UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LEANDRO LESQUEVES COSTALONGA

# Polvo Violonista: Sistema Multiagente para Simulação de Performances em Violão

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Profª. Drª. Rosa Maria Vicari
Orientadora

Prof. Dr. Daniel Wolff
Co-orientador

Porto Alegre, abril de 2005

## CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Costalonga, Leandro Lesqueves

Polvo Violonista: Sistema Multiagente para Simulação de Performances em Violão / Leandro Lesqueves Costalonga. – Porto Alegre: PPGC da UFRGS, 2005.

87f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientadora: Rosa Maria Vicari, Co-orientador: Daniel Wolff.

1. Computação Musical. 2. Sistema Multiagentes. 3. I. Vicari, Rosa Maria. II. Wolff, Daniel. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. José Carlos Ferraz Hennemann

Vice-Reitor: Prof Pedro Cezar Dutra Fonseca

Pró-Reitora de Pós-Graduação: Valquiria Linck Bassani

Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux

Coordenador do PPGC: Prof. Flávio Rech Wagner

Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

# AGRADECIMENTOS

Agradeço aos meus orientadores, Rosa Vicari e Daniel Wolff, pela paciência e dedicação que tiveram comigo ao longo do mestrado. Ainda, um agradecimento especial ao Professor Eloi Fritsch que mediou minha vinda a Porto Alegre e com quem aprendi tudo que sei sobre computação musical. Não poderia deixar de lembrar e agradecer aos meus colegas e companheiros de grupo: Evandro Miletto, Luciano Flores, Vinicius Nobile e Rafael de Oliveira.

Especial agradecimento aos meus pais e avós que não mediram esforços e dedicação na minha formação educacional. Ainda, agradeço ao Yuuki, meu grande parceiro canino das madrugadas de estudo e solidão nos frios domingos gaúchos.

# SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS ........................................................ 7
LISTA DE FIGURAS ........................................................................................ 8
LISTA DE TABELAS ....................................................................................... 10
RESUMO......................................................................................................... 11
ABSTRACT..................................................................................................... 12
1 INTRODUÇÃO ........................................................................................... 13
1.1 Motivação ........................................................................................... 14
1.2 Trabalhos Relacionados ........................................................................ 15
1.3 Objetivos.............................................................................................. 20
1.4 Contribuições ....................................................................................... 21
2 FUNDAMENTAÇÃO TEÓRICA ................................................................. 23
2.1 Conceituação de Agentes ...................................................................... 23
2.2 Referencial Teórico de Música .............................................................. 26
2.3 Resumo do Capítulo ............................................................................. 32
3 CONCEPÇÃO DO SISTEMA...................................................................... 34
*3.1* Requisitos Identificados ....................................................................... 34
3.2 Abordagem Multiagente ....................................................................... 37
3.3 Esquema de comunicação ..................................................................... 43
3.4 Escolha das tecnologias envolvidas na construção .................................. 44
3.5 Resumo do Capítulo ............................................................................. 49
4 BIBLIOTECAS PARA PROGRAMAÇÃO MUSICAL - DESENVOLVIMENTO FOCANDO REUSO.............................................. 50
4.1 Pacote de Formação de Acordes (br.ufrgs.inf.lcm.formacaoAcorde); ........ 50

