essas partes são lidas seqüencialmente, logo, não há necessidade de leitura de outras informações para a identificação do formato do arquivo, pois a cada trecho não há dependência com os demais. Pela grande simplicidade da linguagem, optou-se por explicar de forma análoga à dada pelo texto [14], utilizado como referência, para a importação desses arquivos. A seguir encontra-se uma explicação mais detalhada de como o arquivo é importado pelo Orfeu, seguindo-o seqüencialmente, sem nenhum *byte* de informação colocado.

12.2 Início do cabeçalho

No cabeçalho, o valor a versão é ignorada. Esse sistema de formato de arquivo foi abandonado na última versão do Guitar Pro, sendo o formato GP5 incompatível com ele. Como o que muda de versão para versão nos arquivos GTP, GP3 e GP4 é apenas o número de recursos (como efeitos na nota e efeitos no "*beat*"), não há necessidade de verificar esse valor, pois sempre o arquivo será igual. Isso permite que o projeto carregue arquivos antigos do Guitar Pro além do GP4 como conseqüência da importação desses arquivos.

12.3 Propriedades da música

Após a versão, encontram-se informações sobre a peça. Essas informações são todas ignoradas, exceto o nome e o autor da música, por serem as únicas propriedades que existem no Orfeu dentre as apresentadas. Infelizmente todos os valores precisam ser lidos. O nome da música é o primeiro *string* iniciado com inteiro do arquivo (após a versão) e o autor é o quinto *string*. Este bloco de informações sobre a peça possui 8 *strings* iniciadas por inteiro, seguida por um bloco de texto variável contendo a `notice`, texto irrelevante para a importação, cujo formato é

```
<inteiro> <string inic. por inteiro> <string inic. por inteiro>...
```

O número inteiro indica quantas linhas a `notice` possui, podendo até mesmo ser zero, e em seguida têm-se as *strings* de cada linha desse bloco de texto. Concluindo as informações sobre a música há apenas mais um *byte* que indica o `triplet feel`.

que não é relevante para a importação, sendo apenas saltado na leitura.

12.4 Letras

As letras das músicas são armazenadas no arquivo do Guitar Pro e são totalmente ignoradas pelo projeto Orfeu. O primeiro número lido é um inteiro (4 *bytes*) que indica a trilha que contém a letra, e a seguir vem um bloco exatamente 5 vezes contendo:

```
<inteiro> <string iniciada por inteiro>
```

O inteiro representa o compasso inicial da linha de letras e a string é o bloco inteiro de letra. Essa string pode armazenar caracteres de fim de linha, e no caso é adotado o mesmo sistema utilizado em arquivos de texto do DOS (CR+LF).

O inteiro aqui apresentado não foi comentado no texto sobre o formato GP4 utilizado como base e foi encontrado por engenharia reversa. Efetuando alterações no Guitar Pro e verificando a janela do mesmo que mostra as letras, foi possível descobrir qual era a informação que faltava no documento utilizado como base.

12.5 Mais propriedades da música

Aqui encontram-se mais 9 *bytes* de propriedades da música, sendo que os 4 primeiros *bytes* representando a pulsação (*TMusic.Beat*). Os 2 *bytes* seguintes representam a fórmula de compasso e os 3 seguintes não são relevantes para o projeto, pois seu significado não é conhecido.

A fórmula de compasso pode, na verdade, ser apenas o primeiro *byte* desses 2 e em seguida haver um inteiro representando a `octave`, ou "oitava de transposição", como diz o texto usado como base, mas no instante em que os compassos passarem a ter mudanças em sua fórmula será visto que estas ocupam 2 *bytes*, onde pode-se constatar que há algo errado nesta parte. Seja qual for o significado dos 5 *bytes* em questão, os mesmos são ignorados pelo projeto.

É importante salientar que no texto usado como base era dito que esse pequeno bloco teria 6 *bytes* e não 9, sendo esse segundo número obtido por engenharia reversa, ao procurar o início da tabela de canais MIDI no arquivo.

12.6 Tabela de canais MIDI

O Guitar Pro impõe valores a todos os canais MIDI em todos os *ports* aceitos (4 *ports*). A numeração utilizada pelo *software* é sempre o valor real do canal ou do *port* acrescentado de uma unidade, transferindo a faixa de valores de canal de 0..15 para 1..16, por exemplo. Para cada par (*port*, canal) há um bloco. A varredura pelos canais é feita por inteiro até mudar o *port*, como mostra a figura 40.

```
Para MIDIPort variando de 0 a 3
  Para MIDIChannel variando de 0 a 15
    Leia bloco
```

Figura 40: Varredura pelos canais em pseudo-linguagem

Esse bloco contém o instrumento MIDI em um inteiro de 4 *bytes*, seguido do volume, panorama, *chorus*, *reverb*, *phaser*, *tremolo* ocupando cada um apenas 1 *byte*, seguidos de mais 2 *bytes* não utilizados pelo Guitar Pro 4.

Para carregar o arquivo, o Orfeu gera uma tabela de canais MIDI simplificada, com apenas os valores relevantes ao projeto: o instrumento, o volume e o panorama. O instrumento já está no valor correto para o uso direto com as saídas MIDI, mas o volume e o panorama estão na faixa de 0 a 16, ao invés de estarem na faixa de 0 a 127 como o MIDI impõe. Para converter entre essas faixas mantendo sempre números inteiros, o volume e o panorama, multiplica-se por 8 seus valores e subtrai-se 1 depois, exceto no caso de o número ser zero e, portanto, já estar correto.

12.7 Dimensões

Seguem aqui apenas 2 inteiros. O primeiro contém o número de compassos e o segundo, o número de trilhas da música. Isso impõe que todas as trilhas tenham exatamente o mesmo número de compassos. Após lidos esses valores, a música já é inicializada com o número de trilhas adequado e com compassos em branco em todas as trilhas.

12.8 Compassos

O primeiro *byte* do compasso é um *byte* de cabeçalho. Sempre que houver um *byte* de cabeçalho, cada *bit* representa algum tipo de informação. Alguns *bits*, quando possuem valor 1, indicam que algum dado será apresentado posteriormente.

