Resumo

A difícil tarefa de solfejar partituras, bem como a necessidade por ferramentas que ajudassem na composição de melodias, impulsionaram o desenvolvimento das *Score Input Languages*: área da Computação Musical que visa a facilitação do processo de composição.

Este trabalho apresenta o compilador de partituras musicais *Solfeggiare*, capaz de transformar documentos MusicXML de descrição de partituras em uma melodia no formato de arquivo MIDI. O compilador, além de gerar a melodia em forma de arquivo, também a executa e possui um suporte à geração de documentos MusicXML a partir de partituras.

Para a sua elaboração, se fez necessário um estudo de teoria musical para ajudar na compreensão da notação gráfica de uma partitura. Também foram estudadas as *Score Input Languages*. Essas linguagens de composição têm por principal característica a análise de partituras tanto de forma gráfica como de forma musical.

A estrutura de um compilador geral foi brevemente explanada bem como as ferramentas e APIs que foram utilizadas no desenvolvimento do *Solfeggiare.* A API JDOM e a JFugue merecem destaque. A primeira é responsável pelo *front-end* do compilador e a segunda pelo *back-end*.

Um estudo aprofundado sobre o protocolo MIDI também está incluído neste trabalho. Como funciona o *hardware* MIDI, qual a sua especificação e o padrão de formatação de seus arquivos foram essenciais para o desenvolvimento das Linguages de Composição.

Ao fim, um estudo de caso é apresentado com um trecho da música “Este Seu Olhar” de Tom Jobim para validar a implementação realizada.

Abstract

The development of Score Input Languages was triggered by the need of a tool that made the composition process easier for composers and also by the difficulty that most of the music students have on the sight reading of musical scores. Score Input Languages are a part of the Composition Languages whose main target is to facilitate the composition process.

This work presents the musical score compiler *Solfeggiare* which is capable of transforming score-description MusicXML documents in a melody in the MIDI file format. Apart from generating melodies, this compiler is also able to play a specific melody and support the generation of MusicXML files from a musical score.

In order to elaborate this work, music theory had to be studied on the whole. Music scores presents a series of graphical symbols that can only be interpreted by someone who knows at least the main concepts of music theory. Score Input Languages were also contemplated. The main task of these Composition Languages is to make either a graphical or a musical analysis of a score.

The general structure of a compiler was briefly explained as well as the tools and APIs that were used on the development of the compiler Solfeggiare. The JDOM and JFugue API were the main APIs used on this work. The former is responsible for the front-end of the compiler. The latter takes part on the back-end.

A deep study over the MIDI protocol is also included on this work. The way its hardware works, what's its specification and the pattern of the MIDI files were crucial to the development of Composition Languages.

At the end, a case study is presented with a small part of the song "Este Seu Olhar" composed by Tom Jobim to validate the implementation of the compiler.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

OCR – Optical Character Recognition
MIDI – Musical Instrument Digital Interface
API – Application Programmin Interface
XML – eXtensible Markup Language
ASCII – American Standard Code for Information Interchange
DTD – Document Type Definition
HTML – Hypertext Markup Language
SGML – Standard Generalized Makup Language
WEB – World Wide Web
DOM – Document Object Model
SAX – Simple API for XML
UML – Unified Modelling Language
DAC – Digital-to-Analog Converter
VCO – Voltage-Controlled Oscilltor
VCF – Voltage-Controlled Filter
VCA – Voltage-Controlled Amplifier
LED – Light Emitting Diode
UART – Universal Asynchronous Receiver/Transmitter
SMF – Standard MIDI Files
SMDL – Standard Music Description Language

Agradecimentos

Em primeiro lugar a Deus por ter-me feito chegar até onde cheguei e por estar sempre presente em meus momentos mais difíceis.

Aos meus pais, Dagoberto Alves Costa e Terezinha Nunes da Costa, por terem sempre sido um exemplo de profissionalismo e ética, de onde pude receber os mais preciosos ensinamentos de vida.

À minha irmã Karina Nunes, por todo o apoio e carinho, e por me fazer entender o que realmente é ser um irmão.

À minha tia Maria de Lourdes Nunes Gentil, por sempre ter acreditado no meu potencial como cantor e ter sempre me estimulado a estudar música.

Ao professor Dr. Ricardo Massa Ferreira Lima, por ter sempre sido um exemplo de professor e profissional, aceitando orientar um trabalho em uma área diferente de sua linha de pesquisa pelo simples prazer de pesquisar. Levarei seus ensinamentos pelo resto de minha vida acadêmica.

Ao professor Dr. Carmelo José Albanez Bastos Filho por ter aceitado me co-orientar neste trabalho e por tê-lo feito de maneira magnífica. Seu entusiasmo e profissionalismo foram essenciais para que eu conseguisse terminar este trabalho.

Ao professor Dr. Francisco Heron de Carvalho Junior por ter me dado a primeira oportunidade de realizar um trabalho de pesquisa e por não hesitar em me ajudar em todos os momentos de dúvida.

A Leila Idrissova, Sílvia Tavares, Priscilla Sarmento e Rêmulo Caminha, amigos de tantos anos e que sempre estiveram comigo em todos os momentos. O verdadeiro significado da palavra amizade eu aprendi com vocês.

A Adélia Barros, Milena Rodrigues, Nivia Quental e Cleyton Mário, preciosos amigos que fiz durante meus anos de faculdade e que me aturaram em todos os momentos mais difíceis da minha vida acadêmica. Vocês viverão no meu coração para sempre.

À todos os artistas e músicos, por terem feito a minha vida mais cheia de alegria e fantasia. Por me fazerem acreditar em algo mágico, que só se revela no palco, no íntimo contato entre o artista e a platéia. Isso é o que faz tudo valer a pena.

Capítulo 1

Introdução

A música sempre esteve presente na humanidade, expressando e provocando os sentimentos mais diversos. Porém, a partir de sua formalização, foi possível reproduzí-la com alto grau de fidelidade. Apenas algumas sociedades conseguiram formalizar o processo musical, tendo como destaque a sociedade ocidental. A notação musical utilizada atualmente é conhecida como "notação musical do ocidente" e, graças a ela, a música hoje pode ser compartilhada por milhões de pessoas.

Com os avanços computacionais, foi inevitável fazer com que esta forma de expressão fizesse parte desse ambiente inovador. O áudio digital foi o grande precursor para a computação musical [1] e a partir de sua evolução é que aplicações mais ousadas puderam ser desenvolvidas.

Linguagens formais proveram flexibilidade e precisão à especificação musical, uma vez que símbolos abstratos são utilizados para representar peças musicais [1]. Dentre as principais vantagens desta formalização está o maior controle que o compositor tem sobre o que está sendo executado, podendo inserir na música trechos extremamente complexos com apenas alguns comandos. Todavia, nem sempre flexibilidade é sinônimo de simplicidade. Simples harpejos podem se tornar mais complexos do que deveriam ser, exigindo, dessa forma, que o compositor tenha noções de tratamento e modulação de sinais para poder obter o som de uma simples nota. Dentro desse escopo de problema se encontra a área de *Score Input Languages* (linguagens que recebem como entrada partituras musicais), principal foco deste projeto, juntamente com a elaboração de um compilador de partituras musicais.

As *Score Input Languages* pretendem abstrair ao máximo os comandos de criação de melodia no computador. O propósito é prover uma interface simplificada para compositores, se aproximando ao máximo da notação de partituras que eles estão acostumados.

1.1 Motivação

Solfejar é uma das tarefas mais difíceis para os iniciantes dos estudos musicais. Essa atividade nada mais é do que o ato de ler uma partitura. Entenda-se "ler" nesse contexto como sendo identificar as notas no pentagrama, que será explanado no Capítulo 2, e cantá-las de acordo com o que está descrito no papel. Essa atividade exige rápido raciocínio e profundo conhecimento de todos os símbolos representativos que podem vir a surgir em uma partitura. Todo bom músico

deve aprender a solfejar partituras e treinar sempre para melhorar sua habilidade e facilitar o aprendizado de instrumentos musicais ou técnica vocal.

A pesquisa por novas ferramentas em *Score Input Languages* passou muito tempo estagnada. A crença de que ferramentas tipo OCR[1] substituiriam estas linguagens congelou as pesquisas e impediu o desenvolvimento de novas aplicações e de melhorias, uma vez que a responsabilidade pela intepretação e execução de uma partitura seria dessas ferramentas de reconhecimento óptico.

Uma das grandes motivações para o uso do MIDI [2] é sua gama imensa de possibilidades de descrição de peças musicais o que fornece um maior realismo aos arquivos descritos na mesma. Por possuir essa característica, essa interface musical foi adotada por toda a comunidade científica e hoje em dia é suportada pelos mais diversos dispositivos eletrônicos.

1.2 Objetivos

O objetivo principal deste trabalho é elaborar um compilador de partituras, por sua parte uma *Score Input Language* e realizar um estudo mais aprofundado em Linguagens Musicais, focando em sua abordagem de síntese de partituras. Da mesma forma, dissertar sobre a teoria de compiladores com o intuito de contextualizar o leitor do que será desenvolvido ao fim deste trabalho.

A elaboração de um compilador de partituras musicais que deverá ser capaz de interpretá-las e gerar um arquivo de saída no formato MIDI será descrita neste trabalho. Para tal, se fará necessário um estudo aprofundado de como se dá a construção de arquivos MIDI e como os mesmos são executados.

A busca por ferramentas de desenvolvimento e *frameworks* que facilitem o processo de desenvolvimento do compilador também será relatada neste trabalho, procurando fazer uma análise comparativa entre elas, explicitando quais as vantagens e as facilidades de cada uma. A Figura 1 representa o esquema do compilador.

Figura 1. Esquema do compilador *Solfeggiare.*

[1] Optical Character Recognition

1.3 Metodologia

Inicialmente, faz-se necessário um apanhado do que está sendo desenvolvido na área de Linguagens Musicais e compiladores e em que estágio se encontram as pesquisas. Após a fase de contextualização terá início o desenvolvimento do compilador de partituras. As partituras a serem interpretadas deverão estar no formato MusicXML [3] definido pela "Recordare", um padrão aberto que está sendo utilizado por várias aplicações de cunho musical. Apesar de possuir um amplo suporte aos mais variados elementos de uma partitura, apenas os elementos essenciais serão utilizados neste trabalho.

A API que será utilizada para construir o compilador chama-se JDOM [4], uma API exclusiva Java para leitura e escrita de documentos XML e que também os valida de acordo com arquivos de validação. A partir da obtenção das informações do XML, uma série de objetos Java será criada e posteriormente utilizada para a elaboração do arquivo MIDI.

Quanto à API de geração dos arquivos MIDI, foi escolhida a API "JFugue". "JFugue" [5] é uma API de código aberto disponível na Internet e bastante intuitiva, pois se utiliza de símbolos representativos bastante conhecidos pela maioria dos músicos.

Todo o compilador será desenvolvido a partir da ferramenta de desenvolvimento chamada "Eclipse" [6]. O "Eclipse" é uma comunidade de código aberto cujos projetos são focados em fornecer uma ferramenta de desenvolvimento de aplicações e software em Java. Sua principal característica é a de ser flexível e altamente extensível através da utilização de *plug-ins* os quais adicionam funcionalidades à plataforma facilitando o processo de desenvolvimento para o usuário.

1.4 Organização do Trabalho

Este trabalho está dividido em 6 capítulos. O Capítulo 2 aborda um pouco de teoria musical bem como elementos computacionais para a manipulação de música. O Capítulo 3 procura explanar sobre compiladores bem como sobre as ferramentas utilizadas neste trabalho para a construção do compilador *Solfeggiare*. Uma especificação da interface MIDI será apresentada no Capítulo 4. No Capítulo 5 a ferramenta *Solfeggiare* é apresentada juntamente com um estudo de caso. A conclusão é o último capítulo e apresenta metas alcançadas, trabalhos futuros e resultados.

Capítulo 2

Teoria Musical e Score Input Languages

Segundo Maria Luisa Priolli, “Música é a arte dos sons, combinados de acordo com as variações de altura, proporcionados segundo a sua duração e ordenados sob leis da estética” [7]. Ela define a música como sendo, além de uma arte, um processo matemático, pois se utiliza de proporções e ajustes para expressar suas peças musicais.

O processo de formalização da expressão musical se deu durante vários anos e a especificação que utilizamos hoje com a notação de partituras e pentagramas, vide Figura 2, é chamada de notação musical do ocidente. O pentagrama são cinco linhas horizontais, paralelas e eqüidistantes, formando entre si quatro espaços [7]. A maior parte das culturas desenvolveu seu próprio sistema musical, contudo nem todas atribuíram grande importância à escrita musical. Boa parte da música não-ocidental e da música ocidental são transmitidas de pessoa para pessoa simplesmente pela tradição oral. Tal circunstância pode tornar a música mais viva e mais vibrante. Todavia este fato também pode explicar o desaparecimento de peças musicais belas e preciosas [8].

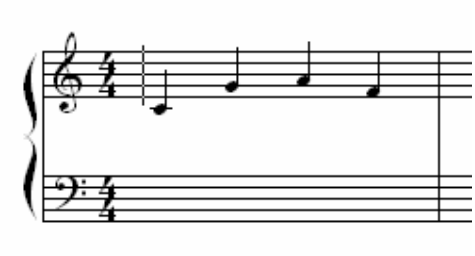

Figura 2. Exemplo de partitura com clave de Sol e de Fá.

A ciência e a tecnologia enriqueceram bastante o processo de composição musical e abriram uma nova porta para os compositores. A melodia ou os arranjos compostos podem ser ouvidos praticamente na hora em que são inseridos, além de possibilitarem a utilização de vários instrumentos diferentes na execução de uma partitura sem requerer a presença de um músico que a execute. O processo de composição é todo controlado pelo usuário. Este pode realizar os mais variados tipos de manipulação e orquestração a um custo relativamente baixo.

Para poder prover esse tipo de abstração ao compositor, onde ele visualiza a música composta como uma partitura, uma linguagem de programação musical teve de ser desenvolvida

para servir de interface entre o ser humano e os dispositivos eletrônicos responsáveis pela execução do som. A principal vantagem da inserção de linguagens de programação formais no processo de composição está na precisão e flexibilidade que elas provêm à especificação musical [1].

As linguagens formais de especificação musical que tratam diretamente uma partitura como entrada e geram música como saída são chamadas de *Score input languages*. Nas seções seguintes conceitos sobre teoria musical serão abordados. Também será explanada uma breve descrição das linguagens de entrada de partitura.

2.1 Teoria Musical

A música é composta de sons e a representação gráfica destes sons se chama notas musicais. São sete as notas musicais: Dó, Ré, Mi, Fá, Sol, Lá e Si. O conjunto sucessivo de notas se chama escala, como mostrado na Figura 3. As notas são escritas nas linhas e nos espaços definidos pelo pentagrama.

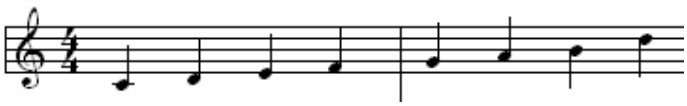

Figura 3. Escala de Dó representada em clave de Sol.

Todavia, a pauta não é suficiente para expressar qual o tipo e a altura da nota em um contexto. Claves são os símbolos utilizados para definir de que tipo são as notas que estão dispostas no pentagrama. A clave de Sol sempre aparece no pentagrama na segunda linha, de baixo para cima, e ela define que todas as notas que estiverem na segunda linha são notas 'Sol'. A clave de Fá pode aparecer tanto na quarta como na terceira linhas e ela define que todas as notas que estiverem em uma destas duas linhas são um 'Fá'. O mesmo raciocínio serve para a clave de Dó que pode aparecer tanto na primeira, quanto na segunda, na terceira e na quarta linhas. Na Figura 4 estão representados os símbolos utilizados para designar as escalas de "Sol" e "Fá".