4.2 Pacote de Geração das Representações dos Acordes (br.ufrgs.inf.lcm.representacaoAcorde) ........ 58

4.3 Resumo do Capítulo ........ 74

5 CONSTRUÇÃO DO PROTÓTIPO ........ 75

*5.1* Interface Gráfica – Pacote Webfake ........ 75

5.2 Adaptações do Pacote de Interface ao Protótipo ........ 79

5.3 Componente para Desenho do Padrão Rítmico ........ 79

5.4 Criando uma composição ........ 81

5.5 Criando Agente ME e Definindo a Harmonia ........ 81

5.6 Criando o Agente MD e os Padrões Rítmicos ........ 83

5.7 Ouvindo os agentes pela "Caixa de Som" ........ 87

5.8 Resultados ........ 88

5.9 Resumo do Capítulo ........ 89

6 CONCLUSÃO ........ 90

6.1 Contribuições e Resultados Obtidos ........ 90

6.2 Publicações ........ 91

6.3 Trabalhos Futuros ........ 93

REFERÊNCIAS ........ 95

ANEXO A ANÁLISE DE SOFTWARES MUSICAIS FOCADOS EM VIOLÃO/GUITARRA ........ 98

Introdução ........ 98

Dicionário de Acordes ........ 98

Editores de Tablatura ........ 99

Sistemas de Treinamento ........ 101

Software para Composição ........ 101

ANEXO B TECNOLOGIAS JAVA APLICADAS A COMPUTAÇÃO MUSICAL ........ 103

Introdução ........ 103

Java Sound (JSDK 1.4.2) ........ 103

**JMSL – Java Music Specification Language** ........ **107**

**JMusic** ........ **112**

**Wire /Wire Provider** ........ **123**

**JavaMIDI - 2001** ........ **124**

**NoSuch MIDI** ........ **124**

**MIDI Share** ........ **124**

**MIDI Kit** ........ **124**

**JFugue** ........ **125**

**Tritonus** ........ **125**

**JASS (Real-time audio synthesis)** ........ **126**

**Jsyn** ........ **127**

**JSCORE** ........ **128**

**Xemo** ........ **128**

**Conclusão** ........ **129**

**ANEXO C DOCUMENTAÇÃO DOS PACOTES DE FORMAÇÃO E REPRESENTAÇÃO DE ACORDES** ........ **130**

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | Application-Program Interface |
| BPM | Batidas por Minuto (beats per minute) |
| CME | Centro de Música Eletrônica (UFRGS) |
| CM | Computação Musical |
| IA | Inteligência Artificial |
| IPS | Instrument Performance System |
| MAS | Multiagent System |
| GUI | Graphical User Interface |
| JDK | Java Development Kit |
| LCM | Laboratório de Computação Musical (UFRGS) |
| MIDI | Musical Instrument Digital Interface |
| OOP | Object oriented programming |
| PC | Personal Computer |
| UFRGS | Universidade Federal do Rio Grande do Sul |
| RMI | Remote Method Invocation |
| UML | Unified Modeling Language |

# LISTA DE FIGURAS

Figura 1.1 Tratamento do padrão rítmico pelo Power Tab........................................... 17
Figura 1.2: Interface para definição harmônica do plug-in Rythm ´n Chord................ 18
Figura 1.3: Representação gráfica de um padrão rítmico............................................. 19
Figura 2.1: Exemplo de tablatura em notação comum nos EUA. ................................. 26
Figura 2.2: Exemplo de registro musical popular normalmente encontrada no Brasil. . 27
Figura 2.3: Diagramações de Acordes (desenhos dos acordes).................................... 28
Figura 2.5: Partitura de bateria ...................................................................................... 30
Figura 2.6: Notação para partituras de bateria............................................................... 31
Figura 3.2: Diagrama de Interação entre os agentes do sistema.................................. 39
Figura 3.3: Dependências do Agente ME....................................................................... 40
Figura 3.4: Dispositivo para tratar a latência das mensagens MIDI.............................. 44
Figura 4.1 : Classe CifraValida ...................................................................................... 52
Figura 4.2 : Classe CifraValidaMaker............................................................................. 52
Figura 4.3 : Autômato finito inicial................................................................................. 54
Figura 4.4: Autômato para tratamento de tétrades (acorde com sétima) a partir do estado S2 .............................................................................................................. 55
Figura 4.5: Autômato para tratamento de acorde com os intervalos de segunda, quarta ou sextas. ............................................................................................... 55
Figura 4.6: Autômato para tratamento de tétrades diminutas....................................... 55
Figura 4.7: Autômato para tratar acordes menores. .................................................... 56
Figura 4.8: Autômato para tratamento de acordes menores com intervalo de sétima.... 56
Figura 4.9: Autômato para tratar acordes força (intervalo dobrado de quinta)............. 56
Figura 4.10: Autômato que trata alteração no baixo do acorde e inversões.................. 56
Figura 4.11: Classes associadas a classe Acorde.......................................................... 57
Figura 4.12: Classe Acorde ........................................................................................... 57
Figura 4.13: Classe AcordeMaker.................................................................................. 58
Figura 4.14: Processo de geração das representações básicas de acordes.................... 59
Figura 4.15: Dicionário de acordes de representações básicas..................................... 59