Se o *bit* menos significativo (zero) estiver ativo (valor 1), o próximo *byte* representa uma mudança no numerador da fórmula de compasso, ou seja, no atributo "quantidade". Quando isso não ocorre, o valor desse atributo deve ser igual ao valor usado no compasso anterior. Foi fixado o valor 4 caso o primeiro compasso não tenha esse *bit* ativo.

Se o *bit* seguinte (1) estiver ativo, o próximo *byte* representa uma mudança no denominador da fórmula de compasso, ou seja, no atributo "qualidade". O tratamento desse valor é absolutamente análogo ao tratamento da quantidade.

Se o terceiro *bit* estiver ativo, o próximo *byte* deve ser pulado. Idem para o quarto *bit*. Se o quinto *bit* estiver ativo, tem-se um marcador para saltar. Todo marcador é da forma:

```
<string iniciada por inteiro> <cor>,
```

em que a *string* representa o nome do marcador. Toda cor ocupa 4 *bytes*.

Se o sexto *bit* estiver ativo, tem-se 2 *bytes* para saltar. Esses *bytes* representam uma nova fórmula de compasso. Esse valor estava incorreto no documento utilizado como base, pois a fórmula de compasso ocupa 2 *bytes* enquanto o texto dizia que ocupava apenas um.

O significado dos *bits* ignorados e dos *bits* que apenas impuseram saltos de um *byte* é irrelevante para o projeto na atual especificação. Todas as trilhas possuem a mesma estrutura de compassos, isto é, a qualidade e a quantidade são passadas para todas as trilhas simultaneamente.

12.9 Trilhas

O Guitar Pro também mapeia trilhas e pautas como sendo a mesma coisa, limitação imposta no Orfeu segundo critérios já apresentados.

O primeiro *byte* da trilha é um *byte* de cabeçalho, porém desta vez o cabeçalho

pode ser ignorado. A seguir tem-se uma *string* representando o nome da trilha.

O próximo inteiro representa a quantidade de cordas que o instrumento terá. Esse número está sempre na faixa entre 4 e 7, sendo que os próximos 7 inteiros contém a nota utilizada na afinação de cada corda, a partir da mais aguda. Os últimos valores devem ser apenas pulados se o número de cordas for menor do que 7. O valor da nota armazenado corresponde ao valor que seria usado para enviar mensagens de `NoteOn` e `NoteOff` para a saída MIDI.

Dos 6 inteiros seguintes, apenas os 2 primeiros representam informações relevantes à importação. Esses valores são, na ordem, o *port* MIDI e o canal MIDI. Esses valores estão na faixa de 1 a 4 e de 1 a 16, respectivamente, e devem ser subtraídos de uma unidade para serem válidos. A partir disso é resgatada na tabela de canais MIDI a configuração utilizada pela trilha.

12.10 `Beats` ou `SimNotes`

A figura 41 mostra o pseudo-código utilizado para a leitura das *SimNotes*.

```
Para Compasso variando do inicial ao final
  Para Trilha variando de 0 a 15
    Lê inteiro N (número de SimNotes do compasso)
Lê os N SimNotes
```

Figura 41: Algoritmo utilizado para ler as SimNotes

O primeiro *byte* da trilha é um *byte* de cabeçalho. Após a sua leitura o `SimNotes` é imediatamente inserido no compasso. Se o sexto *bit* estiver ativo, o próximo *byte* deve ser pulado. Esse *bit* é redundante e significa que não há notas neste `SimNotes`.

O *byte* seguinte contém a duração do `SimNotes`, valendo 2 para a semicolcheia, 1 para a colcheia, 0 para a semínima, -1 para a mínima e assim em diante. Calculando 2 elevado a 7 e subtraindo o valor da duração, obtém-se exatamente o sistema de numeração de duração da `TrhyFigType`, o que faz a conversão bastante direta.

Se o *bit* menos significativo estiver ativo, a `SimNotes` tem exatamente um ponto, caso contrário a mesma não tem ponto nenhum.

Vários outros *bits* podem estar ativos e ter conteúdos bastante grandes aqui. O único efeito que é possível resgatar do Guitar Pro é o *staccato*, que é aplicado à nota no Guitar Pro e não à `SimNotes`. A verificação dos demais *bits* foi feita de acordo com o texto de referência do formato do Guitar Pro e todo o seu conteúdo é ignorado. Após o término dessa verificação, encontrar-se-ão as notas pertencentes a este `SimNotes`.

Há um erro grave no documento, que não indica como os `beats` são divididos em `notes`. Foi detectado, através de engenharia reversa, que basta apenas 1 *byte* para essa definição, pois o Guitar Pro trabalha com no máximo 7 cordas. Separando em *bits* esse *byte*, o sexto *bit* representa a corda mais aguda, o quinto *bit* representa a corda seguinte e assim em diante, até chegar no *bit* menos significativo, no caso de o instrumento ter 7 cordas. Quando um desses *bits* tem valor 1, significa que está presente alguma nota nele; quando tem valor 0, significa que não há. Dessa forma é possível que haja um intervalo de tempo totalmente sem notas e outro com notas em todas as cordas. Após esse *bit* encontram-se as notas do dado `SimNotes`, a partir da nota que está na corda mais grave até a que está na corda mais aguda.

12.11 Notas

Diferente dos outros blocos, este bloco está dentro do bloco da `SimNotes`. Essa separação somente foi feita para auxiliar a compreensão do que está acontecendo dentro de um `SimNotes`. A corda em que essa nota aparece já é conhecida através do *byte* final do bloco de `SimNotes`.

O primeiro *byte* da trilha é um *byte* de cabeçalho. Como em nenhum caso usado como teste foi possível fazer o quinto *bit* desse *byte* ser falso, a verificação desse *bit* é feita conforme o pedido no documento usado como base para este texto.

Se o quinto *bit* estiver ativo então há um *byte* indicando o tipo de nota (e não 2 *bytes*, como sugeria a especificação encontrada). O valor 1 representa uma nota normal, o valor 2 representa que a nota é continuação de uma ligadura de valor e o valor 3 representa uma nota morta, mostrado como X no Guitar Pro. Apenas as notas com valor 1 serão inseridas na `SimNotes` no Orfeu. Essas informações foram todas descobertas por engenharia reversa.