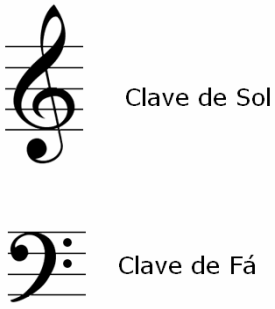

Figura 4. Símbolos representativo para a clave de Sol e de Fá.

2.1.1 Notas, Pausas e Acidentes

Os tempos de execução de notas são representados por símbolos diferentes que denotam diferentes frações de tempo em um compasso. Um compasso é um escopo de tempo definido por uma fração, como podemos ver na Figura 2. A fração 4 por 4 define que um compasso será composto por quatro notas onde cada uma é um quarto de tempo, ou seja, serão quatro semínimas. Na partitura, os compassos são delimitados por linhas verticais no pentagrama. A Tabela 1 possui o mapeamento de notas, seus nomes e suas respectivas frações de tempo.

Mas música não é somente som. O silêncio também faz parte do processo de tensão e relaxamento inerente da música. Assim como as notas, as pausas também possuem uma representação específica e estão associadas a frações de tempo. A Tabela 2 possui o mapeamento das representações para pausas, com suas designações e frações de tempo correspondentes.

Tabela 1. Tipos de notas.

Nota	Nome	Fração de Tempo
	Semibreve	1
	Mínima	1/2
	Semínima	1/4
	Colcheia	1/8
	Semicolcheia	1/16
	Fusa	1/32
	Semifusa	1/64

Tabela 2. Tipos de pausas.

Pausa	Nome	Fração de Tempo
	Pausa de Semibreve	1
	Pausa de Mínima	1/2
	Pausa de Semínima	1/4
	Pausa de Colcheia	1/8
	Pausa de Semicolcheia	1/16
	Pausa de Fusa	1/32
	Pausa de Semifusa	1/64

Cada duas notas definem intervalos musicais, por exemplo, entre a nota Dó e a nota Ré existe um intervalo de 2ª maior. O intervalo seria a diferença de altura de som entre as notas. Na

escala diatônica de Dó (Dó, Ré, Mi, Fá, Sol, Lá, Si e Dó), onde não há acidentes, os intervalos são de tom, tom, semitom, tom, tom, tom e semitom, ou seja, de Dó para Ré existe um intervalo de tom enquanto que de Mi para Fá existe um intervalo de semitom. Estes são os intervalos-padrão definidos em uma escala sem acidentes. Os acidentes definem novos intervalos entre as notas que podem ser suprimidos ou aumentados.

O acidente sustenido, que é representado pelo símbolo #, aumenta a altura de uma nota em meio tom, ou seja, um intervalo entre Dó# e Ré não tem mais um tom de diferença, mas sim um semitom. O acidente bemol que é representado pelo símbolo *b*, diminui a altura de uma nota em meio tom, ou seja, um intervalo entre Mi*b* e Fá não tem mais um semitom e sim um tom.

2.1.2 Ligaduras e Pontos de Aumento

A notação musical do ocidente possui elementos que permitem ao compositor aumentar o tempo de execução de uma nota específica. Com isso, menos notas são escritas em um mesmo compasso, tornando a partitura mais enxuta e legível.

Arcos que ligam notas em uma partitura são conhecidos como ligaduras. As ligaduras podem ser de: aumento de tempo e execução [7]. As ligaduras de aumento de tempo são utilizadas em notas que possuem uma mesma entonação, ou seja, possuem a mesma altura. As ligaduras indicam um acréscimo no tempo de execução de uma nota. Já as ligaduras de execução são utilizadas para dar um efeito de notas executadas umas após as outras, dando a sensação de estarem efetivamente ligadas. Na Figura 5, o primeiro compasso mostra o exemplo de uma ligadura de tempo onde duas notas semínimas 'Sol' estão conectadas pelo arco. No terceiro compasso da mesma figura, o arco está conectando duas notas semínimas que possuem alturas diferentes, uma é 'Ré' e a outra é 'Mi'. Quando este compasso for executado não haverá a sensação de interrupção de som entre as notas ligadas.

O ponto de aumento é colocado à direita de uma figura representativa de uma nota e serve para aumentar em metade do valor a duração desta nota [7]. Na Figura 5, temos o exemplo de uma semínima pontuada no segundo compasso. A semínima pontuada deixa de possuir o valor de tempo de 1/4 para ter o valor de 3/8. Para completar o tempo do compasso basta adicionarmos uma colcheia com valor de 1/8 somado com a mínima de valor 1/2 que já existia no mesmo compasso.

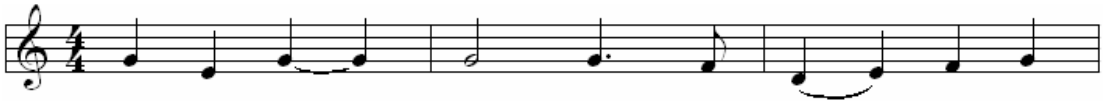

Figura 5. Ligaduras e pontos de aumento em uma partitura.

2.1.3 Acordes e harpejo

As notas musicais podem ser executadas tanto simultaneamente quanto seqüencialmente. Às notas executas de forma seqüencial dá-se o nome de harpejo. Ao conjunto de mais de três notas executadas de maneira simultânea dá-se o nome de acorde. A Figura 6 exemplifica graficamente estes dois conceitos.

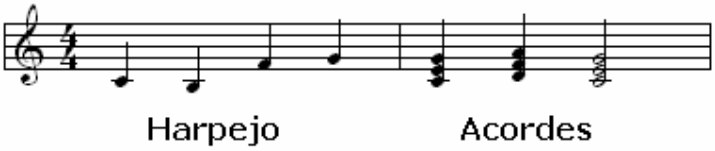

Figura 6. Representação de um harpejo e de acordes em uma partitura.

Os acordes representam, geralmente, a harmonia de uma peça musical dando-lhe um alicerce e um sentido de preenchimento [8]. As melodias têm que se encaixar no contexto (entenda-se "contexto" aqui com sendo a tonalidade[2] da música) gerado pela harmonia sobre pena de desafinação caso não haja esse casamento.

As "tríades", vide Figura 7, são as diferentes formações dos acordes. Elas possuem este nome por serem um conjunto de três notas distintas que são executadas simultaneamente. Suas três notas constituintes são a fundamental, nota mais grave e que dá o nome ao acorde; a 3ª (terça) chamada nota modal, que determina o caráter do acorde (maior ou menor); e a 5ª (quinta). As tríades podem ser: maior, menor, diminuta ou aumentada. A tríade diminuta se caracteriza pela presença de dois intervalos de terça menor entre as notas quinta e terça e entre a terça e a fundamental. Já a aumentada se caracteriza pela presença de intervalos de terça maior entre essas notas.

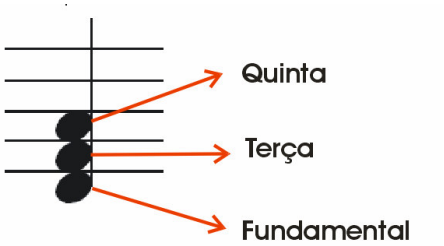

Figura 7. Exemplo de tríade com suas notas específicas.

2.1.4 Notação ABC

ABC é uma linguagem para notação de música - melodia, letra e cifra - usando caracteres em formato ASCII [9]. Foi proposta em 1991 por Chris Walshaw para melodias folclóricas da Europa Ocidental. Pela sua facilidade de escrita e entendimento, ela foi adotada por uma grande parte das aplicações computacionais.

As cifras de músicas, que descrevem a harmonia da mesma, ajudaram este tipo de formatação a se popularizar. Os caracteres descritos na Tabela 3 aparecem entre a letra da música, em uma linha específica situada acima dos versos. Outros caracteres podem aparecer agregados a estes símbolos para denotar notas a mais que foram inseridas na tríade ou mesmo representar diferentes tipos de tríade de um mesmo acorde.

[2] Escala musical em que uma peça musical é composta.

Tabela 3. Notação "ABC".

Símbolo	Nota
A	Lá
B	Si
C	Dó
D	Ré
E	Mi
F	Fá
G	Sol

Melodias também são descritas com a notação ABC. O modelo MusicXML e a API JFugue, que serão explanados mais a frente, se utilizam do mesmo para representar notas específicas de uma peça musical.

2.2 Linguagens de Composição: Score Input Languages

Dentre os diferentes tipos de linguagens de composição musical, destacam-se neste trabalho as linguagens chamadas *Score Input Languages*. A idéia principal dessa linguagem está no fornecimento de dados como música. Partituras prontas são dispostas na entrada, e como saída obtêm-se a peça musical executada, ou em forma de arquivo ou instantaneamente. As *Score Input Languages* mais conhecidas são a DARMS [10], MUSTRAN [11] e FORMULA [12]. A última, além de suportar a leitura de partituras como entrada, possui um mecanismo especial de controle de sintetizadores musicais com MIDI. As duas primeiras foram desenvolvidas com o intuito de arquivamento ou simples análise de uma partitura [1].

Devido à alta complexidade de se codificar uma partitura de forma a ser interpretada pelo computador, alguns estudiosos, como Milton Babbit em 1965, argumentaram que as *Score Input Languages* seriam futuramente substituídas por ferramentas de reconhecimento óptico de caracteres (OCR) [13] que leriam as partituras e as executariam. De fato alguns trabalhos foram realizados nesta área como o proposto por Entwistle [14]. O que alcançou maior visibilidade foi o WABOT-2 [15] que lê a partitura, interpreta-a e a executa. Alguns outros trabalhos estão em andamento nesta área, porém nada que chegue a substituir as *Score Input Languages*.

Uma característica dessas linguagens é que elas geralmente utilizam uma representação interna similar à representação musical utilizada nas partituras. No exemplo da Figura 8, podemos verificar que a *string* de caracteres utilizada para representar a partitura se assemelha ao que está descrito na mesma. Este exemplo está codificado de acordo com a linguagem musical Musica [16]. No primeiro compasso, o número 4 representa a duração da nota, neste caso uma semínima. A letra *A* representa um Lá e o apóstrofo que aparece em sua frente indica um Lá executado na escala principal de Dó. O resto da *string,* "GAG", indica as quatro notas subseqüentes. Elas não possuem indicador de duração, pois se trata de semínimas. Nota do mesmo tipo da primeira. A barra invertida "/" é utilizada para separar compassos. A mesma idéia é repetida para os demais compassos presentes.

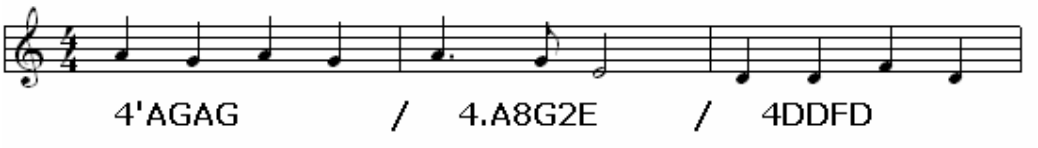

Figura 8. Exemplo de notação de partitura na linguagem musical "Musica".

Problemas de desempenho musical de uma partitura são os mais comuns e mais citados quando se fala de *Score Input Languages*. A simples leitura e execução de uma partitura por um computador gera uma melodia mecanizada, sem elementos de interpretação, fazendo com que o som executado tenha um aspecto artificial. Existe uma série de elementos performáticos que podem estar presentes em uma partitura, porém nem todos são suportados pelas *Score Input languages*. Existe um esforço para que as melodias executadas por computador se tornem cada vez mais fiéis e naturais, porém essa discussão está fora do escopo deste trabalho.

Em contrapartida, as *Score Input Languages* possibilitaram aos compositores um controle maior sobre os diferentes processos que estão sendo executados. Um exemplo está na possibilidade de gerar um acompanhamento musical automático de acordo com uma entrada melódica. Na tese de doutorado defendida por Ramalho [17] um acompanhamento de baixo é gerado automaticamente para melodias de Jazz.

2.2.1 Impacto do MIDI nas Score Input Languages

Com o advento do protocolo MIDI, que será explicado no Capítulo 4, muito se especulou sobre o desaparecimento das *Score Input Languages*. O MIDI abriu novas possibilidades de composição musical através de seqüenciadores ou editores gráficos. Todavia esse protocolo ainda impõe uma série de limitações à expressão musical. Muitos dos símbolos performáticos presentes em uma partitura não podem ser expressos por MIDI nem existe nenhum padrão em vista que venha a inserir este tipo de suporte. Por isso, espera-se que as *Score Input Languges* ainda sejam bastante utilizadas até que se desenvolvam leitores confiáveis de partituras.

Capítulo 3

Compiladores e Ferramentas para compilação de dados

O desenvolvimento de *software* em larga escala pela indústria de TI se deu pelo advento de ferramentas que possibilitassem aos desenvolvedores abstrair a linguagem de máquina. Linguagem de máquina é a linguagem interpretada pelos computadores, descrita em forma de *bits* e *bytes*, tornando a leitura por um ser humano um trabalho árduo. Linguagens de alto nível utilizam uma estrutura padronizada para escrever programas, facilitando o entendimento do código. Os compiladores ou tradutores são programas que aceitam dados em representação textual (algoritmos), chamada "código fonte" ou "linguagem de alto nível" e que produzem uma representação desses mesmos dados em uma outra linguagem que seria a linguagem alvo [18].

O processo de compilação é totalmente escondido do usuário final, uma vez que para ele apenas o resultado interessa e não necessariamente quais passos são executados. Um compilador deve possuir as seguintes características:

- *Legibilidade* – as linguagens de alto-nível devem possuir uma sintaxe que lembre bastante uma língua falada, como o Inglês. A possibilidade do código escrito na linguagem de um compilador ser quase auto-documentável faz com que se diminua as ambigüidades e que o código seja mais compreensível e depurável;
- *Portabilidade* – a saída gerada pelo compilador deve ser compatível com o maior número de máquinas possível. Diante da heterogeneidade que existe hoje de *hardware*, é impraticável desenvolver um compilador que gere saídas compatíveis apenas com uma determinada arquitetura. Muito provavelmente o compilador cairia em desuso;
- *Estrutura e orientação a objeto* – é notório que o desenvolvimento de código de forma estruturada ou orientada a objeto melhorou bastante a leitura ou legibilidade dos códigos desenvolvidos;
- *Generalidade* – as linguagens devem ser capazes de gerar aplicações de propósito mais geral possível, sem se restringir a nenhum domínio específico;

- *Brevidade* – os programas desenvolvidos devem poupar os desenvolvedores de ter que escrever muitas linhas de código para conseguir executar tarefas simples;
- *Checagem de erro* – o compilador deve identificar erros em tempo de compilação.

A presença de tais características é desejável e não necessariamente obrigatória para que um compilador gere uma saída aceitável. Nem sempre todos esses requisitos podem ser satisfeitos, porém todos têm que ser levados em consideração durante a elaboração de um compilador.

3.1 Processo de Compilação

De modo geral, os passos que um compilador executa para obter um resultado final podem ser resumidos em três etapas distintas: análise léxica, análise sintática e análise semântica.

Na "análise léxica" o compilador interpreta os dados de entrada, que geralmente são *strings* de *bytes* ou caracteres, e os transforma em *tokens*. Os *tokens* são macros que identificam e separam os dados de entrada. Alguns caracteres podem ter um sentido próprio como as chaves '{' e '}' ou podem fazer parte de uma unidade maior como o caractere 'y' em uma variável de programa "xyz" [19].

É comum a utilização de autômatos finitos para identificar padrões de entrada. Os diferentes estados finais do autômato representam os vários *tokens* que são definidos pelas linguagens.