Figura 4.16: Ligações da classe Instrumento ........................................................ 60
Figura 4.17: Classe InstrumentoCorda ................................................................ 61
Figura 4.18 : Criando um violão.......................................................................... 62
Figura 4.19: Particularidades de uma representação de acorde para um instrumento de corda................................................................................................................ 63
Figura 4.20: Representação do acorde Dó Maior. ................................................. 64
Figura 4.21: Algumas possibilidades do acorde Sol Maior no violão...................... 64
Figura 4.22 : Implementação de um gerador de representações de acordes para instrumentos de corda. ........................................................................................ 66
Figura 4.23: Classe RepresentacaoAcordeInstrumentoCordaMaker.......................... 67
Figura 4.24: Classe ProcessadorRepresentacaoAcordeInstrumentoCorda.................. 69
Figura 4.25: Processamento de dobramentos em reapresentações básicas. ............... 71
Figura 4.26: Simulador de teste da escolha das representações dos acordes. ............ 73
Figura 5.1: Diagrama de classes simplificado .......................................................... 76
Figura 5.2: Aplicação Exemplo ............................................................................... 78
Figura 5.3: Estrutura de árvore que organiza os agentes do sistema.......................... 79
Figura 5.4: Componente para criação do padrão rítmico. .......................................... 80
Figura 5.5: Exemplo de padrão rítmico. .................................................................. 80
Figura 5.6: Tela de criação da composição. ............................................................. 81
Figura 5.7: Informações de um Gdim....................................................................... 82
Figura 5.8: Tela do Agente ME. .............................................................................. 82
Figura 5.9: Tela do Agente MD ............................................................................... 83
Figura 5.10: Tela da aplicação do padrão rítmico a harmonia. ................................... 86
Figura 5.11: Tela de execução da composição. ......................................................... 86
Figura 5.12: Tela de *Log*. ........................................................................................ 87
Figura 5.13: Tela do Agente Caixa de Som .............................................................. 88

# LISTA DE TABELAS

Tabela 3.1: Comparação entre API's que processam áudio. ........................................ 46
Tabela 3.2: Comparação entre API's que sintetizam áudio.. ........................................ 46
Tabela 3.3: Comparação entre API's que sequenciam eventos MIDI.. ......................... 46
Tabela 3.4: Comparação entre API's que enviam eventos s MIDI a dispositivos externos. ................................................................................................ 47
Tabela 3.5: Comparação entre API's que geram sons a partir de eventos MIDI. .......... 48
Tabela 3.6: Comparação entre API's que exibem informações musicais em interfaces gráficas. ................................................................................................ 48
Tabela 3.7: Comparação entre API's que trabalham com formatos de arquivos musicais. ................................................................................................ 48
Tabela 3.8: Comparação entre API's criadas para serem usadas na composição musical. ................................................................................................ 49
Tabela 4.1: Símbolos definidos na notação brasileira de cifragem ............................... 53
Tabela 4.2 : Cálculo de Similaridade ............................................................................ 72

# RESUMO

Este trabalho apresenta um sistema multiagente que permite simular execuções musicais em violão. Em uma execução musical no violão observam-se elementos distintos que, trabalhando em conjunto, produzem a sonoridade desejada pelo músico. Ou seja, as ações tomadas pelo violonista, através do sincronismo de suas mãos sobre o instrumento, influenciam diretamente na sonoridade obtida. A idéia básica deste trabalho é desenvolver uma comunidade de agentes que represente as entidades envolvidas em uma performance musical de violão. Quatro agentes foram identificados e modelados no contexto desta dissertação. São eles:

**Mão Esquerda (ME):** Responsável pela execução dos acordes, ou seja, deve possuir o conhecimento de formação de acordes dada a afinação do instrumento bem como a interpretação das cifras que representam os acordes.

**Agente Mão Direita (MD):** Responsável pelo ritmo impresso na música;

**Caixa de Som (CS):** Permite que os usuários simplesmente escutem a composição, sem nenhuma (ou muito pouca) interferência na composição.

**Agente Solista (SL)**: Projetado somente para ler arquivos MIDI e enviar notas para o Agente Caixa de Som (CS).

O conhecimento relativo ao reconhecimento das cifras, geração do acorde e posterior cálculo do desenho do acorde para um instrumento de corda foi encapsulado em duas bibliotecas que visam auxiliar no desenvolvimento de outros novos projetos que necessitem de funcionalidades similares. Ainda, são abordadas as questões da comunicação entre os agentes e componentes gráficos utilizados na captura de informações rítmicas.