A seguir são verificados os *bits* 0 e 4, para saltar o número de *bytes*, conforme especificado no documento já citado diversas vezes aqui.

Conforme pedido pelo documento, agora é testado o *bit* 5 do cabeçalho; se estiver ativo a nota é lida como uma posição no braço do instrumento. Caso seja do tipo 1 (normal) então é inserida ao `SimNotes` aqui.

A única informação que falta coletar é o *staccato*, que corresponde ao *bit* 1 do segundo cabeçalho de efeitos da nota. Não há necessidade de explicitar essa parte aqui no texto por ser exatamente o que consta no documento usado como base. Todos os valores precisam ser lidos para serem saltados e poder chegar na nota seguinte.

12.12 Tabela de acordes

Como não há necessidade de carregar os acordes da música armazenados nessa tabela, o conteúdo que existe após o último `SimNotes` ter suas notas lidas é totalmente ignorado, e portanto essa tabela nunca é lida.

13 Exportar para MIDI

13.1 O arquivo MIDI

Um arquivo MIDI é um arquivo que contém mensagens MIDI guardadas para serem enviadas a uma saída MIDI. Antes de qualquer mensagem há um valor chamado *delta-time* qem que é guardada a informação de quanto tempo precisa passar até que a mensagem seja enviada. Essas mensagens são guardadas em trilhas dentro do arquivo MIDI. O método chamado pela janela principal ao solicitar pelo *menu* a opção de exportar arquivo MIDI é o responsável por tudo o que está aqui.

Os arquivos MIDI podem ser do tipo 0, 1 ou 2. O tipo 0 é um tipo que deve ter apenas uma trilha, mesmo que tenha conteúdo de todos os canais. O tipo 1 significa que há N trilhas que começam simultaneamente no instante inicial (modo síncrono), não necessariamente um canal por trilha. O tipo 2 significa que cada trilha será tocada de forma independente (modo assíncrono).

Cada arquivo MIDI é dividido em partes chamadas *chunks*. Existem 2 tipos de *chunks*, o *header chunk* (cabeçalho) e o *track chunk* (trilha). Um arquivo MIDI sempre é formado de um *header chunk* seguido de um ou mais *track chunks*. Os *chunks* iniciam com 4 caracteres de identificação (`MThd` e `MTrk`, para o *header chunk* e o *track chunk*, respectivamente), depois um inteiro de 4 *bytes* (no formato *big endian*, ao contrário do *little endian* usado no arquivo GP4) contendo o tamanho dos dados do *chunk*. Todo dado depois disso será dado do *chunk*.

Em um *header chunk* há sempre 6 *bytes* de dados (portanto aqui o campo de tamanho do *chunk* é sempre constante), divididos em 3 números de 2 *bytes*. O primeiro número representa o formato ou tipo do arquivo MIDI como 0, 1 ou 2, como visto anteriormente. O segundo apresenta o número de trilhas presentes no arquivo. O terceiro informa como será feita a divisão rítmica para contagem dos *delta-times*.

Em um *track chunk* os dados são as mensagens ditas anteriormente, sempre precedidas de um *delta-time*, podendo haver quantas mensagens quantas forem necessárias.

13.2 Formato inteiro de comprimento variável MIDI

O *delta-time* é um número no formato de tamanho variável do MIDI. Esse formato utiliza o *bit* mais significativo de cada *byte* para indicar se haverá outro *byte* ou não para descrever a quantidade. Quando ativo, esse *bit* indica que há mais um *byte*. Por exemplo, no *byte* $81 ($ representa que o número está em hexadecimal), o *bit* mais significativo está ativo, indicando que há mais um *byte* do valor e que os 7 bits mais significativos já foram dados. Se o próximo *byte* for $80 tem-se então mais 7 *bits* do número e sabe-se que há mais um *byte*. Se o próximo *byte* for $45, finalizamos o valor com 3 *bytes*, fornecendo um número de 21 *bits*. A tabela 8 mostra um exemplo para esse caso.

Tabela 8: Exemplo de seqüência MIDI

Hexadecimal	8	1	8	0	4	5
Binário	1000	0001	1000	0000	0100	0101
Sem os *bits* de tamanho	000	0001	000	0000	100	0101
Reorganizando	0000	0001	0000	0000	0100	0101
Hexadecimal	0	1	0	0	4	5

Logo, o número armazenado foi $4045. A conversão inversa (de $4045 para $818045) é feita pela subfunção `writeVarLen`, que recebe o inteiro como parâmetro.

13.3 *Header chunk*

Para exportar o MIDI, a seqüência gerada será apenas a da `Staff` apresentada na tela. Isso sugere que o arquivo seja do tipo 0, de trilha única. Logo há apenas 2 *chunks* na estrutura, sendo que o cabeçalho (em hexadecimal) fica como mostrado na tabela 9:

A vantagem de ter o *delta-time* em milisegundos é que agora pode-se mandar apenas os valores de saída do `tickDuration` dos `SimNotes`, existente para propiciar a saída MIDI.

13.4 *Track chunk*

De início, é colocado o identificador do *chunk*:

Tabela 9: Cabeçalho para a seqüência gerada

4D 54 68 64	MThd, identificando o início de um *header chunk*
00 00 00 06	Tamanho do *chunk* a partir do *byte* seguinte, no caso de *header chunk* é sempre 6
00 00	Arquivo do tipo zero (trilha única)
00 01	Número de trilhas
E7 28	Divisão temporal; esse valor torna a unidade do *delta-time* igual a milisegundos

4D 54 72 6b, que corresponde ao MTrk, identificando o início de um *track chunk*. Aqui é guardada a posição do cursor (apesar de ser constante), e após isso salta-se 4 *bytes* (tamanho do *chunk*). Um iterador é criado para passar por todas as notas. A lógica é mostrada através da figura 42.

```
// Este bloco faz a primeira nota soar
writeVarLen(0)
Para cada nota no SimNotes apontado
  Escreve uma mensagem de NoteOn para a nota

Enquanto não estiver na última nota
  // Este bloco faz a nota anterior parar
  writeVarLen(Duração da SimNotes) // SimNotes.tickDuration
  Para cada nota no SimNotes apontado
    Escreve uma mensagem de NoteOff para a nota
  // Este bloco faz a nova nota soar
  writeVarLen(0)
  Move o iterador para a próxima nota
  Para cada nota no SimNotes apontado
    Escreve uma mensagem de NoteOn para a nota

// Este bloco faz a última nota parar
writeVarLen(Duração da SimNotes)
Para cada nota no SimNotes apontado
  Escreve uma mensagem de NoteOff para a nota
```

Figura 42: Lógica do iterador em pseudo-linguagem

A mensagem é exatamente a mensagem descrita anteriormente, contendo sempre 3 *bytes* (Status, Param1 e Param2). O parâmetro de dinâmica do NoteOn e NoteOff é sempre fixo aqui. Observe que há uma grande analogia com o que é feito na execução sonora.