A "análise sintática" é a fase que vem após a análise léxica. De posse de todos os *tokens*, o compilador irá checar se a ordem dos mesmos está de acordo com a gramática definida pela linguagem. Para realizar tal tarefa, uma árvore de análise também conhecida como *parse tree* é montada. A árvore de análise é utilizada para definir em que ordem os comandos devem ser executados e para desfazer possíveis ambigüidades.

Uma estrutura hierárquica de comandos, declarações e expressões devem ser definidas para a linguagem e, a partir destas regras, a árvore de análise gramatical é estabelecida. *Context free grammars* ou gramáticas livres de contexto são aquelas que não permitem que apareçam ambigüidades, ou seja, não importa a ordem em que a avaliação seja feita na cadeia de comandos, uma mesma saída sempre será gerada.

Por fim, a "análise semântica" procura atribuir tipos específicos a determinados comandos ou variáveis. A checagem de tipo faz parte do processo de compilação, porém pode ser mais ou menos rigorosa, dependendo das regras definidas pelo compilador.

Para o *Solfeggiare*, as etapas do processo de compilação se dão de forma mais simplificada. As fases de análise léxica e sintática são realizadas inerentemente pelo documento XML de descrição da partitura, através de um arquivo DTD cujo detalhamento será realizado na próxima seção. Após o final do processo executado pelo *front-end* (análise léxica, sintática e semântica responsável pela geração do código intermediário), o *backend* (interpretação do código intermediário advindo do *front-end* e geração do arquivo de saída) será o responsável pela geração dos arquivos MIDI.

3.2 XML e MusicXML

XML ou *eXtensible Markup Language* é um formato independente de plataforma para descrição dos dados que podem ser usados em muitos aplicativos [20].

Por possuir esta característica, o XML hoje é utilizado pela grande maioria das novas aplicações desenvolvidas, principalmente pelo fato de conseguir integrar duas plataformas diferentes de forma automática, sem obrigar os desenvolvedores a forçadamente alterar suas plataformas para que elas possam se comunicar. Assim como o HTML (*Hypertext Markup Language*), o XML é uma linguagem utilizada para transporte de metadados que propiciam informações sobre a estrutura dos dados, porém o XML se mostra mais poderoso quando permite que o usuário crie e insira seus próprios elementos.

O XML se originou de uma linguagem anterior chamada SGML (*Standard Generalized Markup Language*), uma linguagem-padrão complexa. Como seu próprio nome diz, ela é uma linguagem de marcação, onde a marcação representa informações adicionais inseridas em um texto que não acrescentam conteúdo. As marcações de SGML são utilizadas para informar a aparência de um determinado trecho do texto ao invés de adicionar informação.

O ponto principal de XML está justamente no fato de ela ser extensível, daí se chamar *eXtensible*. Um documento XML é em sua totalidade organizado a partir de *tags*. Essas *tags* indicam diferentes elementos do documento e estabelecem uma hierarquia. As *tags* utilizadas nos arquivos XML são definidas pelo desenvolvedor e encaixadas à medida que forem necessárias. A *tag* principal de um documento XML é formalmente conhecida como *tag root*[3]. Para documentos HTML o cenário que temos hoje é diferente. Os navegadores ou *browsers* que interpretam estes documentos exigem uma ordem predefinida das *tags* HTML para que eles possam interpretá-las. Essa flexibilidade do XML pode também gerar o caos se não houver nenhum mecanismo de validação para estes arquivos. A maneira mais utilizada de se validar os dados de um arquivo XML bem como sua estrutura, é com arquivos DTD (*Document Type Definition*). Existem outras formas de se validar um arquivo XML como "*XML schema*", mas não entraremos em detalhes uma vez que não faz parte do escopo do nosso projeto. A Figura 9 a seguir mostra um exemplo de um arquivo XML com sua referência ao documento DTD de validação.

O XML para o *Solfeggiare* será o modelo utilizado para descrição de uma partitura. O padrão MusicXML [3] define um modelo hierárquico para descrição de partituras, assim como uma série de documentos DTD para validação do mesmo.

Com o crescente número de programas de edição musical, percebeu-se que não existia, além do MIDI, nenhum outro formato que permitisse a troca de informações musicais. O MIDI, apesar de ser bastante poderoso, não dá suporte à notação musical como, por exemplo, partituras. O MusicXML surgiu para atender a demanda por um formato que pudesse ser mais explícito quanto às notas e tons de uma peça musical. Também vale salientar que documentos XML são muito utilizados em aplicações *WEB*[4] facilitando a troca de informações.

O formato definido pelo MusicXML é universal e atualmente aceito pela maioria dos *softwares* de edição/criação de partituras (ou *Western music notation*, como explicado no Capítulo 2).

---

[3] Elemento principal de um arquivo XML. Na hierarquia é o que engloba todos os outros.

[4] WEB – rede mundial de computadores também conhecida por *World Wide Web*.

3.2.1 Descrevendo o MusicXML

O MusicXML é inerente do XML e capaz de representar tanto partituras como tablaturas[5] (formato simplificado para representação de notas e acordes musicais em um instrumento). Seu elemento principal (ou *tag root*) está definido como *score-partwise* ou *score-timewise*. Basicamente, a diferença entre eles está na forma como a partitura foi transcrita. O primeiro elemento divide uma partitura em partes que são divididas em compassos. O segundo divide-se em compassos e os compassos se dividem em diferentes partes [3]. Cabe ao desenvolvedor ou à aplicação decidir qual será o melhor modelo para representar a partitura.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 1.0 Partwise//EN"
"http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
  <movement-title>Nome da Peça</movement-title>
  <identification>
    <creator type="composer">Compositor</creator>
    <creator type="transcriber">Transcritor</creator>
  </identification>
  <part-list>
    <score-part id="P1">
      <part-name>Nome da Parte</part-name>
      <score-instrument id="P1-I3">
        <instrument-name>Instrtumento da Parte</instrument-name>
      </score-instrument>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>Divisões das notas semínimas</divisions>
        <key>
          <fifths>Índice do ciclo das Quintas</fifths>
          <mode>Modo</mode>
        </key>
        <time>
          <beats>Batidas</beats>
          <beat-type>Tipo da Batida</beat-type>
        </time>
        <clef>
          <sign>Clave</sign>
          <line>Linha da Clave</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>Tipo da Nota</step>
          <octave>Oitava da Nota</octave>
        </pitch>
        <type>Duração da Nota</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

Figura 9. Esquema de documento XML. No exemplo um arquivo MusicXML.

Toda partitura descrita neste formato possui uma seção especial contendo informações específicas do tom em que a melodia está descrita, divisão de tempo e demais elementos característicos de uma peça. Todos eles estão contidos dentro do elemento *measure*, que possui o

[5] O enfoque deste trabalho estará nas partituras, por isso não nos aprofundaremos em tablaturas.

atributo *number* com valor 1/um. A Figura 10 mostra o trecho do documento correspondente ao primeiro compasso.

O elemento do arquivo MusicXML que engloba todas as informações de um compasso se chama *measure*. Nele podemos encontrar todas as informações das notas ou pausas presentes em um compasso. Cada nota ou pausa é representada em um compasso como um elemento *note*. A pausa possui um comportamento especial e é representada pela presença de um elemento *rest* em um elemento *note*. Já as notas são representadas por elementos *pitch* que também se localizam em um elemento *note*. Cada elemento *pitch* possui qual a nota (elemento *step*) que deverá ser executada assim como qual a oitava associada a ela (elemento *octave*).

Uma nota pode possuir diversos elementos alteradores como sustenidos, bemóis, fermatas e sinais de intensidade (para o nosso caso estaremos utilizando apenas os sustenidos e bemóis. A representação deles se dá através do elemento *alter* dentro de um elemento *note*). O *alter* possui um valor inteiro, em que o sinal representa sustenido (+) ou bemol (-) e o algarismo representa a quantidade de alterações presentes na nota.

Ligaduras, como definido no Capítulo 2, são representadas pela presença de elementos *tie* em elementos *note*. O início de uma ligadura é demarcada pelo elemento *tie*, que possui o atributo *type* definido como "*start*". Já o fim de uma ligadura se dá pela presença do mesmo elemento *tie* só que com o atributo *type* definido como "*stop*". Acordes também são suportados pelo MusicXML e são representados pela presença do elemento *chord* sem qualquer valor associado.

```
<measure number="1">
  <attributes>
    <divisions>2</divisions>
    <key>
      <fifths>3</fifths>
      <mode>major</mode>
    </key>
    <time symbol="cut">
      <beats>2</beats>
      <beat-type>2</beat-type>
    </time>
    <clef>
      <sign>G</sign>
      <line>2</line>
    </clef>
  </attributes>
  <note>
    <rest/>
    <duration>2</duration>
    <voice>1</voice>
    <type>quarter</type>
  </note>
</measure>
```

Figura 10. Informações gerais da partitura no MusicXML.

3.2.2 MusicXML e MIDI

Como o protocolo MIDI, que será visto no Capítulo 4, é bastante utilizado e aceito, o MusicXML possui uma definição específica para tratar arquivos deste tipo. Uma séria de elementos extra foram inseridos à definição original do MusicXML para dar suporte ao MIDI.

A *tag sound* é utilizada especificamente para representar uma mensagem MIDI que informa ao sintetizador o tempo que deverá ser associado a uma nota semínima. O valor presente no atributo *tempo* é um inteiro com os pulsos associados à semínima. Outras *tags* também estão associadas aos arquivos MIDI como o elemento *score-instrument*. Para cada voz/melodia

presente na partitura um instrumento MIDI diferente pode ser associado e seu respectivo canal MIDI pode ser definido na *tag midi-instrument*. A Figura 11 mostra um trecho de uma partitura que possui suporte a MIDI.

```
    <part-name>Voice</part-name>
    <score-instrument id="P1-I3">
      <instrument-name>Voice</instrument-name>
    </score-instrument>
    <midi-instrument id="P1-I3">
      <midi-channel>1</midi-channel>
      <midi-program>53</midi-program>
    </midi-instrument>
  </score-part>
  <score-part id="P2">
    <part-name>Piano</part-name>
    <score-instrument id="P2-I2">
      <instrument-name>Acoustic Grand Piano</instrument-name>
    </score-instrument>
    <midi-instrument id="P2-I2">
      <midi-channel>2</midi-channel>
      <midi-program>1</midi-program>
    </midi-instrument>
```

Figura 11. Trecho de um arquivo MusicXML com suporte a MIDI.

3.3 JDOM

Dentre todas as bibliotecas desenvolvidas para leitura de arquivos XML em Java, JDOM [4] é a que se mostra mais simples. Baseada no *Document Object Model* (DOM), esta biblioteca tem a preocupação de se parecer ao máximo com Java, tornando o processo de leitura e escrita de documentos XML mais intuitivo para o desenvolvedor. A Figura 12 mostra um diagrama UML de classes da biblioteca JDOM.

A API JDOM faz parte do projeto Apache [21], em conseqüência, é uma biblioteca *open source*[6] que pode ser utilizada em qualquer projeto que seja ou não *open source*.

Apesar de ser baseado em DOM, o JDOM possui algumas características do SAX (*Simple API for XML*), o que o torna mais atrativo. O DOM utiliza um esquema de árvore para representar um documento XML, porém, ele carrega todo o documento de uma vez na memória, o que pode se tornar um ponto fraco quando se têm grandes documentos XML que devem ser processados por máquinas com as mais diferentes arquiteturas. Já o SAX é mais leve, não carrega todo o documento de uma vez e trabalha com o esquema e eventos. Cada novo elemento encontrado no documento é representado por um evento, todavia, esta abordagem de eventos não é intuitiva para programadores Java e ainda não permite que se faça um acesso randômico ao documento, nem que se altere o mesmo. JDOM possui ambas as características principais dessas duas APIs, o esquema de árvore que permite que haja acesso randômico ao documento e é leve porque não carrega o documento XML todo na memória de uma vez.

[6] *Open source* – biblioteca de livre utilização por possuir uma licença pouco restritiva.

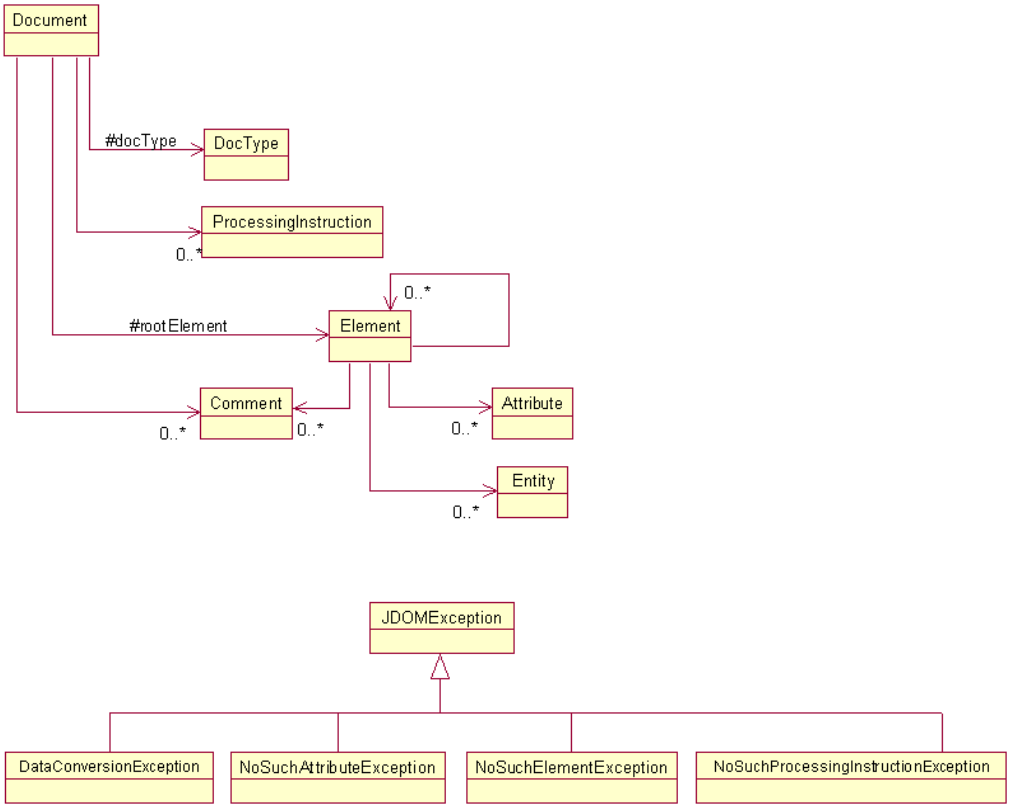

Figura 12. Diagrama de classes da biblioteca JDOM.

Como podemos observar através do modelo de classes UML (*Unified Modelling Language*) [22] do JDOM, sua estrutura é bastante simplificada. Os objetos do tipo *Document* são utilizados para criar uma instância de um documento XML que se deseja ler. A partir deste objeto, obtêm-se uma série de outras instâncias de classe que irão representar todos os elementos de um documento XML. Os mais importantes neste caso são: o *Element,* que é utilizado para representar *tags* do documento; e o *Attribute,* que representa os atributos presentes em cada *tag* ou *Element*. Também é válido citar o objeto *DocType,* que representa qual o documento que valida o XML que está sendo lido.

Como podemos observar pela Figura 13, o código JDOM é bastante simplificado para a leitura de um arquivo XML e se baseia na forma como Java funciona par dar ao desenvolvedor melhor entendimento e um aprendizado mais rápido.

```
File XMLFile = new File(adega);
SAXBuilder builder = new SAXBuilder(false);
Document adegaInXML = null;
try {
   adegaInXML = builder.build(XMLFile);
} catch (JDOMException jdome) {
   jdome.printStackTrace();
} catch (IOException ioe) {
   ioe.printStackTrace();
}
Element adegaRoot = adegaInXML.getRootElement();
Element vinho = adegaRoot.getChild("vinho");
Element nome = vinho.getChild("nome");
String nomeVinho = nome.getText();
String safra = nome.getAttributeValue("safra");
String uva = vinho.getChild("uva").getText();
```

```
<adega>
  <vinho>
    <nome safra="1998">Concha y Toro</nome>
    <uva>Merlot</uva>
  </vinho>
</adega>
```

Figura 13. Exemplo de código JDOM para leitura de arquivo XML.