O material musical produzido pelo sistema está contido no CD-ROM em anexo, bem como a documentação das API's.

**Palavras-chave**: computação musical, sistema multiagente, sistemas de performance musical, padrões rítmicos, formação de acordes.

A Multiagent System to Simulate Guitar Performances

# ABSTRACT

This work presents a multiagent system that allows the simulation of musical guitar performance. Distinct elements are clearly identified in a guitar performance. These elements working together produce the desired sound imagined by the musician. In other words, the actions taken by the synchronized musicians' hands directly affect the sound obtained. The main idea of this work is the development of a community of agents that represents the entities involved in a guitar performance.

Four agents were identified and modeled in this work context. They are:

**Left-Hand Agent (LH):** Responsible for the chord's execution, this means, it must have the knowledge of chord's composition once the instrument tuning is given, as well as the textual chord's symbol recognition.

**Right-Hand Agent (RH):** Responsible for the music rhythm.

**Speaker Agent (SPK):** Allow the user to hear the music, without interfering.

**Solo Agent:** Designed to read a MIDI file and send the events to the SPK Agent. In the future, it will represent a human interacting with the system.

The knowledge related to the chord's symbol recognition, chord's composition and chord shape calculus were encapsulated in two libraries that aim to assist future works with similar features. Also, communication issues and graphical components used to capture rhythm information are presented. The sound material produced as a result of the system is in the attached CD-ROM, as well as the API documentation.

**Keywords**: computer music, multiagent system, musical performance system, rhythm pattern, chord's composition.

# 1 INTRODUÇÃO

Com a evolução e popularização da informática, ferramentas usadas no processo de criação sonora migraram para o computador pessoal a um custo aceitável aos músicos e entusiastas da música. Muitos desses usuários não possuem formação musical acadêmica criando uma demanda por software mais simples de serem usados e que não exigem profundos conhecimentos musicais. Apesar de sua popularidade, o violão não teve sua interface projetada em muitas dessas ferramentas, sendo preferido os teclados, seja por uma questão histórica ou pela facilidade na implementação. Assim sendo, os violonistas (amadores ou profissionais) devem aprender a lidar com os teclados para fazer uso destas ferramentas.

Em uma execução musical no violão observam-se elementos distintos que, trabalhando em conjunto, produzem a sonoridade desejada pelo músico. Ou seja, as ações tomadas pelo violonista, através do sincronismo de suas mãos sobre o instrumento, influenciam diretamente na sonoridade obtida. Cada elemento tem um papel neste cenário. Muitas das atividades destes papéis, por vezes, possuem uma interdependência. Este é o caso da mão esquerda e da mão direita do violonista ao tocar um acorde.

Um dos objetivos do sistema proposto é realizar a simulação do ritmo nas performances de violão popular, permitindo inclusive simular a execução da música por diversos violonistas.Tal ferramenta pode ser usada para auxiliar no ensino do instrumento e dos conceitos rítmicos, na criação da parte do violão em uma composição ou simplesmente por entretenimento, como no caso de músicos amadores.

O uso de agentes artificiais representando as mãos do violonista permite simular situações para além da condição natural do instrumentista, como por exemplo, a sobreposição de diversos ritmos em uma mesma seqüência harmônica, bem como a inclusão de melodias, criando-se um efeito similar ao de um violonista com diversas mãos direitas tocando paralelamente. Para efeito de convenção, adotou-se como padrão o instrumentista destro, aquele que digita os acordes com a mão esquerda e percute as cordas com a mão direita.

Popularmente, os padrões rítmicos no violão são conhecidos como “batidas” e “dedilhados”. Estes padrões rítmicos associados com os acordes compõem a linha harmônica de uma música acompanhada por violão. Os acordes já são suficientemente

bem representados pelas cifras ou até mesmo por gráficos indicando o posicionamento dos dedos no instrumento, entretanto os padrões rítmicos não possuem uma representação clara e universalmente aceita em notações musicais[1] alternativas a clássica.

O foco do trabalho está no ritmo e sua representação (notação) para músicos (ou aprendizes) sem uma formação acadêmica. O objetivo é conseguir representar computacionalmente padrões rítmicos para violão e trabalhá-los de forma a obter novas sonoridades.