A partir do instante em que a música terminou de ser passada, a posição do arquivo armazenada é apontada para colocar o tamanho do mesmo, agora conhecido.

14 Resultados Obtidos

14.1 Características e limitações da exibição em partituras

A união das bandeirolas das figuras rítmicas não foi realizada, por ser dispensável para a interpretação do que está escrito, servindo apenas para facilitar a leitura musical. Foi verificado que a união das bandeirolas é um atributo a ser colocado no compasso (`TMeasure`) em uma seqüência de números ordenados (uma lista ligada), embora não tenha sido possível implementar essa característica.

A multiplicidade de vozes apenas é possível utilizando-se trilhas separadas, devido ao fato de a classe `TMelRhy` não ter sido implementada, pois sua conexão com outras classes e, principalmente, sua exibição visual iriam levar mais tempo do que o disponível. Essa classe também interferiria no funcionamento da execução sonora, na importação, na exportação, no formato de arquivo baseado em XML do Orfeu, na transposição, no sistema de iteração dentro da música (classe `TCursor`).

A representação de ritmos a partir de divisões complexas da pulsação a partir do uso de quiálteras foi projetada, porém não foi implementada, pelo fato de adicionar complexidade em todos os aspectos do projeto, de forma similar à classe `TMelRhy`.

Existe uma regra para a colocação de acidentes ocorrentes que impõe que eles continuam "valendo" até o final do compasso, mas apenas para a oitava em que eles apareceram. Embora seja simples resumir a regra em uma frase, essa regra mostrou-se um problema grande a ser resolvido, mas o projeto contempla tal detalhe de notação gráfica, devido à sua importância para os músicos e por não haver alternativa quanto à sua exibição. Vale citar que um programa, o NoteWorthy Composer, *software* de edição de partituras voltado aos compositores, não obedece a essa regra de grafia musical.

Não houve necessidade de inserir mudanças de clave, sendo a clave inicial fixada para toda a música. O mesmo ocorreu com a tonalidade da música, sempre fixada em dó maior (ou lá menor), permitindo a mudança de tonalidade apenas com base em acidentes ocorrentes.

A ligadura de valor não foi inserida, pois apesar de ser uma ferramenta bastante utilizada no controle das durações pelo usuário, não era requisito, por este adicionar apenas uma complicação gráfica ao projeto, já que a mudança na parte sonora é bastante simples, bastando garantir que a nota continua soando até o final da última nota ligada, sem nova emissão sonora.

A exibição de múltiplas trilhas em uma única tela não foi realizada pelo fato de mostrar-se desnecessária em uma primeira versão do projeto. Para um compositor isso é normalmente interessante, fornecendo uma visão ampla de seu projeto musical, porém, até mesmo o Guitar Pro 3 não tinha esse recurso, mostrando que isso não é fundamental para um músico. A complexidade gráfica envolvida nessa exibição é grande, embora não haja grandes interferências com o restante do projeto.

O *software* Orfeu suporta indefinidos pontos de aumento. No Guitar Pro, só é possível colocar um ponto e o Sibelius está limitado em três. O Orfeu admite quantos pontos o usuário necessitar sem prejuízo da visualização.

Com relação às figuras rítmicas, o Orfeu possui faixa mais ampla que o Guitar Pro e até que o Sibelius, pois este último não possui a "Máxima", utilizada em músicas renascentistas.

A quebra de compassos é sempre feita após o mesmo ultrapassar um determinado tamanho constante. Essa quebra reflete o que naturalmente ocorre na impressão de partituras, em que a partitura continua no sistema (ou linha) seguinte.

Como uma facilidade a mais para o usuário, foi criada uma barra de status que mostra o percentual completo do compasso e a nota apontada pelo cursor. Essas informações poderiam ser reproduzidas em formato de áudio e seriam o início para permitir que deficientes visuais pudessem utilizar o *software*.

Há um cursor acompanhando o que está sendo tocado. Redimensionar essa janela durante a execução não altera em nada o programa.

14.2 Características e limitações da exibição em tablaturas

A representação visual do ritmo não foi colocada nas tablaturas. Como não há requisito de detalhamentos na visualização da tablatura, foi considerado o suficiente inserir informações que constam em arquivos de texto utilizados como tablatura que são distribuídos pela *internet*. Esses arquivos de texto contém apenas as linhas com suas respectivas afinações e números indicando onde a nota deve ser tocada, além de barras que delimitam blocos rítmicos que normalmente são os próprios compassos, informações essas exibidas na tablatura.

Esse editor é basicamente o mesmo editor de partituras, apenas com a parte gráfica adaptada ao novo sistema de edição. Em particular, a barra de status passa a

exibir a afinação da corda apontada pelo cursor, já que não há mais uma nota apontada e, portanto, não é necessário exibir a informação da situação do cursor do modo de editor de partitura.

O som, assim como na edição de partituras, é apresentado exibindo-se um cursor na trilha que está sendo visualizada. Todas as trilhas são executadas simultaneamente no ritmo das mesmas, conforme feito com a partitura, pois o objeto que realiza tal processo é o mesmo.

Diferente do esquema de visualização de partituras, as tablaturas são redimensionáveis, ou seja, a largura não foi fixada e é modificada junto ao tamanho da janela, podendo ser redimensionada e atualizada até mesmo quando a execução sonora foi iniciada. Porém, por razões de compartilhamento de processamento, essa alteração da tela pode fazer com que o som ritmicamente, por efetuar uma operação relativamente lenta em um instante inapropriado.