3.4 JFugue

A API JFugue [5] foi desenvolvida para programação de música em Java, tendo um crescimento considerável nos últimos anos, principalmente por Java ser uma linguagem comumente aceita e reconhecida como ideal para programação das mais diversas aplicações. Há, contudo, algumas características que fazem de Java uma das linguagens mais usadas para desenvolvimento de softwares musicais como, por exemplo, o fato de incluir uma biblioteca para tratamento de som e mensagens MIDI (*Java Sound*) [23].

A JFugue se utilizou dessa infra-estrutura já fornecida por Java para fazer o seu alicerce e, a partir dela, estabeleceu suas próprias regras. Uma das principais vantagens do JFugue está em sua flexibilidade. A programação musical é toda feita a partir de *strings* que guardam as informações da melodia que deve ser executada. Um simples '*play("C")*' executa um dó central com som de piano que é o instrumento padrão do MIDI. Por ser baseada no *Java Sound*, JFugue possui suporte a MIDI.

3.4.1 Padrões e strings musicais

Todos os dados que são interpretados pela API JFugue são repassados através de *strings* ou *patterns*. Os *patterns* ou padrões identificam trechos de uma melodia que podem ser futuramente reutilizados em uma música como, por exemplo, os refrões. Eles são objetos construídos a partir de *strings* ou de outros padrões ficando a cargo do desenvolvedor da aplicação decidir qual abordagem se encaixa melhor. A seguir uma lista de comandos que podem existir em um padrão:

1. **Notas, acordes e pausas:** especifica nota ou acorde a ser tocado assim como o tempo associado ao mesmo.
2. **Tempo:** especifica a velocidade da música.
3. **Voz:** especifica qual voz (ou canal) em que uma melodia específica será executada.
4. **Mudança de instrumento:** modifica o instrumento que está sendo utilizado para a execução da melodia.

5. **Mensagens de controle:** especifica qualquer tipo de mensagem de controle do MIDI.
6. **Variáveis:** define uma variável que pode ser utilizada por outros comandos.

3.4.2 Notas e Pausas

As notas são representadas nos padrões e *strings* de acordo com a Tabela 4.

Tabela 4. Símbolos *versus* notas. Notação da API JFugue de acordo com os padrões da música ocidental.

Símbolo	Nota
A	Lá
B	Si
C	Dó
D	Ré
E	Mi
F	Fá
G	Sol

Tabela 5. Símbolos *versus* pausas. Representação do JFugue para pausas.

Símbolo	Pausa
w	Pausa de semibreve
h	Pausa de mínima
q	Pausa de semínima
i	Paula de colcheia
s	Pausa de semicolcheia
t	Pausa de fusa
x	Pausa de semifusa

Juntamente com o símbolo indicativo da nota, o tempo de execução da mesma deverá ser especificado (vide Tabela 5) assim como a oitava em que a nota será executada (vide Figura 14). As oitavas são representadas por um número inteiro inserido logo após o símbolo da nota e servem para representar qual a altura que a nota será executada. Acidentes como sustenidos (#) e bemóis (b) também são suportados pela API e devem ser adicionados às respectivas notas.

Notas combinadas também são suportadas como, por exemplo, ligaduras e acordes. As ligaduras expressam notas que devem ser tocadas em seqüência como se estivessem "ligadas". A *string* "X[LEGATO]=ON" é utilizada para representar o começo de um trecho de notas ligadas e a *string* "X[LEGATO]=OFF" denota o fim deste trecho. Exemplo: "X[LEGATO]=ON C5h E5q G5q X[LEGATO]=OFF". Já os acordes são notas que devem ser executadas simultaneamente. Apesar do JFugue possuir uma notação especial para representar acordes, neste projeto trataremos apenas a formação de acordes com o símbolo de adição (+). Exemplo: "C5q+E5q+G5q".

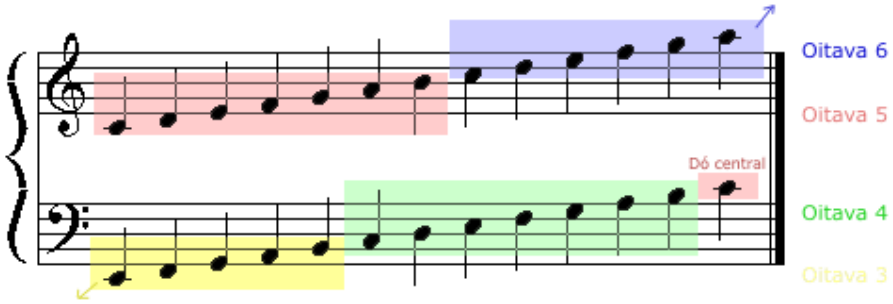

Figura 14. Compasso de uma partitura com a exemplificação das diferentes oitavas presentes em duas claves diferentes. Acima a clave de Sol e abaixo a clave de Fá.

3.4.3 Tempo

Os sinalizadores de tempo especificam qual a velocidade em que a música será executada em pulsos por semínima. Valores pequenos indicam que a música será executada mais rapidamente enquanto que valores maiores indicam uma música mais espaçada, com um intervalo de tempo maior, dando a impressão de ser executado de forma mais lenta.

O tempo pode ser especificado em qualquer parte de um padrão ou *pattern* de um programa descrito em JFugue e deve ser sempre representado da seguinte forma: “T120”. O símbolo “T” especifica que aquele elemento descreve informação de tempo da música e o inteiro que o segue indica os pulsos por semínima da melodia.

3.4.4 Voz

A API JFugue possui suporte a múltiplas vozes em uma partitura dando a sensação de polifonia, ou seja, mais de uma melodia sendo executada ao mesmo tempo. Em MIDI isso pode ser representado por diferentes canais e é justamente com este intuito que o JFugue possui este suporte a múltiplas vozes.

Existe uma gama de 16 vozes ou canais que podem ser utilizados. No padrão ou *pattern* isto é representado pela *string* “V0”. Neste caso indica que as notas que estão dispostas após este símbolo serão executadas no canal 1 (um) do MIDI. Lembramos que, assim como em MIDI, o canal 10 ou “V9” é reservado para instrumentos percussivos.

Dentre todos os elementos suportados pelo JFugue, os acima citados foram os mais utilizados e de maior importância para o desenvolvimento deste projeto. Uma série de elementos extras como mensagens de controle e instrumentação não foram aqui citados por se tratarem de comandos mais avançados e que não impedem o entendimento do funcionamento da API JFugue.

Todos os recursos descritos neste capítulo foram utilizados no compilador *Solfeggiare*, ajudando-o a se tornar simples e funcional sem prejudicar o resultado final, uma melodia simples e corretamente executada.

Capítulo 4

Protocolo MIDI

MIDI (*Musical Instrument Digital Interface*) é um protocolo que já foi descrito por vários profissionais como sendo um esquema de interconexão entre instrumentos e computador, como uma linha de conexão para transferência de dados entre um instrumento e outro, ou como uma linguagem para transmissão de partituras musicais entre computadores e sintetizadores [1]. Este protocolo, na verdade, é composto por um esquema de interconexão de *hardware* e um método para comunicação de dados que é capaz de realizar controle em tempo real de instrumentos musicais. O controle exercido pelo MIDI sobre os instrumentos é realizado através de mensagens capazes de transmitir sinais de sincronização entre os diversos instrumentos, além de informações sobre uma nota musical específica a ser executada por um deles.

Todo dispositivo MIDI possui um microprocessador que interpreta e gera dados no formato MIDI. O dado interpretado por estes dispositivos advém de mensagens MIDI enviadas e recebidas por dispositivos que suportam o protocolo MIDI. As mensagens tanto podem ser informações de notas musicais (formando uma melodia) quanto informações de controle (mudança de instrumento, início de um evento de nota). O tratamento de timbre não faz parte do escopo do MIDI. As mensagens trocadas pelos dispositivos não estabelecem o timbre no qual a nota deve ser executada, mas sim em qual instrumento, ficando sobre responsabilidade do microprocessador interpretar esta mensagem e executá-la gerando o som do instrumento que possui seu timbre específico. O protocolo MIDI não padroniza os timbres dos instrumentos, ou seja, dependendo do fabricante o som de um piano pode ser um pouco diferente entre dois dispositivos MIDI.

4.1 Especificação MIDI 1.0

O controle de sintetizadores de som a partir de computadores (também conhecidos como sistemas híbridos) começou bem antes do advento da interface MIDI. Inicialmente, os computadores geravam uma palavra binária (*stream* de *bytes*) que continham diretivas para funções de controle. Os campos desta *stream* de *bytes* eram devidamente separados pelo *hardware* do sintetizador e, posteriormente, enviados a conversores analógico-digital (A/D) e, então, entregues aos sintetizadores, que seriam os responsáveis pela execução do sinal enviado.

Os primeiros sistemas híbridos desenvolvidos foram o GROOVE [24] e o HYBRID [25], que obtiveram relativo sucesso. Naquela época, cada sistema tinha seu próprio protocolo de controle e o *hardware* era customizado.

A Figura 15 mostra um exemplo de esquema de sistema híbrido. Os sinais digitais são gerados pelo computador que, posteriormente, os direciona para um dispositivo multiplexador. Este dispositivo será o responsável por rotear os sinais pelos vários canais existentes. O sinal ao chegar em um desses canais é transformado em forma de onda por um DAC[7] e, em seguida, repassado ao sintetizador. Dentro do sintetizador existem 3 módulos: o VCO (*voltage-controlled oscillator*), que ficará responsável pela modulação do sinal de forma a obter a forma de onda que foi enviada pelo computador; o módulo VCF *(voltage-controlled filter*) será o responsável por possíveis aplicações de filtros sobre o sinal gerado pelo VCO e; o módulo VCA (*voltage-controlled amplifier*) amplificará o sinal.

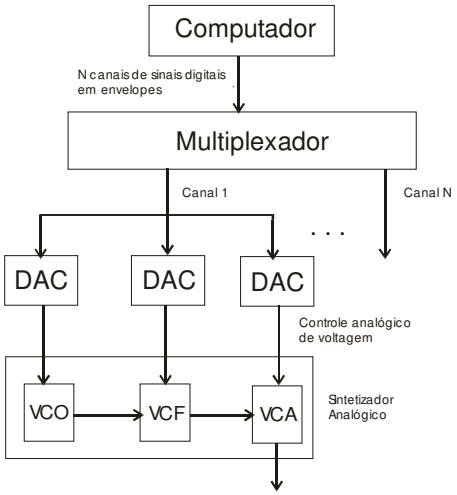

Figura 15. Esquema de controle híbrido.

No fim dos anos 70 os microprocessadores se tornaram capazes de controlar estes sistemas híbridos e, então, começaram a ser comercializados sistemas híbridos controlados por microprocessadores. Todavia, todos os sintetizadores desenvolvidos até então não possuíam uma interface comum de comunicação. Instrumentos que fossem desenvolvidos com tecnologia vigente não eram compatíveis com demais sintetizadores, o que impôs uma limitação muito forte ao crescimento da utilização dos sistemas híbridos.

Visando tornar os sistemas híbridos compatíveis, um consórcio de empresas americanas e japonesas decidiu trabalhar na especificação de um protocolo de comunicação comum entre os sintetizadores que ficou formalmente conhecido como MIDI (*Musical Intrument Digital Interface*). Em 1983 a especificação do MIDI 1.0 foi publicada e até hoje os sistemas híbridos desenvolvidos suportam e utilizam amplamente este protocolo. Obviamente, esse novo protocolo introduziu novas idéias e possibilidades como a composição de música, e a troca de melodias e arranjos entre músicos.

Dentre as possibilidades musicais que o MIDI oferece, podemos citar como sendo a mais importante a separação que existe entre o dispositivo de entrada e o sintetizador. Graças a esta característica peculiar, MIDI possibilita que apenas um instrumento, como o teclado, possa gerar sons de vários instrumentos diferentes e que vários instrumentos sejam compatíveis. Um microfone pode ser facilmente utilizado como um gerador de entrada para o MIDI se utilizarmos

[7] DAC – conversor analógico-digital

um conversor pitch-to-MIDI. As freqüências de entrada são convertidas em pitches e então repassadas ao dispositivo.

O fato de MIDI ter sido um protocolo de interface entre computador e instrumento o fez ser genérico o suficiente para gerar arquivos musicais independentes da plataforma em que irão ser executados. O desenvolvedor não precisa se preocupar em qual plataforma ou sintetizador seu arquivo será executado, pois todos suportam e entendem as mensagens MIDI da mesma forma. Obviamente existem algumas diferenças de timbre entre um sintetizador e outro, uma vez que empresas diferentes possuem seus próprios samples com o som dos instrumentos.

O que fez o protocolo MIDI se tornar interessante no desenvolvimento deste trabalho foi a possibilidade do desenvolvimento de softwares de performance interativa. Composição algorítmica, edição de partituras e seqüênciamento permitem que o usuário possa criar suas próprias melodias sem ter que se preocupar com manipulação de sinal. As informações, contendo nota, altura e tempo são repassadas ao dispositivo de entrada MIDI. O dispositivo torna-se responsável por gerar os sinais correspondentes e enviá-los ao sintetizador. O Solfeggiare se faz valer desta característica para gerar suas melodias a partir de partituras descritas em MusicXML [3]. O MusicXML é uma especificação em XML de um modelo hierárquico para representação de partitura que é utilizado e gerado por várias ferramentas que estão no mercado. Sua especificação possui suporte a MIDI com a inserção de elementos específicos no arquivo MusicXML para descrever parâmetros que podem ser utilizados na geração de um arquivo MIDI a partir de uma partitura.

4.1.1 Especificaçao do hardware MIDI

O *hardware* MIDI implementa um único protocolo para recepção e transmissão de mensagens MIDI. O tráfego dessas mensagens é realizado bit a bit como uma *stream* de *bytes* de forma assíncrona, ou seja, as mensagens são enviadas no momento em que são geradas. A composição do hardware de MIDI se divide em portas e interfaces com o computador.

Existem três tipos de portas MIDI, a IN, a THRU e a OUT, como exemplificado na Figura 16. Todas seguem o padrão alemão DIN, porém são especialmente adaptadas para a utilização no MIDI. A porta IN de entrada está conectada a um opto-isolator que recebe os sinais de entrada. O *opto-isolator* gera luz através de LED (*light-emitting diode*) em uma caixa opaca. O foto-sensor presente transforma a luz em corrente elétrica. Todo este procedimento é tomado para reduzir o ruído presente nas transmissões das mensagens MIDI.

Como a transmissão de mensagens é feita de forma assíncrona, um dispositivo especial é utilizado para realizar a interface entre as portas e os dispositivos MIDI. UART (*universal asynchronous receiver/transmitter*) recebe os bits que são transmitidos um-a-um, ignorando o primeiro e o último bit, formalmente conhecidos como start e stop bits. Após ter agregado todos os bits de uma mensagem, esse dispositivo a repassa para o dispositivo MIDI que irá tratar a informação presente na mesma.

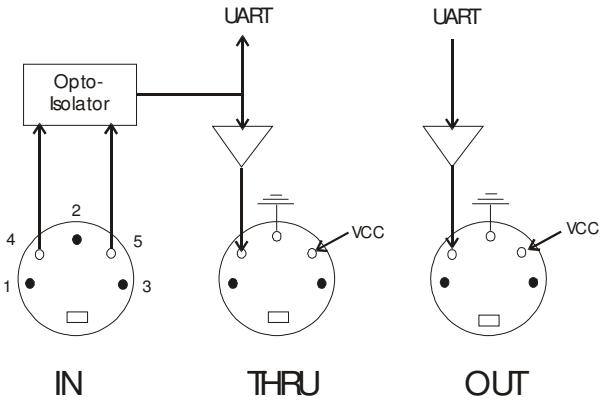

Figura 16. Esquema de portas MIDI.