A idéia básica deste trabalho é desenvolver uma comunidade de agentes que represente as entidades envolvidas numa performance de violão, baseando-se na hipótese de que diversos problemas musicais podem ser resolvidos por meio de comunidades de agentes (WESSEL, 2002; MIRANDA, 2002; WULFHORST, 2001).

## 1.1 Motivação

Vários são os fatores que motivam a realização desse trabalho. Considerando-se as pesquisas que têm sido desenvolvidas no Laboratório de Computação Musical, do Instituto de Informática e do Centro de Música Eletrônica, do Instituto de Artes da UFRGS, relacionados à área de tecnologia aplicada a música, a presente pesquisa representa uma continuidade dessas atividades ampliando o seu alcance e sua divulgação.

Quando se fala em notação alternativa a notação musical clássica é importante salientar que as mesmas devem registrar, ao menos, as principais informações da música para que ela possa ser reproduzida tal qual foi composta (harmonia, melodia e ritmo). Visando um público crescente que demanda simplicidade, é comum haver uma separação dos diversos elementos musicais (WEST et al. 1991) e normalmente, as informações registradas sofrem simplificações e adequações ao instrumento que será usado na execução. Notações alternativas que utilizam acordes cifrados, como tablatura [2], são talvez as mais usadas para registrar musicas populares que serão tocadas no violão, apesar de serem normalmente extremamente limitadas no aspecto rítmico.

Apesar de existirem algumas diferenças em função da cultura local do músico e do estilo musical, as cifras são largamente utilizadas nas notações musicais mais simples e populares (SHER, 1991) e se concentram no componente harmônico da música, supondo o conhecimento da melodia e do ritmo por parte do músico, ou seja, no caso do

[1] Método de representação gráfica dos sons de uma obra musical, de seu valor, duração etc. de modo que ela possa ser lida para execução.

[2] Forma de notação em que a nota a ser tocada (ou o acorde) é indicada pela posição dos dedos do executante no braço do instrumento. Estenderemos o conceito de tablatura para qualquer notação, voltada a instrumentos trasteados, que se utilize acordes cifrados e sua diagramação.

violão, as informações das cifras são praticamente destinadas à mão esquerda do músico.

As informações sobre o momento e a maneira em que cada uma das notas de um acorde vai soar estão ligadas ao ritmo e ao andamento da música. Estas informações, relacionadas à mão direita do violonista, geralmente não são tratadas nas tablaturas, o que pode se transformar em um problema na performance musical. Os diversos tipos de notação gráfica para a música contemporânea retornam ao caráter indeterminado das notações mais antigas, dando ao intérprete participação na própria feitura da obra (ZAHAR, 1982).

Uma notação proporcional, capaz de registrar valores de tempo, começou a ser utilizada no século X, usando notas de diferentes formatos, embora as linhas de compasso somente fossem introduzidas no séc. XV. A atual notação em pentagrama desenvolveu-se gradualmente no decorrer desse período, e tem estado continuamente em uso há mais de 400 anos (ZAHAR, 1982). Notações registradas em papel, uma mídia analógica e estática, e que conseguem trabalhar com ritmo, como a partitura[3], exigem um treinamento prévio do leitor junto a um tutor, justo o que notações simplificadas tentam evitar. Sendo o computador um excelente medidor de tempo, essas notações musicais simplificadas ganham força em uma versão computacional.

## 1.2 Trabalhos Relacionados

Apesar do seu potencial para software de acompanhamento automático, a geração de ritmo no violão não tem sido discutida na literatura da computação musical (DAHIA, 2003). Recentemente começaram aparecer pesquisas no Brasil que trabalham com ritmos de violão em sistemas computacionais, onde a UFPE tem merecido destaque com o sistema de acompanhamento musical denominado Cyber-João.

No Cyber-João, 211 fragmentos musicais de tamanho fixos foram indexados e classificados. Estes fragmentos foram extraídos de gravações originais de João Gilberto através da ferramenta VexPat (SANTANA, 2003). A escolha da Bossa Nova como ritmo deu-se em virtude da boa documentação e a escolha do uso de fragmentos musicais ao invés de gerar notas em tempo real foi justificada pelo pouco esforço de processamento. É fato, entretanto, que o grupo de pesquisa já havia obtido bons resultados com o sistema ImPact (RAMALHO, 1997) que utiliza a mesma técnica de fragmentos musicais, porém com outro instrumento (baixo) e ritmo (jazz).