15 Aceitação do projeto - requisitos funcionais

Listando primeiramente os requisitos funcionais do projeto, na ordem em que foram apresentados, tem-se:

15.1 Editor de partituras

(I). Manipulação das partituras

(II). Abrir/Salvar partituras

O editor de partituras foi realizado, incorporado ao (`FormMain`), janela principal, permitindo alterações na música através do teclado de computador, do *mouse* (apenas com *widgets* em janelas, como a janela de edição de uma trilha) e do teclado MIDI (apenas a modificação de um acorde na música; o acorde modificado é o apontado pelo cursor no instante em que o teclado é utilizado).

Utilizando o padrão XML, foi definida uma linguagem que define uma música no formato do Orfeu. Dessa forma o Orfeu é capaz de armazenar e carregar músicas representadas por partituras em seu formato próprio.

15.2 A representação visual da partitura deve ser inteligível

(I). Altura jamais visualmente ambígua

(II). Ritmo inteligível

A altura de uma nota em uma partitura é representada pela posição da mesma no pentagrama. No caso particular em que apenas uma melodia é representada não há dúvidas quanto à identificação da nota, porém no caso de intervalos de uníssono aumentado ou uníssono diminuto, não há uma notação adequada em partitura para representá-los, únicos casos em que não seria possível evitar a ambigüidade. Nesses casos nem todas as notas são representadas, forçando para que não haja ambigüidade por evitar que alguma nota seja exibida. Felizmente este é o único caso em que a representação visual não corresponde ao que está armazenado (e que pode ser reproduzido sonoramente), conforme previsto pela teoria musical.

O ritmo, na representação em partitura, é totalmente inteligível para qualquer figura entre a quartifusa (figura utilizada na Sonata Patética de Beethoven, logo em sua primeira página) até a máxima (figura utilizada principalmente em obras do período renascentista, utilizada por exemplo na primeira nota da Missa de Beata Virgine de Josquin des Prês). Mesmo havendo simultaneidade sonora representada graficamente, a duração é visualmente representada com perfeição. As fórmulas de compasso são devidamente exibidas e a pulsação (*beat*) é mostrada na tela de edição das propriedades de uma música.

15.3 Importação de GP4

Conforme dito no tópico sobre o formato dos arquivos de tablatura do Guitar Pro até a sua versão 4.06, foi inserido no projeto a possibilidade de importar arquivos nos formatos GTP (Guitar Pro até 2), GP3 (Guitar Pro 3) e GP4 (Guitar Pro 4), salvos pelo Guitar Pro até a versão 4.06.

Diversas características são perdidas durante a importação, por não haver equivalentes no projeto, já que o objetivo daquele *software* é bastante diferente do Orfeu. Há uma diferença muito grande entre o foco dado à polirritmia e à representação de partituras pelos *software*, características deixadas em segundo plano pelo Guitar Pro. No entanto o Guitar Pro demonstrou dar bastante importância à representação de acordes cifrados, como uma notação alternativa para a música popular específica do instrumento. Provavelmente esta característica ajuda o Guitar Pro a limitar o número de cordas do modo de tablatura na faixa de 4 a 7 cordas.

15.4 Editor de tablaturas

(I). Manipulação das tablaturas

(II). Abrir/Salvar tablaturas

O editor de tablaturas foi feito, com base no sistema de edição utilizado pelo Guitar Pro, utilizando características de editores de partituras. Esse sistema de edição utilizando-se o teclado ficou, no mínimo, tão simples quanto o do Guitar Pro 4, por incluir todos os recursos factíveis por teclado pelo mesmo além de outros.

O formato de arquivo do Orfeu, baseado em XML, contém anexos que fazem

a mesma música sempre poder ser representada como partitura e como tablatura. Diferente do Guitar Pro, a representação nativa do projeto é a partitura, e a tablatura é colocada como um anexo à notação.

15.5 Equivalência entre partituras e tablaturas

(I). Formato de armazenamento de músicas com ambos

(II). Conversão de tablatura para partitura

(III). Conversão de partitura para tablatura

Como já foi dito, as partituras e as tablaturas compartilham o mesmo formato de representação interna e de armazenamento baseado em XML. Basicamente, as partituras são armazenadas no formato XML e as tablaturas são anexos às partituras, incluindo informações extras de exibição. Um paralelo a isso é o fato de a tablatura incluir a corda em que a nota se encontra e a partitura incluir o lado para o qual a haste deve ser desenhada. Neste exemplo, a informação de um modo não tem valor nenhum para o outro. A equivalência existe, mas para isso cada modo de exibição passa a ter dados redundantes não utilizados na impressão. Todas as informações são armazenadas, mesmo que sejam redundantes em algum dos modos de edição.

Todo dado inserido em algum dos editores é imediatamente jogado ao outro editor, ou seja, há uma conversão de partitura para tablatura e de tablatura para partitura feita de forma direta, com um mapeamento fixado. Essa conversão é normalmente inadequada e deve ser manipulada pelo usuário caso seja conveniente, ou utilizar o algoritmo baseado em autômato adaptativo para encontrar uma digitação adequada para o modo tablatura.

15.6 Conversão adaptativa em um dos casos acima

Utilizando-se um algoritmo baseado em um autômato adaptativo, é possível a partir da música já armazenada encontrar uma digitação no modo tablatura que seja conveniente ao usuário. Para fazer isso, os dados do modo tablatura inclusos na música são basicamente descartados e substituídos pelos obtidos em uma nova conversão de partitura para tablatura, analisando o contexto em que o trecho se encontra e obedecendo a critérios previamente estabelecidos.

15.7 Personalização de algum recurso sonoro que não é totalmente explícito em partitura

O recurso sonoro utilizado foi o *staccato*. Tal recurso consiste na reprodução do som por uma duração de metade de seu tempo, ou próximo disso. A personalização consiste em poder adequar o *staccato* para ser executado conforme o usuário desejar.

15.8 Criação de uma seqüência

A criação da seqüência no formato MIDI é essencial para que outros requisitos abaixo sejam aceitos. O simples fato de a música ser reproduzida já é o suficiente para dizer que uma seqüência foi criada.