As mensagens MIDI, Figura 17, são compostas por 10 *bits*. O primeiro e o último *bit* são conhecidos como *start* e *stop* e são utilizados para sinalizar os dispositivos antes e após o envio das mensagens. No padrão MIDI, o valor do *start bit* é '0' (zero) e o *stop bit* é '1' (um). Dos oito bits restantes, que representam a mensagem MIDI, o mais significativo é um *bit* de *status* para informar se a mensagem é um comando ou é um argumento para um comando.

A interpretação das mensagens MIDI que são trocadas pelos sintetizadores ou processadores de sinal é realizada por um *driver*[8] especial, que tanto pode ser um microprocessador independente ou ser parte do sistema operacional da máquina.

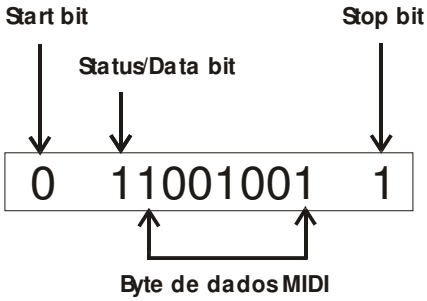

Figura 17. Esquema de uma típica mensagem MIDI.

As interfaces com o computador de MIDI são utilizadas quando o computador que será utilizado não possui portas MIDI acopladas. Existem três tipos de interfaces MIDI: serial, paralela e *multiline*.

A interface serial funciona da mesma forma que o protocolo de troca de mensagens MIDI. Os *bits* são enviados um a um para a interface MIDI que fica responsável por repassar a mensagem aos outros dispositivos.

A interface paralela envia todos os 10 *bits* de uma mensagem para a interface MIDI. Por possuir esta característica, as mensagens MIDI trafegam com alta velocidade, deixando o processador livre para executar outras tarefas. Porém, os demais dispositivos MIDI continuam recebendo as mensagens de forma serial. Não haverá maior velocidade de transmissão das mensagens em todo o sistema.

Já a interface *multiline*, também chamada *multiport*, conecta o computador a diversas linhas MIDI independentes. Cada linha pode ser abstraída como uma rede de dezesseis canais

[8] *Driver - hardware* especializado utilizado para controlar dispositivos eletrônicos.

MIDI. Este tipo de interface possibilita ao usuário utilizar mais do que as dezesseis portas MIDI que são definidas pelo protocolo.

4.2 Canais MIDI

O protocolo de MIDI especifica dezesseis diferentes canais para transmissão de streams de mensagens diferentes. Diferentemente dos tocadores de fita, esses canais não são físicos, mas sim endereços eletrônicos diferentes que podem ser assinalados a diferentes dispositivos cujas mensagens trafegam sobre um mesmo barramento MIDI.

Em um típico programa MIDI, um harpejo de violão pode ser passado por um determinado canal enquanto que o acompanhamento percussivo pode ser enviado em outro canal. Os dispositivos MIDI que estiverem habilitados em cada um desses canais irão interpretar as mensagens e executá-las de forma que se ouvirá o violão e a percussão ao mesmo tempo, em harmonia.

Apesar de possuir um número fixo de canais, um programa MIDI pode utilizar quantos canais forem necessários, contanto que possua um esquema multiline de interface.

4.3 Mensagens MIDI

As mensagens MIDI foram estabelecidas pelo protocolo MIDI e são compostas por palavras de 10 *bits*. Cada palavra possui seus respectivos *bits start* e *stop* e mais 8 *bits* com a mensagem MIDI específica. Mais de uma palavra pode fazer parte de uma mensagem, de acordo com a complexidade do comando a ser interpretado pelos sintetizadores. A Tabela 1 especifica as principais mensagens MIDI.

As mensagens MIDI são compostas por *bytes* de *status* e de dados, sendo que as de *status* geralmente se iniciam com o *bit* 1, já as de dados se iniciam com o *bit* 0. Em uma típica mensagem de "note-on" (vide Tabela 6), três *bytes* são enviados, sendo o primeiro *byte* só para indicar o status da mensagem. Os quatro primeiros *bits* da mensagem de *status* especificam qual a função MIDI que está sendo chamada, enquanto que os quatro últimos indicam para qual canal MIDI a mensagem se destina. Mensagens de dados são utilizadas na execução de uma função. Por exemplo, em uma típica mensagem de "note-on", a mensagem de dados MIDI carrega as informações de *pitch* e *velocity*[9] de cada nota.

[9] Velocidade em que a nota foi pressionada.

Tabela 6. Mensagens MIDI.

Mensagem MIDI	Ação
Note-on	Informações da nota a ser executada pelo sintetizador.
Note-off	Indica ao sintetizador que a nota não está mais sendo executada ou pressionada (teclado).
Polyphonic Key Pressure	Indica qual foi a nota pressionada, o canal e a pressão exercida pelo instrumentista.
Channel Pressure	Indica a pressão "média" de todas as notas executadas em um determinado canal MIDI.
Control Change	Informa a um dispositivo específico a intensidade exercida sobre um pedal ou mesmo se ele está sendo pressionado ou não.
Pitch Bend	Informa a distorção de altura (pitch) sobre uma determinada nota.
Program Change	Mensagem que contém um byte para escolha de canal e outro para escolha de efeito. Indica ao dispositivo MIDI que ele deve, por exemplo, mudar de um instrumento para outro durante a execução da melodia. Também indica certos tipos de efeito sobre um canal como reverberação.
Bank Select	Mesma funcionalidade da mensagem "Program Change". Porém mais utilizada quando o dispositivo possui mais de 128 programas.
Local/Remote Keyboard Control	Desconecta um teclado de seu dispositivo sintetizador. Permite que o sintetizador seja controlado por outro dispositivo externo ao sistema enquanto o teclado continua enviando mensagens de dados.
All Notes Off	Mensagem de emergência que pára todas as notas que estão sendo executadas.
Reset All Controllers	Retorna todos os controladores para o seu estado inicial.
ModeSelect	Seleciona o modo MIDI a ser utilizado.
Song Position Pointer	Mede a quantidade de pulsos de clock que foram executados desde o começo de uma música.
Song Select	Seleciona uma das músicas-padrão disponíveis no dispositivo.
Tune Request	Inicia rotinas em um sintetizador analógico para sincronizar os osciladores.
End System Exclusive	Finaliza uma mensagem exclusiva do sistema previamente enviada.
MIDI clock	Usado como pulso de tempo. É transmitida 24 vezes a cada semínima.
Start	Gerada quando um seqüenciador ou uma máquina de percussão é iniciada.
Stop	Interrompe a execução do seqüenciador ou máquina de percussão.
Continue	Continua a execução do seqüenciador ou máquina de percussão.
Active Sensing	Evita notas "presas" caso haja interrupção da transmissão dos dados MIDI.
System Reset	Reinicia um dispositivo.
System Exclusive	Utilizada para enviar mensagens exclusivas de um dispositivo.

4.4 Representação MIDI de Pitch

Em português, *pitch* pode ser traduzido como altura de notas musicais em oitavas. Uma mensagem de "*note-on*" em MIDI possui 7 *bits*. Como se utiliza a codificação binária, um total de 128 *pitches* são suportados por MIDI. O intervalo infra-sônico de MIDI varia entre 0 e 12 (ou C0[10] e C1) entre as freqüências de 8.17Hz e 16.35Hz. A chave 60 é utilizada para definir o C4 ou dó natural (em torno de 261.63Hz). Alguns definem o C5 como sendo o dó natural ou até mesmo o C3. Os sintetizadores dispõem de um mecanismo de flexibilização que permite que o usuário defina qual a freqüência que será atribuída a cada nota.

4.5 Mensagens de canal e de sistema

As mensagens MIDI ainda podem ser divididas em duas categorias: mensagens de canal; e mensagens de sistema. As mensagens de canal são aquelas exclusivas de um canal MIDI. As mensagens de sistema são enviadas a todos os dispositivos e a todos os canais por se tratarem de mensagens de controle.

As mensagens de canal de voz são as mais usuais e conhecidas, uma vez que elas carregam os dados de cada nota a ser executada. Estão incluídas aqui também as mensagens que sinalizam alterações nas notas como a presença de pedais e alterações chamadas *bend*[11].

As mensagens de sistema podem ser utilizadas como mecanismo de sincronização ou como transportadoras de mensagens exclusivas do dispositivo que está sendo utilizado. As mensagens de sincronização são enviadas em intervalos regulares, juntamente com um *clock* que será o responsável pelo alinhamento de todos os dispositivos. Já as mensagens exclusivas devem ser definidas pelos fabricantes dos dispositivos. Por sua vez, os dispositivos devem ser responsáveis pela interpretação e execução das mesmas.

4.6 Modos MIDI

Em MIDI existem 5 modos diferentes:

1. Omni-on Poly
2. Omni-on Mono
3. Omni-off Poly
4. Omni-off
5. General MIDI

[10] C0 – dó na oitava de número 0 (zero). Esta notação será muito utilizada daqui pra frente. A letra especifica o tipo da nota, no caso "dó", e o número especifica em qual oitava está o "dó".

[11] Distorção de altura de uma nota.

O primeiro modo MIDI, mais conhecido como “Omni”, define que um instrumento deverá escutar (receber mensagens) todos os canais MIDI, mas apenas responderá em um. Este modo é mais utilizado para realização de testes de interconexão entre dispositivos.

O modo “Omni-on Mono” define que o dispositivo receberá mensagens de todos os canais, porém só poderá responder em um único canal, tocando uma nota de cada vez. Este modo entrou em desuso.

“Poly” é como é chamado o terceiro modo MIDI. Recomendado para ambientes com múltiplos instrumentos. Cada canal é assinalado para um instrumento diferente e as notas são executadas de forma polifônica.

O quarto modo, ou modo “Multi”, é utilizado geralmente com instrumentos MIDI que são multi-timbre. Como este instrumento é capaz de executar múltiplas vozes, canais MIDI consecutivos são assinalados para ele e este dispositivo será responsável por executar todas essas vozes. Um exemplo deste cenário é a guitarra MIDI. Cada corda da guitarra é assinalada a um canal MIDI e o sintetizador possui um esquema de timbre para cada corda, possibilitando maior realismo na execução do som.

O último modo é mais conhecido como GMM ou General MIDI Mode. A idéia é poupar o usuário de ter que configurar manualmente o sintetizador/instrumento para executar em um modo específico, pois o mapeamento é feito de maneira automática. Os dez primeiros canais são pré-assinalados onde o canal quatro geralmente fica com a melodia; o oito para harmonia; e o dez para percussão. Os demais canais são livres para demais melodias ou harmonias descritas em instrumentos diferentes.

4.7 Standard MIDI files

Apesar da especificação original do MIDI padronizar a linguagem para controle musical, ela não descreveu nenhum padrão para a formatação dos arquivos de dados que seriam interpretados pelos dispositivos MIDI. O cenário que se tinha era de vários arquivos com dados MIDI em formatos diferentes que só eram interpretados pelos dispositivos que os tinham gerado. Para resolver este impasse a comunidade MIDI estabeleceu em 1988 a especificação chamada *Standard MIDI files* (SMF) [1].

Os dispositivos continuam com seu funcionamento interno específico porém a padronização dos arquivos MIDI tornou tais dispositivos mais portáveis e possibilitou a troca de informações musicais de maneira mais fácil.

A principal diferença entre os *Standard MIDI files* e os arquivos que tinham apenas o dado bruto MIDI é que os dados MIDI nos SMFs possuem etiquetas de tempo. Cada mensagem vem com um rótulo que pode ser de 8 ou 32 *bits*, especificando em que momento do pulso de *clock* ela deverá ser executada. Além disso os SMF especificam também três tipos de trilhas para os arquivos: Tipo 0 (zero); tipo 1 e tipo 2.

O tipo 0 (zero) possui apenas uma trilha com dados de vários canais diferentes. O tipo 1 é para arquivos *multitrack* utilizados em seqüenciadores, e o tipo 2 carrega dados para programas baseados em padrões. Neste último tipo os dados são organizados em coleções de seqüências do tipo 0 (zero).

Nos arquivos do tipo 0 etiquetas de tempo ficam misturadas com os dados. Nos arquivos do tipo 1 todas as trilhas diferentes possuem a mesma etiqueta de tempo, ou seja, todos serão executados simultaneamente, enquanto no tipo 2 cada trilha possui sua marca de tempo

independente e cada trilha executa em seu tempo específico. Esta última abordagem também é conhecida por *drum machine format*.

4.8 Limitações de MIDI

A especificação de MIDI, apesar de ser simples e barata e possui todas as vantagens já listadas acima, contém alguns pontos fracos que precisam ser abordados e devem ser conhecidos por todos os profissionais que desejam utilizá-lo. As limitações de MIDI podem ser divididas em três categorias: limitações de largura de banda, limitações de roteamento de rede e limitações de representação musical [26].

A limitação de largura de banda se dá devido à quantidade de dados de controle que são necessários durante a execução de uma música. A utilização de vibrato em apenas uma voz pode consumir praticamente toda a largura de banda MIDI. Algumas formações de acorde também podem parecer harpejos ao invés de um som harmônico, mais especificamente quando o tempo das notas que formam o acorde é muito pequeno. Isto se deve à característica seqüencial de MIDI, em que as notas são enviadas em mensagens diferentes, uma de cada vez. Alguns defendem que este tipo de limitação introduz certa "vida" às peças interpretadas pelo MIDI, porém a introdução desses tipos de atraso deveria ser parametrizada e não execuções ao acaso.

Quanto à questão do roteamento das mensagens MIDI, o cenário que se tem hoje é que cada via de comunicação MIDI exige seu próprio canal, ou seja, a comunicação em um único cabo não é *full-duplex*. Para realizar a comunicação entre dois dispositivos MIDI em duas vias, é necessária a presença de dois cabos para conectar os dispositivos. Cada cabo implementa a comunicação e seu sentido. Portanto, o número de cabos pode ser muito grande, caso mais de dois dispositivos sejam utilizados. A quantidade de cabos é duas vezes igual à quantidade de dispositivos e em um ambiente como um estúdio isso pode se tornar uma grande barreira.

As limitações de representação musical estão mais relacionadas à questão do timbre. As mensagens MIDI não especificam nada em relação ao timbre que a nota que está sendo enviada terá. O resultado é que uma mesma nota pode soar como um baixo acústico em um sintetizador ou como um piano em outro. Isto se deve ao fato de MIDI ser um protocolo de representação de música que procura ser independente do dispositivo que irá utilizá-lo. Da mesma forma, a representação de *pitch* de MIDI é fraca. Apesar de possibilitar a utilização de *bends* em notas, o MIDI aplica a mensagem de *bend* a todas as notas do canal MIDI para onde a mensagem foi enviada, tirando um pouco da flexibilidade do protocolo.

4.9 MIDI e multimídia

Através dos anos o protocolo MIDI, por sua alta flexibilidade, começou a ser utilizado por cada vez mais dispositivos geradores de áudio. Tanto como uma maneira de se trocar música, como ferramenta para composição de arranjos e melodias. Recentemente, a indústria dos celulares tomou conhecimento das possibilidades do MIDI. Desde então, todos os celulares começaram a

vir de fábrica com um pequeno sintetizador MIDI no sistema operacional do aparelho. Foi desta forma que se popularizou o conhecido *ringtone*.

Arquivos MIDI que estejam de acordo com os padrões SMF podem ser executados em quase todos os celulares que suportam melodias polifônicas. Obviamente, existem algumas limitações performáticas e de *hardware*. Porém, elas não se mostraram barreiras para o desenvolvimento de música para estes dispositivos.