15.9 Reprodução de uma seqüência

O som é reproduzido na saída MIDI, previamente escolhida por uma tela de configurações. Essa saída MIDI (sintetizador) deve obedecer ao padrão General MIDI para que o mapeamento dos timbres seja igual ao exibido na tela, embora não importe qual é o padrão para que a seqüência seja executada, importando apenas para a correta relação entre os instrumentos selecionados e os instrumentos que estão executando a música.

15.10 Detalhes da reprodução

(I). Polifonia

(II). Poliritmia

A polifonia já é uma característica do formato MIDI, por permitir 16 canais simultâneos por *port*. Essa característica foi herdada do MIDI pelo *software* a partir de seu uso, bastando enviar os sinais MIDI corretamente. Essa polifonia também não se limita a 16 notas, visto que um mesmo timbre pode estar fazendo mais de uma nota.

A poliritmia foi levada a alguns extremos no projeto. A mudança de fórmula de compasso e sua total personalização foram feitas por completo, incluindo possibilidades de modificação não presentes na maioria dos softwares citados por este texto (o

único que apresenta essa personalização é o *NoteWorthy Composer*. Essa personalização consiste em permitir que cada trilha tenha compassos diferentes entre si, mesmo que não haja encaixe entre eles, permitindo, até certo ponto, uma complexidade rítmica similar ao composto por compositores modernos como Igor F. Stravinski. Essa complexidade é apenas barrada por uma pulsação fixada para a música toda, que define em bpm (*beats per minute*) a duração de notas cuja figura rítmica é a semínima. A partir da proporcionalidade entre as figuras é possível obter qualquer combinação rítmica na estrutura de dados, permitida pelo editor e exibida corretamente, embora apenas uma trilha por vez.

15.11 Interface com o teclado

Conforme já foi dito anteriormente, o teclado tem atalhos que fornecem total liberdade ao cursor e sua edição. Com o auxílio de atalhos do sistema operacional todo o sistema de *menus* da janela principal e as demais janelas são acessíveis mesmo na ausência de *mouse*, apesar de a edição das afinações da tablatura ter dificuldade considerável sem o *mouse*, é possível. Essa edição de afinações se torna extremamente simples quando comparada com a edição apenas com o teclado. Vale salientar que o Guitar Pro, além de não permitir uma quantidade qualquer de cordas na tablatura, não permite a realização de todas as ações unicamente através de teclado, exigindo o *mouse* para diversas ações, em particular, para a edição da afinação de cada trilha.

15.12 Impressão

Um componente básico de impressão apenas remete o que está impresso na tela para a saída de impressora, através do sistema operacional. Apesar da falta de configuração do recurso, isso basta para garantir o que foi pedido pelo requisito.

15.13 Exportação MIDI

A exportação MIDI, conforme dito anteriormente, foi realizada de uma forma bastante simplificada. Apenas as informações mínimas relevantes foram colocadas: altura e duração. Apenas a trilha em edição é convertida, logo não há polifonia na seqüência exportada, a fim de simplificar o arquivo final.

16 Aceitação do projeto - requisitos não-funcionais

Tendo todos os requisitos funcionais aceitos com êxito. Os mesmos estão listados s seguir.

16.1 Compatibilidade

- *Windows XP*
- *Pentium IV*
- *AMD Sempron*
- *AMD Athlon*

Os computadores utilizados no desenvolvimento foram AMD Sempron com Windows XP, nos quais também eram realizados testes. O projeto foi executado em AMD Athlon e em Pentium IV a fim de confirmar que o funcionamento do mesmo não depende do processador. Devido às características do Delphi e da API do Windows isso era previsto.

16.2 Recursos

- Seqüenciação da música através do Orfeu
- Síntese sonora externa ao projeto

As seqüências foram criadas pelo Orfeu em conjunto com a síntese por *software*, restando recursos o suficiente até mesmo para redigir um texto em outro aplicativo ao mesmo tempo em um AMD Sempron de 1.8GHz e 512MB de RAM.

16.3 Mínimo de duas vozes na polifonia

Foi realizado um teste com seis vozes de polifonia sem nenhuma falha rítmica perceptível. Para o teste bastou colocar acordes completos de violão usando as 6 cordas.

16.4 Latência entre o pedido de execução e seu início deve ser de no mínimo 1 segundo

O tempo entre o pedido de execução e o início de execução na maior parte das vezes não foi humanamente perceptível, dificilmente ultrapassando o valor de 0,1s. Essa latência depende mais do sistema operacional e do dispositivo de saída do que do projeto em questão.

16.5 Deve funcionar mesmo que não haja como sintetizar o som

Na ausência de sintetizador de som uma mensagem de erro é exibida apenas na tentativa de executar a música. Todos os recursos do programa continuam acessíveis e essa tentativa de executar a música não o trava.

17 Itens adicionais implementados

17.1 Transpor partituras

O recurso de transposição de partituras foi implementado, permitindo que o usuário transponha todas as suas notas sem a perda dos intervalos reais para até quatro oitavas, ascendente ou descendente. Esse recurso pode ser aplicado a todas as trilhas de uma só vez para facilitar para o usuário. Comparando com o Guitar Pro, o recurso de transposição deste apenas realiza a transposição em semitons, perdendo a informação original dos intervalos segundo a teoria musical. Além disso a transposição do Guitar Pro se limita a uma oitava. Esse recurso de transposição se compara ao do Sibelius, por possuir, também, uma transposição dada por um intervalo como o representado em `TInterval`.

17.2 Entrada MIDI

O uso de um teclado MIDI ou outro instrumento com conexão MIDI ao computador mostrou-se extremamente adequado ao usuário, facilitando a edição da música e permitindo que o mesmo não tenha grande domínio sobre teoria musical para realizar a edição. O teclado insere apenas o acorde sendo tocado, ignorando a duração do mesmo.

17.3 Poliritmia total sobre uma pulsação

A execução do ritmo, como foi dito anteriormente, está mais completo do que o pedido pelos requisitos. A polirritmia inserida é capaz de executar qualquer música desde que haja proporcionalidade entre as figuras rítmicas. A divisão em compassos não é importante. Essa proporcionalidade surge do fato de se manter uma pulsação, mesmo que esteja a um mínimo múltiplo comum entre figuras de unidades de compasso ou mesmo que nunca haja uma simples sincronia entre eles, bastando obedecer à proporcionalidade entre figuras.