Existe uma especificação de MIDI somente para dispositivos móveis, chamada *GM-Lite*, que pode ser encontrada em [2].

Atualmente, a evolução dos toques MIDI em dispositivos móveis pode ser observada nos programas desenvolvidos pelos fabricantes dos dispositivos que possibilitam o usuário a editar e criar seus próprios *ringtones*. Esses "mixadores" permitem que o usuário, a partir de uma melodia base, possa criar seu próprio toque habilitando ou desabilitando certos canais MIDI em alguns momentos.

Além da imensa gama de possibilidades, os arquivos MIDI também são utilizados por profissionais que desejem gravar seus acompanhamentos antes da gravação em estúdio. Geralmente, utiliza-se um teclado que grave arquivos MIDI. O acompanhamento é gravado em algum dispositivo de armazenamento e, posteriormente, passado para a mesa controladora do estúdio, que utilizará o mesmo quando for necessário inserir a voz na gravação.

Capítulo 5

Solfeggiare: compilando uma partitura

Como explanado no capítulo 2, as *Score Input Languages* foram inicialmente concebidas com o propósito de execução musical ou da simples construção gráfica de partituras musicais. O SMDL (*Standard Music Description Language*) é uma linguagem que para descrição musical foi considerada como um dos maiores esforços na tentativa de realizar uma análise musical em cima de uma partitura [27]. Uma série de outras linguagens foram desenvolvidas nessa mesma época, como a SCORE [28], SCRIPT [29] e ASHTON [30]. Todas elas possuíam seu próprio vocabulário de comandos e uma maneira particular de transformar os dados de entrada em música ou em partitura gráfica.

O compilador desenvolvido neste trabalho, chamado *Solfeggiare*, pode ser visto como uma *Score Input Language*, uma vez que recebe como entrada uma partitura no formato MusicXML e gera um arquivo nos padrões MIDI com a música descrita pela mesma.

Todas as linguagens desenvolvidas anteriormente com este propósito possuíam suas próprias especificações e padrões, tornando a tarefa de composição muito restrita a uma determinada arquitetura. O *Solfeggiare* utiliza o padrão XML para uniformizar a maneira como a partitura é descrita. Estando o arquivo de entrada do compilador de acordo com os padrões MusicXML, o compilador poderá transformar esses meta-dados em música.

Por possuir esta característica, o *Solfeggiare* é integrável com outras aplicações, rompe a barreira do desuso, como previsto por Milton Babbit em 1965 [1] para as *Score Input Languages*, e se moderniza. Ferramentas OCR[12] que seriam o futuro das *Score Input Languages* se tornariam interfaces entre a partitura gráfica e as linguagens de composição, isolando responsabilidades e modularizando o desenvolvimento.

[12] *Optical character recognition* – reconhecimento óptico de caracteres. Aplicações que, a partir de uma imagem, geram um arquivo texto com a transcrição do texto presente na mesma.

5.1 Solfeggiare – O compilador de Partituras

O processo de compilação definido pelo *Solfeggiare* ocorre de maneira mais simples do que quando comparado com compiladores de linguagens de programação como Java e C. O *front-end* e o *back-end* do *Solfeggiare* são bem mais simples e estão estabelecidos em cima de modelos e padrões bem aceitos pela comunidade científica.

O formato da entrada deste compilador deve estar de acordo com o padrão definido pelo MusicXML [3], anteriormente explanado. Apenas as partituras descritas pelo modelo *score-partwise* são suportadas pelo *Solfeggiare*. O processo de análise inicial do arquivo de entrada a partir da navegação de um documento MusicXML denota o *front-end* do compilador. Após a geração de objetos em memória que possuam os dados do arquivo XML, o *Solfeggiare* inicia a validação da semântica dos mesmos, criando padrões em formato texto que serão posteriormente utilizados para a geração do arquivo MIDI com a melodia. Esta segunda etapa caracteriza o *back-end* do compilador e é, em sua totalidade, suportado pela API JFugue [5] de geração de arquivos MIDI. A Figura 18 representa o diagrama de classes em UML [22] do *Solfeggiare*.

Interfaces foram definidas para os processos de leitura e geração do arquivo MIDI para tornar o compilador extensível e permitir que experimentos com outras ferramentas que melhorem o processo de compilação.

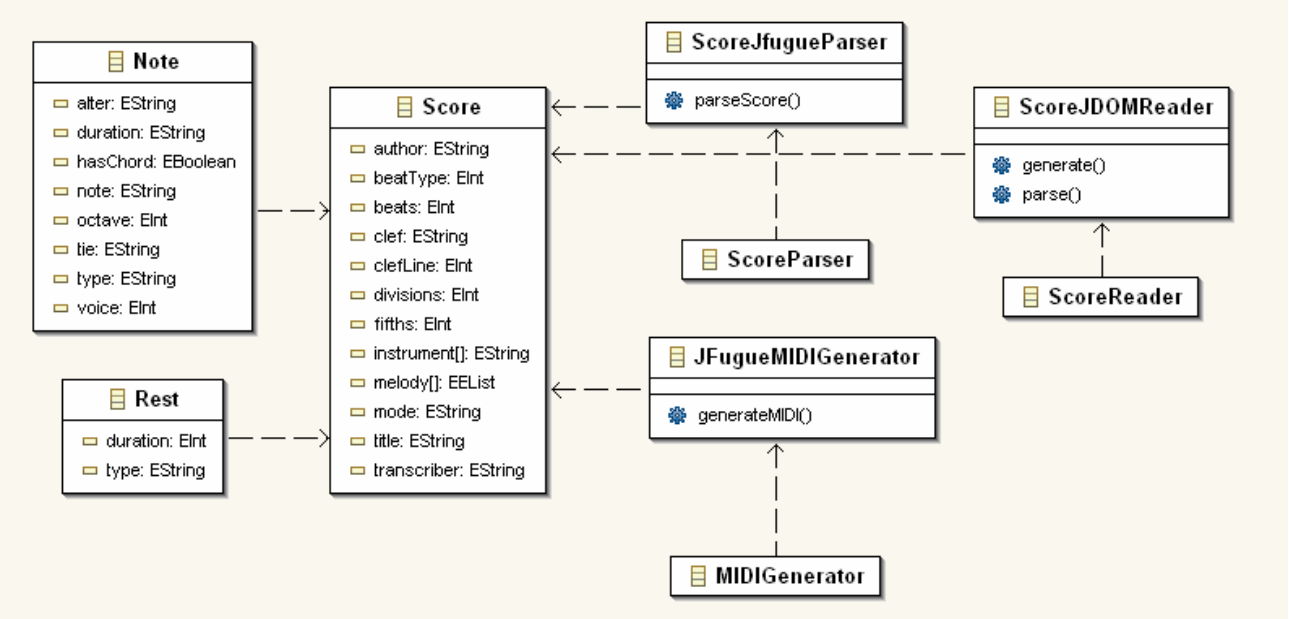

Figura 18. Classes de negócio e de entidade do *Solfeggiare*.

5.2 Front-end

Como já explanado, o *front-end* do *Solfeggiare* está mais voltado para a análise e a obtenção dos dados de um arquivo de entrada no formato MusicXML. A descrição do formato base de arquivo MusicXML que será lido pelo compilador, foi apresentada no Capítulo 3.

Inicialmente as informações gerais da partitura são extraídas pelo compilador. Nome da peça, compositor, tradutor da partitura e as partes em que está dividida a peça representam as informações iniciais. Mais de uma parte na partitura significa que há mais de uma voz, ou seja, mais de uma linha melódica ou uma linha harmônica extra. Posteriormente, dados sobre os instrumentos de cada voz são obtidos. Caso não haja nenhuma definição de instrumento para a voz, o padrão será "piano".

De posse dos identificadores da partitura, o *Solfeggiare* inicia a recuperação da informação relativa ao conteúdo de cada porção em que a partitura está dividida. Cada porção está dividida em compassos. O primeiro compasso de cada porção possui os dados relativos à clave, divisão rítmica do compasso, tonalidade da melodia e etc. A clave, apesar de ser muito importante em uma partitura clássica, não tem tanta importância para o compilador em questão, pois a altura de cada nota é denotada pela oitava que é explicitada ao lado de cada nota. As informações sobre divisão rítmica são fundamentais, principalmente para a organização das notas em compassos. A tonalidade da melodia fornece poder de abstração ao usuário sobre quais acidentes deverão ser inseridos nas notas de acordo com a tonalidade em que ela está inserida.

Com o contexto de execução de uma peça totalmente formado, o *front-end* do *Solfeggiare* passa para o processo de obtenção das notas e pausas do modelo. Em cada compasso, as notas ou pausas são extraídas e analisadas. Todos os atributos para a classe "Note" e a classe "Rest", vide Figura 1, devem ser definidos neste momento. O atributo "alter", que denota a presença de um acidente na nota, só será utilizado quando a nota tiver um acidente explicitamente inserido na partitura. Caso o acidente em uma nota tenha sido inserido devido à tonalidade em que ela se encontra, o *Solfeggiare* irá abstraí-lo.

A cada passo da análise dos compassos, uma estrutura de dados indexada pelo número da melodia é preenchida com os elementos correspondentes. Esta mesma estrutura será posteriormente utilizada pelo *back-end* do *Solfeggiare* para gerar a melodia. A Figura 19 exemplifica o diagrama de seqüência do processo do *front-end*. A classe "*SolfeFrame*" é uma classe de fronteira responsável pela interação com o usuário.

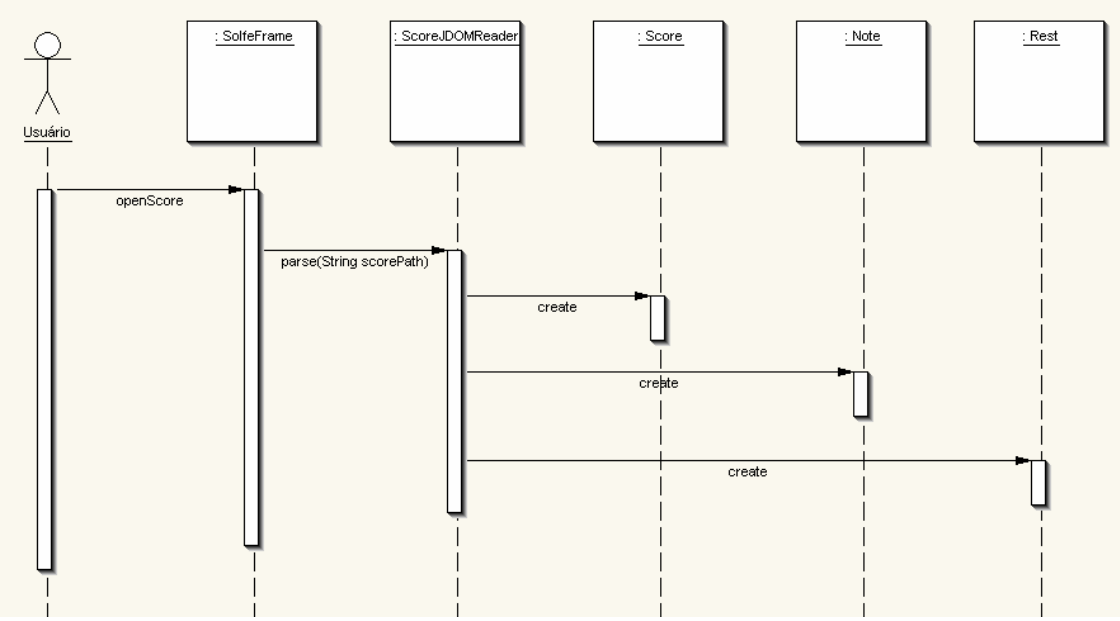

Figura 19. Diagrama de seqüência do processo do *front-end*.

5.3 Back-end

Após o fim do processo de análise e validação do arquivo de entrada, o *Solfeggiare* procede com a montagem do padrão que será responsável pela criação do arquivo MIDI. Durante o processo de montagem do padrão assunções são feitas de acordo com algumas características da partitura. Um exemplo está na análise da tonalidade da música. Algumas tonalidades possuem notas com acidentes devido a distribuição de tons e semitons para modos maiores e menores. A Tabela 1 exemplifica as tonalidades maiores e as suas respectivas notas. Em cada objeto do tipo "*Score*" há a indicação de qual é a quinta (atributo "*fifths*" da classe) em que a partitura foi descrita. A definição deste parâmetro se dá de acordo com o ciclo das quintas[13], mecanismo para definição de tonalidade.

Cada tonalidade possui seu conjunto de notas e algumas podem possuir notas com acidentes. A Tabela 7 exemplifica para cada tom maior, o padrão de notas que será utilizado. Esta implementação inicial do *Solfeggiare* não possui um suporte específico para acidentes em tons menores. Neste caso, todos os acidentes presentes em uma peça descrita em um tom menor deverão ser explicitados no arquivo MusicXML,

De posse da tonalidade da melodia, o *back-end* do *Solfeggiare* inicia o processo de criação do padrão da melodia de acordo com a API JFugue [5], definido no Capítulo 3. Cada melodia presente na partitura é analisada de cada vez. Os compassos são divididos e as notas e pausas presentes em cada um transformadas de acordo com o padrão da API de geração do arquivo MIDI. Alguns elementos específicos para a geração do arquivo MIDI também são recuperados neste momento e transformados em mensagens MIDI que indicarão, por exemplo, qual o instrumento a ser utilizado para a execução da melodia.

[13] Mecanismo utilizado para definição de acidentes em tons maiores e menores. Intervalos de quintas ascendentes e descendentes são utilizados para esta definição, tendo como partida o Dó maior.

Tabela 7. Conjunto de notas por tonalidade.

Tom	Padrão de Notas
Dó Maior	C, D, E, F, G, A, B
Sol Maior	C, D, E, F#, G, A, B
Ré Maior	C#, D, E, F#, G, A, B
Lá Maior	C#, D, E, F#, G#, A, B
Mi Maior	C#, D#, E, F#, G#, A, B
Si Maior	C#, D#, E, F#, G#, A#, B
Fá# Maior	C#, D#, E#, F#, G#, A#, B
Dó# Maior	C#, D#, E#, F#, G#, A#, B#
Fá Maior	C, D, E, F, G, A, B*b*
Si*b* Maior	C, D, E*b*, F, G, A, B*b*
Mi*b* Maior	C, D, E*b*, F, G, A*b*, B*b*
Lá*b* Maior	C, D*b*, E*b*, F, G, A*b*, B*b*
Ré*b* Maior	C, D*b*, E*b*, F, G*b*, A*b*, B*b*
Sol*b* Maior	C*b*, D*b*, E*b*, F, G*b*, A*b*, B*b*
Dó*b* Maior	C*b*, D*b*, E*b*, F*b*, G*b*, A*b*, B*b*

Todos os elementos que podem alterar a execução de uma nota são verificados durante a fase de análise de uma nota. Acidentes, ligaduras e formações de acorde são suportados pelo *Solfeggiare*. Para as pausas há um comportamento particular. Alguns tradutores utilizam a mesma notação de tempo de notas para pausas, alguns utilizam uma notação com números inteiros baseado no elemento "*divisions*" presente no MusicXML. Este elemento indica quantos pulsos possui uma nota semínima na partitura descrita. O tempo de execução da pausa é sempre um número divisível pelo inteiro presente neste elemento particular.

Nota a nota, o compilador realiza esta análise e, ao término do processo, o padrão que foi estabelecido é repassado ao módulo responsável pela geração do arquivo MIDI. A Figura 20 exemplifica o diagrama de seqüência do processo do *back-end*.

A Figura 22 exemplifica o fluxograma do algoritmo do *back-end*. Devido ao grande número de passos da execução do mesmo, a fase de análise de notas e de pausas foi denotada como sub-rotina. A análise das notas está descrita no fluxograma da Figura 21.