17.4 XML

O formato de arquivo da saída poderia ter sido implementado em binário, utilizando o recurso de armazenamento de variáveis do tipo `record` do Delphi, porém esse formato seria de difícil manipulação e poderia ficar travado nos recursos previsos para ele na primeira versão do *software*. Para evitar esse problema um formato de arquivo baseado no XML foi elaborado para constituir o formato interno do Orfeu.

17.5 N cordas

Uma das maiores inovações do *software*, um *software* editor de tablaturas que provê total liberdade para a escolha do número de cordas que o instrumento tem. Dessa forma é possível separar o traste de cada uma das 12 cordas de um violão de 12 cordas e também separar as cordas para um baixo de 9 cordas, ambos instrumentos em que isso é impossível nos *softwares* convencionais como o Guitar Pro (impossível até mesmo na versão 5).

17.6 Afinações e configurações em "INI"

O músico pode, se desejar, manipular um acervo de afinações, armazenando-as para uso posterior, ou removendo-as. Todos os recursos de manipulação das afinações para N cordas foram colocados em uma única tela, a tela de edição das propriedades de uma trilha. Os dados dessas afinações são armazenados em um arquivo "INI", formato padrão de arquivos de configuração do Windows, sendo que caso esse arquivo não exista a própria aplicação se encarrega de criá-lo. Apenas uma afinação foi colocada no código para evitar a ausência de afinações no acervo.

18 Conclusão

Através dos itens 15 e 16 pode-se concluir que o projeto foi aceito, pois todos os requisitos funcionais e não-funcionais foram atendidos de forma, no mínimo, satisfatória. Em alguns casos o projeto e a implementação superaram as espectativas, como nas capacidades de polifonia, na tela de edição das propriedades de uma trilha, na tela de troca de fórmula de compasso e nos atalhos da interface com o teclado de computador.

Para que o *software* Orfeu torne-se um produto competitivo com os outros existentes no mercado há alguns itens que devem ser desenvolvidos, listados a seguir:

- União das bandeirolas das figuras rítmicas
- Multiplicidade de vozes na mesma trilha
- Visualização e representação sonora da ligadura de valor
- Ligaduras de fraseado
- Escolha da clave
- Escolha da escala musical
- Quiálteras
- Trilhas com pelo menos duas pautas
- Permitir visualizar a música toda de uma vez
- Mais ornamentações, ou pelo menos a representação gráfica das mesmas
- Permitir deixar o compasso todo na mesma linha ou sistema (não quebrá-lo)
- Inclusão dó ritmo na notação de tablatura

Apesar de haver ainda vários itens necessários para que o Orfeu torne-se um produto aceito pelo mercado, vale ressaltar que o projeto foi aceito, pois todos os requisitos foram cumpridos satisfatoriamente e, tendo em vista a quantidade de elementos que foram desenvolvidos e o tempo disponível para seu desenvolvimento, o desenvolvimento do mesmo saiu como o esperado, senão melhor. Para importar arquivos do Guitar Pro 4, por exemplo, pois o texto encontrado contendo sua descrição possuía

erros que não eram esperados a partir do instante em que o mesmo foi descoberto. Apesar do imprevisto, foi possível concluir a importação de arquivos do Guitar Pro.

É importante ressaltar também que todos os músicos têm queixas com relação aos *softwares* presentes no mercado e, por isso, tentamos incluir no Orfeu o que foi considerado bom, melhorar o que foi considerado ruim e até projetar o que os outros *softwares* não eram capazes de fazer. Outro ponto importante é que o Orfeu tem um diferencial: a conversão adaptativa de partitura para tablatura. Essa conversão é algo freqüentemente buscado por guitarristas, baixistas e violonistas, e nenhum *software* atual faz uma conversão aceitável. Seria interessante também, ao torná-lo um produto, aprimorar o algoritmo para promover ainda mais o Orfeu.

Referências

[1] Object-oriented efficiency comparison: Java, c++ and c#. Acesso em 15 de abril de 2007. Disponível em: `http://www.cs.hut.fi/Opinnot/T-106.290/K2004/Final/mbjorkqv.ps.gz`.

[2] F.A. F. Anselmo. Desvendando o caminho das pedras. Technical report, Borland, 1997.

[3] A. C. Costa, A. F. Borgatto, and Demétrio C. G. B. Curso de introdução ao latex. Technical report, ESALQ, 2002.

[4] I. Darwin. Gui development with java. *Linux Journal*, 61(4), 1999.

[5] S. Garfinkel. Java: slow, ugly and irrelevant. Acesso em 18 de abril de 2007. Disponível em: `http://archive.salon.com/tech/col/garf/2001/01/08/bad_java/print.html`.

[6] E. D. Neto. Redimensionando imagens proporcionalmente. Acesso em 16 agosto de 2007. Disponível em: `http://www.activedelphi.com.br/modules.php?op=modload&name=News&file=article&sid=568`.

[7] J. J. Neto. Adaptive automata for context-sensitive languages. *Sigplan Notices*, 29(9):115–124, September 1994.

[8] T. Pedrazzi, A. H. Tchemra, and R. L. A. Rocha. Adaptive decision tables - a case study of their application to decision-taking problems. In *Proceedings of International Conference on Adaptive and Natural Computing Algorithms - ICANNGA*, pages 341–344, March 21-23 2005.

[9] J. Spolski. Elegance. Acesso em 1 de abril de 2007. Disponível em: `http://www.joelonsoftware.com/items/2006/12/15.html`.

[10] J. Spolski. Exceptions. Acesso em 3 de abril de 2007. Disponível em: `http://www.joelonsoftware.com/items/2003/10/13.html`.

[11] B. Stroustrup. C# language. Acesso em 15 de abril de 2007. Disponível em: `http://students.cs.tamu.edu/jchen/cpsc689-606/language.pdf`.

[12] P. Tyma. Why are we using java again? *Communications of the ACM*, 41(6):38–42, June 1998.

[13] V. Vranic. Multiple software development paradigms and multi-paradigm software development. Acesso em 13 de abril de 2007. Disponível em: http://www.fiit.stuba.sk/0ranic/pub/MultiParadigmSwDev.pdf.