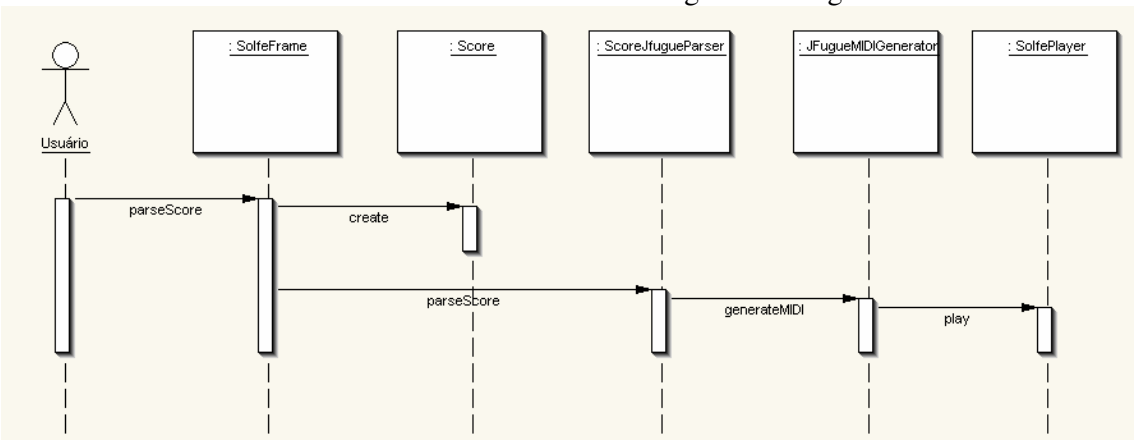

Figura 20. Diagrama de seqüência do processo do *back-end*.

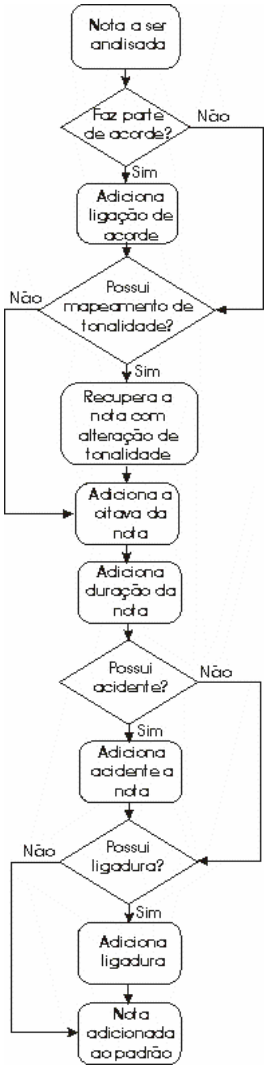

Figura 21. Fluxograma da análise de nota.

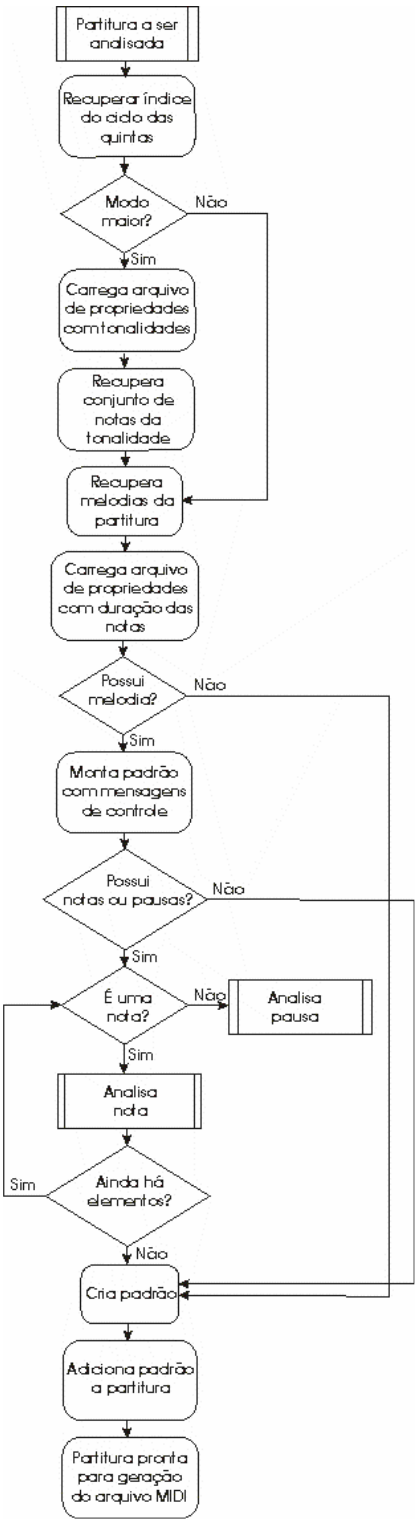

Figura 22. Fluxograma do algoritmo do *back-end*.

5.4 Estudo de Caso: Este Seu Olhar

Para exemplificar o funcionamento do *Solfeggiare*, uma peça musical foi escolhida como estudo de caso. “Este seu Olhar” é uma Bossa Nova em Fá maior que foi composta por Tom Jobim. Neste trabalho é apresentada apenas uma parte melódica da mesma. A formação melódica é relativamente simples e possui alguns dos elementos alteradores de nota que o *Solfeggiare* suporta. A Figura 23 apresenta os compassos iniciais da partitura de “Este Seu Olhar”.

Como pode-se observar, o primeiro compasso da partitura apresenta algumas informações importantes. A clave de Sol especifica que todas as notas que estiverem na segunda linha, de baixo para cima, são notas Sol. O símbolo do bemol (*b*) que aparece logo após a clave indica que a melodia está descrita em Fá maior. Todas as notas Si que aparecerem na partitura devem ser executadas com um bemol. Logo após o bemol, vemos uma letra C cortada. Este símbolo denota que os compassos dessa partitura possuem o escopo de tempo de duas notas mínimas, denotando um andamento 2 por 2. A primeira nota da partitura se encontra no segundo espaço, logo acima da segundo linha, denotando uma nota Lá. Esta nota terá a duração de uma semínima.

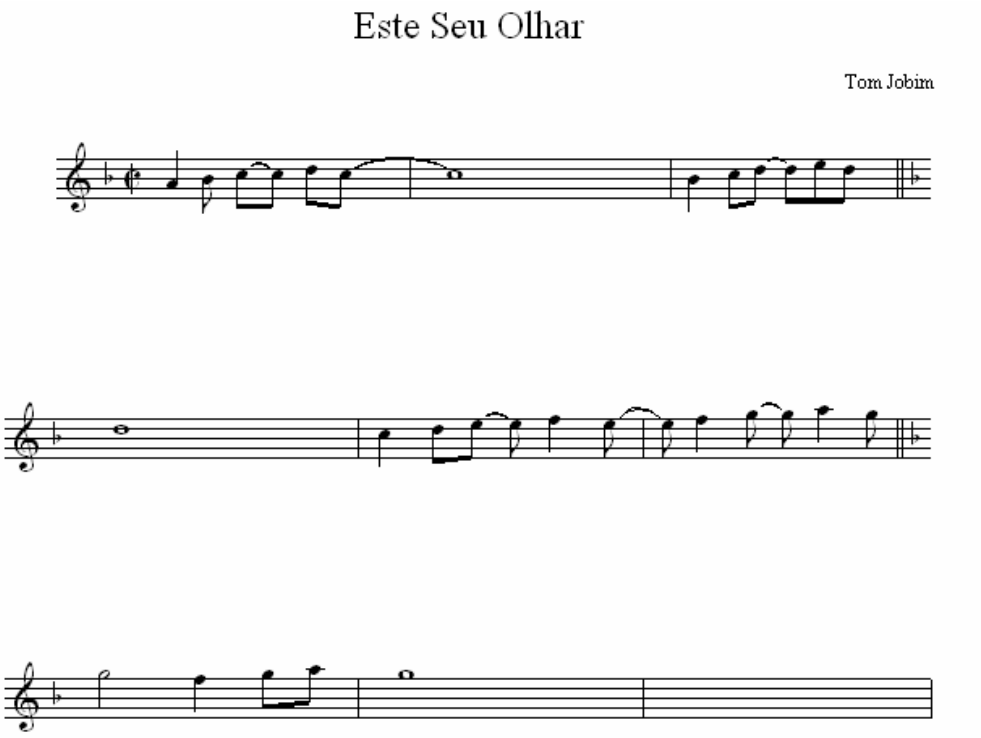

Figura 23. “Este Seu Olhar” – Tom Jobim.

Primeiramente, o arquivo MusicXML referente a esta partitura deve ser criado pelo tradutor, lembrando sempre de validá-lo com os documentos DTD de validação de arquivos XML. Esse trabalho tanto pode ser feito pelo usuário como por alguma ferramenta OCR de leitura de partituras. A Figura 24 mostra o arquivo XML referente à partitura exemplificada na Figura 23.

A partitura foi construída no modo *score-partwise*, onde as diferentes partituras presentes em uma peça são separadas em partes diferentes. O nome da peça, do autor e do tradutor estão contidas nos elementos *movement-title* e *identification*. O elemento *part-list* possui a descrição de

todas as partes presentes neste documento MusicXML. Para este caso só temos uma parte e ela é identificada como "Piano". Esta parte deverá ser executada pelo instrumento piano, como especificado pelo elemento *instrument-name*. Após a descrição de cada parte presente na partitura, encontra-se a lista de compassos de cada parte distinta (elemento *part*).

O primeiro compasso de cada parte possui informações gerais da partitura como a clave em que ela está descrita, o andamento e etc... O elemento correspondente ao compasso é o elemento *measure*. Também estão incluídos neste compasso as notas e pausas pertencentes a ele. Os compassos seguintes somente possuem informações de notas, pausas e elementos performáticos.

Após a leitura do arquivo, o *Solfeggiare* analisa todas as notas, identifica ligaduras, tonalidade e acidentes e procede com a criação do padrão da mesma. De posse do padrão, um arquivo MIDI é gerado. O nome do arquivo MIDI será sempre da seguinte forma: "<nome do compositor>-<nome da peça>.midi". Caso alguns dos parâmetros para nomeação do arquivo não sejam encontrados, a *string* "*Unknown*" será utilizada.

Terminado o processo de *parsing*, o arquivo MIDI com a melodia é gerado no sistema de arquivos do *Solfeggiare*. Para que o usuário possa escutar a melodia, um tocador de arquivos MIDI é apresentado pela aplicação e a melodia executada. A Figura 25 mostra o *Solfeggiare* em execução e a Figura 26 o momento de execução da melodia.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 1.0 Partwise//EN"
<score-partwise>
  <movement-title>Este Seu Olhar</movement-title>
  <identification>
    <creator type="composer">Tom Jobim</creator>
    <creator type="transcriber">Marcelo Nunes</creator>
  </identification>
  <part-list>
    <score-part id="P1">
      <part-name>Piano</part-name>
      <score-instrument id="P1-I3">
        <instrument-name>Piano</instrument-name>
      </score-instrument>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>24</divisions>
        <key>
          <fifths>-1</fifths>
          <mode>major</mode>
        </key>
        <time>
          <beats>2</beats>
          <beat-type>2</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>A</step>
          <octave>4</octave>
        </pitch>
        <type>quarter</type>
      </note>
      <note>
        <pitch>
          <step>B</step>
          <octave>4</octave>
        </pitch>
        <type>eighth</type>
      </note>
      <note>
        <pitch>
          <step>C</step>
          <octave>5</octave>
        </pitch>
        <type>eighth</type>
        <tie type="start"/>
      </note>
```

Figura 24. "Este Seu Olhar" transcrito para MusicXML.

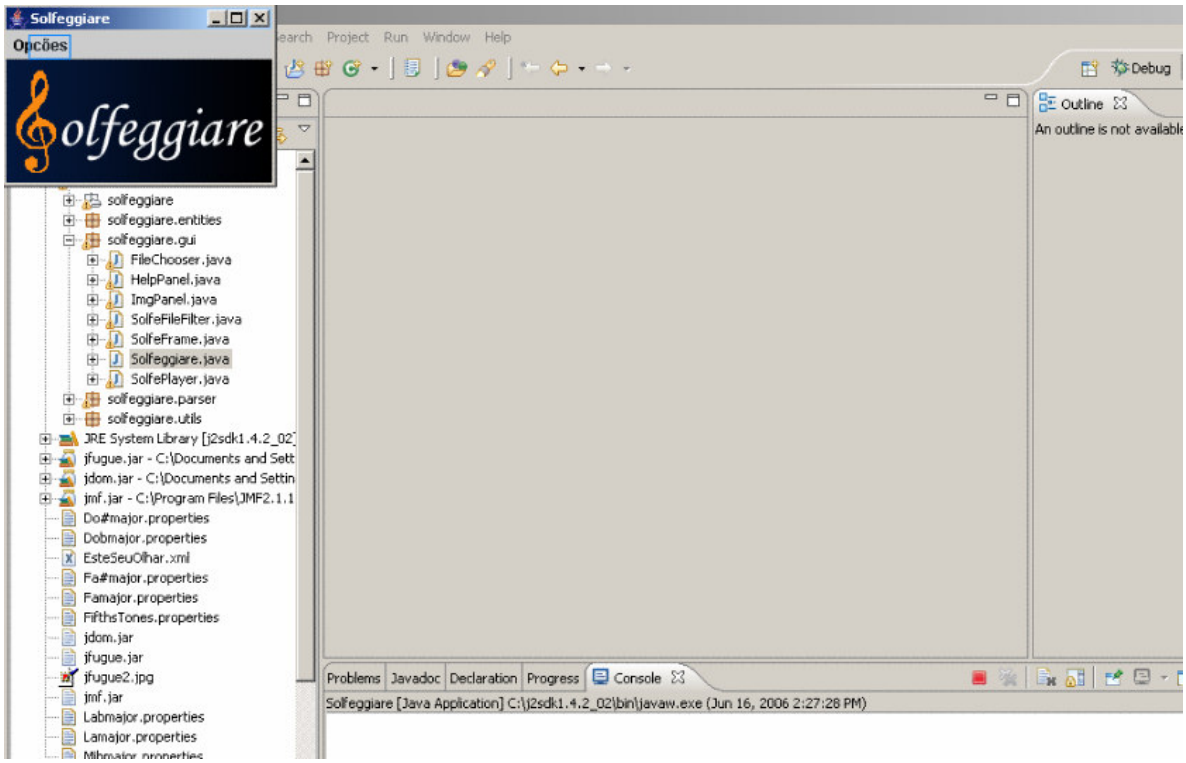

Figura 25. *Solfeggiare* em execução.

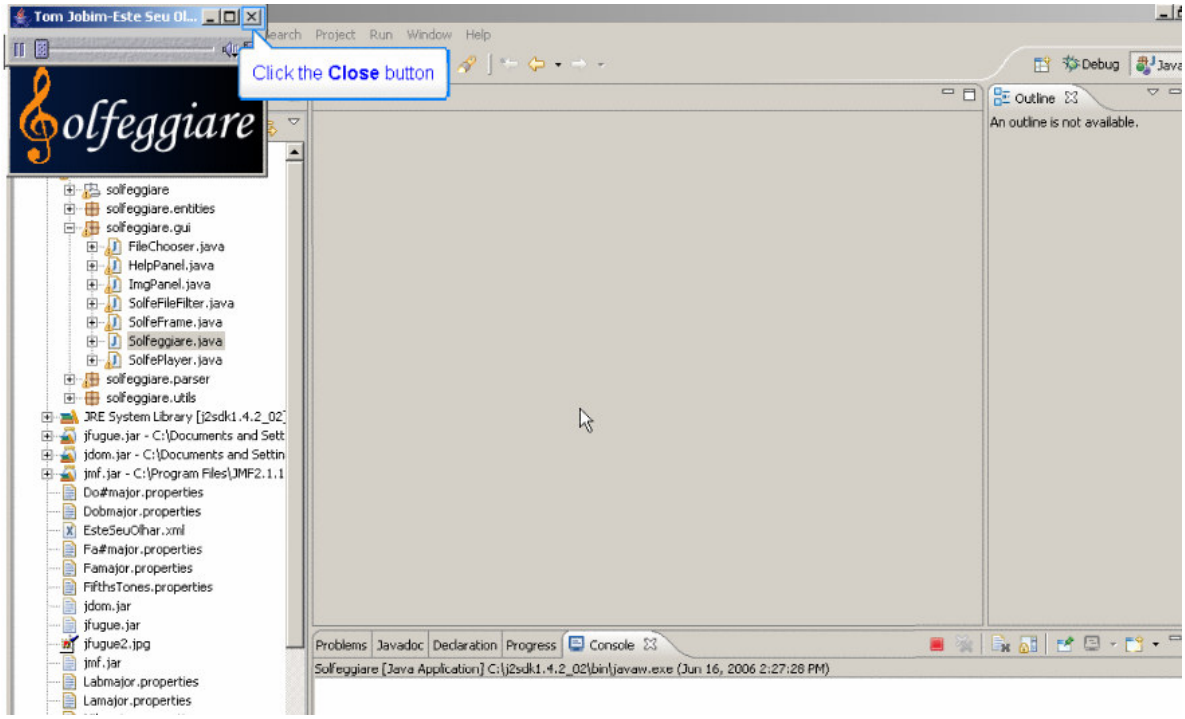

Figura 26. Momento de execução da melodia.

Capítulo 6

Conclusões e Trabalhos Futuros

Um estudo em linhas gerais sobre teoria musical e teoria dos compiladores foi elaborado neste trabalho, visando contextualizar leitores não especialistas no assunto, bem como familiarizá-los com a ferramenta Solfeggiare, desenvolvida como parte deste trabalho de conclusão de curso.

Todas as ferramentas utilizadas bem como o protocolo MIDI [2] foram estudados e explanados em um nível de detalhes que não prejudicasse a compreensão da ferramenta *Solfeggiare* que foi desenvolvida e não se tornasse maçante para o leitor deste trabalho.

O *Solfeggiare*, uma ferramenta de propósito geral para compilação de partituras musicais, capaz de transformar descrições de partituras em melodias gravadas. Todas as melodias geradas são armazenadas em arquivos MIDI, amplamente suportados pela grande maioria dos computadores e sintetizadores existentes no mercado.

As pesquisas por ferramentas OCR que substituiriam as *Score Input Languages* ainda não deram grandes passos. As *Score Input Languages* ainda são bastante utilizadas por compositores e o seu desenvolvimento tem diminuído bastante o tempo de composição. Elas também abriram um novo leque de possibilidades para os compositores no que diz respeito a utilização de diversos intrumentos e arranjos diferentes durante o processo de composição. A utilização de *frameworks* modernos no desenvolvimento do *Solfeggiare* mostrou que muito ainda pode ser feito principalmente pela gama de possibilidades que estes podem prover. APIs de criação de arquivos MIDI e de suporte a composição em Java estão em desenvolvimento, algumas com o prospecto de inserir suporte gráfico ao processo de composição.

As ferramentas e as APIs *open source* se mostraram cruciais durante todo o desenvolvimento, diminuindo a complexidade da implementação bem como o tempo de codificação.

6.1 Contribuições e Trabalhos Futuros

O desenvolvimento e a evolução de *Score Input Languages* pode ajudar a desmistificar o aprendizado da teoria musical e, conseqüentemente, da notação musical do ocidente. O *Solfeggiare* dá um primeiro passo para facilitar o aprendizado da música e dar a oportunidade a todos, jovens e idosos, para compor melodias ou até mesmo acelerar este processo.

Vários pontos de extensão puderam ser identificados ao fim da elaboração deste trabalho. O próprio *Solfeggiare* já possui um módulo de geração de partituras no formato MusicXML [3] a partir de um objeto representativo de uma partitura. Este módulo pode ser futuramente utilizado para integração com uma interface gráfica de transformação de partitura gráfica em arquivo MusicXML de maneira transparente para o usuário.

O *back-end* do *Solfeggiare* pode ser atualizado dando maiores possibilidades a elementos performáticos presentes em partituras. O compilador atual trata os elementos mais simples. Um compilador com mais possibilidades performáticas seria mais atraente e daria maior visibilidade ao *Solfeggaire*, podendo se tornar, inclusive, um diferencial dele para as demais *Score Input Languages*.

O projeto *Audiveris* [31] que está sendo desenvolvido por Hervé Bitteur planeja liberar uma nova versão onde uma ferramenta OCR seria utilizada para ler partituras gráficas. Tais partituras serão interpretadas por esta ferramenta e transformadas em arquivos MusicXML, abrindo mais um leque de possibilidades para o *Solfeggiare*. A integração com o *Audiveris*, que também é desenvolvido em Java, dará nova vida a esta *Score Input Language*, evitando que o compilador fique obsoleto como previsto por Milton Babbit.

6.2 Dificuldades Encontradas

A maioria dos artigos científicos relevantes publicados na área de Computação Musical é publicada no *Computer Music Journal* que possui uma assinatura relativamente cara. As bases de dados de artigos mais conhecidas como IEEE e CAPES possuem um número de artigos muito pequeno. Conseguir referências e trabalhos relacionados a *Score Input Languages* foi um trabalho árduo.

Uma partitura pode ser tão complexa quanto desejar o compositor. Uma série de elementos performáticos pode ser introduzida, porém nem todos são triviais de serem interpretados e executados por uma máquina. As ligaduras, por exemplo, não podem ser reproduzidas por qualquer ambiente interpretador de mensagens do protocolo MIDI. Algumas máquinas podem não conseguir executar alguns elementos, dando à música um aspecto "mecânico", sem um ar natural.

Todos os *frameworks* e APIs que foram utilizados são *open source*, ou seja, de livre utilização pela comunidade. Os maiores projetos são bem reconhecidos e bem documentados, porém os menores, apesar de bons, não estão bem documentados. Isto aumenta o tempo de implementação e cria algumas barreiras durante o desenvolvimento de um projeto.

Os arquivos MIDI gerados pela API JFugue nem sempre podem ser executados fielmente em todos os dispositivos. Algumas mensagens de controle não são suportadas por arquiteturas mais simples, anulando alguns efeitos performáticos que foram inseridos na melodia e tornando a execução da mesma mecânica, artificial.

Bibliografia

[1] ROADS, C. et al. 1995. The Computer Music Tutorial. Massachusetts. The MIT Press, 1234 p.
[2] MIDI 1.0 Specification. International MIDI Association. Disponível em: http://www.midi.org. Acessado em 12 de abril de 2006.
[3] GOOD, M. et al. MusicXML Definition, Disponível em: http://www.recordare.com/xml.html. Acesso em: 4 de maio de 2006.
[4] HUNTER, J., MCLAUGHLIN, B. JDOM, Disponível em: http://www.jdom.org. Acesso em: 4 de maio de 2006.
[5] KOELLE, D. JFugue: Java API for Music Programming, Disponível em: http://www.jfugue.org. Acesso em: 4 de maio de 2006.
[6] ECLIPSE. Eclipse Foundation. Disponível em: http://www.eclipse.org. Acesso em: 12 de maio de 2006.
[7] PRIOLLI, M. 1979. Princípios Básicos da Música Para a Juventude. Rio de Janeiro. Casa Oliveira de Músicas LTDA. 144 p.
[8] DE SOUZA, C. et al. 1997. O Mundo Maravilhoso da Música. São Paulo. Melhoramentos. 125 p.
[9] WIKIPEDIA. Wikipedia – A enciclopédia livre. Disponível em: http://www.wikipedia.org. Acesso em: 5 de abril de 2006.
[10] ERICKSON, E. 1975. The DARMS project: a status report. Computers and the Humanities. cap. 7, p. 291-298.
[11] WENKER, J. 1972. MUSTRAN II – an extended music translator. Computers and the Humanities. cap. 7.
[12] ANDERSON, D., KUIVILA, R. Formula: A Programming Language for Expressive Computer Music. IEEE Computer Society Press, 1992.
[13] PARKER, J.R. 1995. Algorithms for Image Processing and Computer Vision. John Wiley and Sons.
[14] ENTWISTLE, J. Visual perception and the analysis of music scores. Trabalho de Graduação. Cambridge. The MIT Department of Humanities. 1973.
[15] MATSUSHIMA et al. Automated recognition system for musical score – the visual system of WABOT-2. Bulletin of Science and Engineering Research Laboratory. Waseda University, cap. 112, p. 25-52.
[16] DE POLI, G. Musica – programme de codage de la musique. Rapports IRCAM 7/78. Paris.
[17] RAMALHO, G. Construction d'un agent rationnel jouant du jazz. Tese de doutorado da Universidade de Paris VI. 1997.
[18] TERRY, P. 1977. Compilers and Compiler Generators – an introduction with C++, International Thomson Computer Press.
[19] VERMEIR, D. 2001. An introduction to compilers. Free University of Brussels.

[20] WAHLIN, D. 2003. XML e ASP.NET para desenvolvedores. Pearson Education do Brasil.

[21] APACHE. The Apache Software Foundation. Disponível em: http://www.apache.org. Acesso em: 13 de maio de 2006.

[22] JACOBSON, I., BOOCH, G., RUMBAUGH, J. 1998. The Unified Software Development Process. Addison-Wesley.

[23] COSTALONGA, L. et al. Bibliotecas Java Aplicadas a Computação Musical. Instituto de Informática – UFRGS. SBCM 2005.

[24] MATHEWS, M., MOORE, F. 1970. GROOVE – a program to compose, store and edit functions of time. Communications of the Association for Computing Machinery. p. 715-721.

[25] KOBRIN, E. 1977. Computer in Peformance. DAAD. Berlin.

[26] MOORE, F. 1988. The dysfunctions of MIDI. Computer Music Journal. p. 19-28.

[27] NEWCOMB, S., GOLDFARB, C. 1989. X3V1.8M/SD-6 Journal of Development, Standard Music Description Language (SMDL), Part One: Objectives and Methodology. San Francisco. International Computer Music Association.

[28] SMITH, L. 1972. SCORE – a musician's approach to computer music. Journal of the Audio Engineering Society. p. 7-14.

[29] New England Digital Corporation. 1981. SCRIPT user guide. White River Junction. New England Digital Corporation.

[30] AMES, C. 1982. Protocol: motivation, design, and production of a composition for solo piano. cap. 11, p. 213 – 238.

[31] BITTEUR, H. The Audiveris Project. Disponível em: http://audiveris.dev.java.net. Acesso em: 14 de maio de 2006.

Apêndice A

Arquivo MusicXML de “Este Seu Olhar”

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 1.0 Partwise//EN" "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
   <movement-title>Este Seu Olhar</movement-title>
        <identification>
                <creator type="composer">Tom Jobim</creator>
                <creator type="transcriber">Marcelo Nunes</creator>
        </identification>
        <part-list>
                <score-part id="P1">
                        <part-name>Piano</part-name>
                        <score-instrument id="P1-I3">
                                <instrument-name>Piano</instrument-name>
                        </score-instrument>
                </score-part>
        </part-list>
        <part id="P1">
                <measure number="1">
                        <attributes>
                                <divisions>24</divisions>
                                <key>
                                        <fifths>-1</fifths>
                                        <mode>major</mode>
                                </key>
```

```
        <time>
            <beats>2</beats>
            <beat-type>2</beat-type>
        </time>
        <clef>
            <sign>G</sign>
            <line>2</line>
        </clef>
</attributes>
<note>
        <pitch>
            <step>A</step>
            <octave>4</octave>
        </pitch>
        <type>quarter</type>
</note>
<note>
        <pitch>
            <step>B</step>
            <octave>4</octave>
        </pitch>
        <type>eighth</type>
</note>
<note>
        <pitch>
            <step>C</step>
            <octave>5</octave>
        </pitch>
        <type>quarter</type>
</note>
<note>
        <pitch>
            <step>D</step>
            <octave>5</octave>
        </pitch>
        <type>quarter</type>
</note>
<note>
        <pitch>
            <step>C</step>
```

```
                <octave>5</octave>
            </pitch>
            <type>eighth</type>
            <tie type="start"/>
        </note>
</measure>
<measure number="2">
        <note>
            <pitch>
                <step>C</step>
                <octave>5</octave>
            </pitch>
            <type>whole</type>
            <tie type="stop"/>
        </note>
</measure>
<measure number="3">
        <note>
            <pitch>
                <step>B</step>
                <octave>4</octave>
            </pitch>
            <type>quarter</type>
        </note>
        <note>
            <pitch>
                <step>C</step>
                <octave>5</octave>
            </pitch>
            <type>eighth</type>
        </note>
        <note>
            <pitch>
                <step>D</step>
                <octave>5</octave>
            </pitch>
            <type>quarter</type>
        </note>
        <note>
            <pitch>
```

```
                    <step>E</step>
                    <octave>5</octave>
                </pitch>
                <type>quarter</type>
            </note>
            <note>
                <pitch>
                    <step>D</step>
                    <octave>5</octave>
                </pitch>
                <type>eighth</type>
                <tie type="start"/>
            </note>
        </measure>
        <measure number="4">
            <note>
                <pitch>
                    <step>D</step>
                    <octave>5</octave>
                </pitch>
                <type>whole</type>
                <tie type="stop"/>
            </note>
        </measure>
        <measure number="5">
            <note>
                <pitch>
                    <step>C</step>
                    <octave>5</octave>
                </pitch>
                <type>quarter</type>
            </note>
            <note>
                <pitch>
                    <step>D</step>
                    <octave>5</octave>
                </pitch>
                <type>eighth</type>
            </note>
            <note>
```

```
                <pitch>
                        <step>E</step>
                        <octave>5</octave>
                </pitch>
                <type>quarter</type>
        </note>
        <note>
                <pitch>
                        <step>F</step>
                        <octave>5</octave>
                </pitch>
                <type>quarter</type>
        </note>
        <note>
                <pitch>
                        <step>E</step>
                        <octave>5</octave>
                </pitch>
                <type>eighth</type>
                <tie type="start"/>
        </note>
</measure>
<measure number="6">
        <note>
                <pitch>
                        <step>E</step>
                        <octave>5</octave>
                </pitch>
                <type>eighth</type>
                <tie type="stop"/>
        </note>
        <note>
                <pitch>
                        <step>F</step>
                        <octave>5</octave>
                </pitch>
                <type>quarter</type>
        </note>
        <note>
                <pitch>
```

```
                        <step>G</step>
                        <octave>5</octave>
                    </pitch>
                    <type>quarter</type>
                </note>
                <note>
                    <pitch>
                        <step>A</step>
                        <octave>5</octave>
                    </pitch>
                    <type>quarter</type>
                </note>
                <note>
                    <pitch>
                        <step>G</step>
                        <octave>5</octave>
                    </pitch>
                    <type>eighth</type>
                    <tie type="start"/>
                </note>
            </measure>
            <measure number="7">
                <note>
                    <pitch>
                        <step>G</step>
                        <octave>5</octave>
                    </pitch>
                    <type>half</type>
                    <tie type="stop"/>
                </note>
                <note>
                    <pitch>
                        <step>F</step>
                        <octave>5</octave>
                    </pitch>
                    <type>quarter</type>
                </note>
                <note>
                    <pitch>
                        <step>G</step>
```

```
                        <octave>5</octave>
                    </pitch>
                    <type>eighth</type>
                </note>
                <note>
                    <pitch>
                        <step>A</step>
                        <octave>5</octave>
                    </pitch>
                    <type>eighth</type>
                </note>
            </measure>
            <measure number="8">
                <note>
                    <pitch>
                        <step>G</step>
                        <octave>5</octave>
                    </pitch>
                    <type>whole</type>
                </note>
            </measure>
        </part>
</score-partwise>
```