[14] L. Vromman. Guitar pro 4.06 file format description. Acesso em 15 novembro de 2007. Disponível em: http://dguitar.sourceforge.net/GP4format.html.

[15] Wikipedia. Xml. Acesso em 3 dezembro de 2007. Disponível em: http://en.wikipedia.org/wiki/XML.

[16] A. Wilson. Gerando imagem para validação de números. Acesso em 19 agosto de 2007. Disponível em: http://www.activedelphi.com.br/modules.php?op=modload&name=News&file=article&sid=290&mode=thread&order=0&thold=0.

[17] F. Zuffo and H. Pistori. Tecnologia adaptativa e síntese de voz: Primeiros experimentos. In *Anais do V Workshop de Software Livre - WSL*, Junho, 2-5 2004.

APÊNDICE A - GLOSSÁRIO

Acidente: sinal que indica o intervalo em semitons de uma nota com relação à mesma sem o acidente.

Acorde: conjunto de três ou mais notas simultâneas.

Altura: qualidade do som que identifica se ele é mais grave ou mais agudo que outro, isto é, nome dado à freqüência fundamental da forma de onda sonora.

Andamento: valor numérico em batidas por minuto indicador da pulsação a ser usada. Tradicionalmente utiliza-se palavras indicadoras do andamento (*adagio, allegro*, etc.), pois o conceito de andamento surgiu muito antes da invenção do metrônomo.

Arquivo GP4: formato de arquivo utilizado no *software* Guitar Pro 4, armazenando tablaturas de guitarra, sejam quais forem os instrumentos.

Arquivo MIDI: formato de arquivo baseado no protocolo MIDI para armazenar seqüências, incluindo informações sobre o ritmo da música que não são necessárias no protocolo.

Bend ou pitch bend: glissando cujos sons intermediários são necessariamente contínuos.

Bequadro: anulador de acidentes.

Bicorde: conjunto de duas notas simultâneas.

Breve: figura rítmica que representa a duração de duas semibreves.

Clave: fixador de uma altura base no pentagrama.

Colcheia: figura rítmica que representa a duração de duas semicolcheias.

Compasso: é uma forma de dividir os sons de uma composição musical em grupos, de forma quantizada.

Dinâmica: qualidade do som que identifica a força com que a nota soa. Difere do volume sonoro pelo fato de que a força com que uma nota é tocada influencia no timbre da mesma.

Figura rítmica: imagem que representa a proporção entre a duração das notas e pausas de uma música.

Fusa: figura rítmica que representa a duração de duas semifusas.

General MIDI: conjunto de mensagens, numeração padrão dos instrumentos musicais e padrão de canais utilizados no MIDI (camada de nível mais alto anexada ao protocolo MIDI).

Glissando: ornamento que indica que a passagem entre duas notas deve ser feita passando por todas as notas entre elas, segundo algum critério que depende da representação do glissando e do instrumento em que ele será tocado.

Ligadura de fraseado: representa a união entre as notas do trecho ligado à melodia, não permitindo nenhuma pausa mínima entre a finalização de uma nota e o início da seguinte.

Longa: figura rítmica que representa a duração de duas breves.

Máxima: figura rítmica que representa a duração de duas longas.

Melodia: conjunto ordenado e com ritmo da execução de pausas e notas sem sobreposição no tempo.

Microdinâmica: é uma pequena variação de dinâmica não representada em partitura que não sai da faixa de dinâmica estabelecida na partitura.

MIDI (*Musical Instrument Digital Interface*): originalmente um protocolo de comunicação entre instrumentos musicais e sintetizadores.

Mínima: figura rítmica que representa a duração de duas semínimas.

Nota: representação gráfica de um som básico emitido por um instrumento, caracterizado por possuir altura fixa, ou por representar um som percussivo.

Nota pontuada: nota seguida de um ponto, o que indica que o som emitido deverá ter duração de da nota que o precede somado com metade da sua duração.

Ornamento: detalhe musical artístico para enfeitar uma música, normalmente sem conter precisão de execução na notação padrão.

Partitura: também conhecida como notação musical padrão, é o sistema de representação visual de músicas a partir de notas e pausas com figuras rítmicas, além de representações de andamento, dinâmica, ornamentação e diversas outras possibilidades artísticas.

Pauta: pentagrama com seu conteúdo.

Pentagrama: conjunto de cinco linhas que representam alturas para um dado instrumento, quando é dada uma clave.

Percussão: sonoridade com ataque (início) mais importante do que sua sustentação de vibração periódica que define a altura do som.

Polifonia: simultaneidade de sons.

Polirritmia: simultaneidade de ritmos.

Pulsação: sensação de periodicidade rítmica de uma música.

Quartifusa: figura rítmica de duração extremamente curta.

Reverberação: é uma característica da interferência do ambiente no som.

Ritmo: qualidade do som que identifica sua duração.

Semibreve: figura rítmica que representa a duração de duas mínimas.

Semicolcheia: figura rítmica que representa a duração de duas fusas.

Semifusa: figura rítmica que representa a duração de duas quartifusas.

Semínima: figura rítmica que representa a duração de duas colcheias.

Seqüência: representação digital de uma melodia ou de uma música através de suas notas, não de seus sons.

Seqüenciador: reprodutor de seqüências em um sintetizador.

Sintetizador: reprodutor de sons a partir de seqüências.

Sistema: é um conjunto de pautas unidas até ocorrer uma quebra na vertical por fim do espaço.

***Slide*:** é o transitar entre notas passando por todas as intermediárias no braço do instrumento.

***Staccato*:** articulação que indica que a nota deve ser tocada rapidamente ou, tipicamente, pausada a partir da metade de sua duração

Tablatura: notação alternativa utilizada por guitarristas e violonistas, composta de

linhas com números indicando a posição em que as notas devem ser tocadas.

Timbre: qualidade do som que identifica seu instrumento de origem, isto é, proporção entre as freqüências dos harmônicos (forma de onda).

Transposição: processo de se modificar a altura de uma nota ou coleção de notas por um intervalo constante. Quando se transpõe uma musica, automaticamente modifica-se o tom em que se encontra.

Trilha: conjunto de pautas.

Trinado: Uma alternância rápida, durante toda a duração da nota, entre o som da mesma e outro som.

Volume: intensidade sonora em escala logarítmica.