A recuperação dos fragmentos que são usados no acompanhamento é feito usando regras de produção e raciocínio baseado em casos (*Case-Based Reasoning*) e tem como entrada a linha harmônica e melódica (intenção musical). Ao que consta, este trabalho

---

[3] A música combinada para todas as partes em uma composição vocal ou instrumental.

continua em andamento, entretanto bons resultados já foram obtidos e avaliados por especialistas.

Um outro trabalho que adota fragmentos musicais classificados por estilos musicais é a ferramenta comercial Band-in-a-box (BAND-IN-A-BOX, 1998). Diversos instrumentos podem ser usados no acompanhamento e solos automáticos e cada estilo musical possui mais de 600 fragmentos de tamanhos variados e indexados por um peso definido pelo usuário. Neste sistema, novos padrões rítmicos também podem ser criados. A escolha do fragmento é randômica.

Na dissertação de mestrado intitulada "Uma Abordagem Multiagentes para Sistemas Musicais Interativos" (WULFHORST, 2002), o autor afirma que dentre as pesquisas que procuram abordar aspectos da Percepção Musical, a maior parte do esforço tem se concentrado principalmente em aspectos relacionados à classificação harmônica. Muitos musicólogos, no entanto, consideram o ritmo como sendo o componente mais fundamental da música, presente desde os primórdios em todas as culturas (GABRIELSON, 1986).

Como estudo de caso de sua dissertação, um sistema de acompanhamento musical foi implementado usando o modelo denominado MMAS (Musical Multiagent System). "Implementamos agentes inteligentes capazes de acompanhar ritmicamente uma execução musical qualquer. Cada agente é responsável pela execução de seu instrumento. Seu papel principal é manter-se em sincronia com os outros agentes, 'ouvindo' e seguindo o andamento dos agentes de maior prioridade" (WULFHORST, 2002).

A técnica utilizada para obter o sincronismo rítmico entre os vários agentes musicais baseou-se na "percepção rítmica", descrito por Dannenberg e considerado pelo mesmo uma tarefa surpreendentemente difícil (DANNENBERG, 1997). Seu programa usa uma janela de tolerância estática, onde cada nota que é tocada neste intervalo é considerada como sendo uma nota tocada no "tempo forte". As outras notas, consideradas como subdivisões do tempo, são ignoradas. Quando a nota "tempo forte" soa antes do momento previsto, automaticamente o programa passa a diminuir proporcionalmente o período do pulso.

Utilizando uma arquitetura "*blackboard*", que centraliza as mensagens trocadas, o sistema permite que cada um dos agentes se auto-ajuste a partir dos eventos musicais lidos, dependendo de suas prioridades (que podem ser definidas pelo usuário). Nesta implementação pode-se definir um padrão rítmico para cada agente. O objetivo de cada agente é tocar seu instrumento em sincronia com os demais. Utilizando-se da técnica de percepção rítmica descrita anteriormente, o agente verifica se naquele momento deve tocar ou não o seu instrumento, de acordo com a definição de seu padrão rítmico.

O sistema proposto deve utilizar uma técnica híbrida para sugerir o padrão rítmico ao usuário. A intenção da música extraída da séria harmônica e melódica e a extração dos padrões rítmicos de composições devem ser similares às relatados no Cyber-João, bem como a percepção rítmica usada no MMAS deve ser utilizada em conjunto uma linha guia (voz ou outro violão). O principal diferencial das duas propostas é o fato do usuário poder definir seu padrão rítmico através de uma interface gráfica ou tocada em instrumento MIDI, assim como o uso de técnicas de I.A pra recuperar e aplicar os padrões rítmicos.

### 1.2.1 Software Comerciais

Foram analisados 26 software, cujo foco é o violão, distribuídos em 4 categorias inter-relacionadas à proposta deste trabalho: dicionário de acordes, editores de tablatura, sistema de treinamento e software para composição. Devido ao grande número de software musicais existentes enfocando o violão, foram escolhidos software cuja data de distribuição de suas últimas versões fossem superiores ao ano de 2002 e que possuíssem uma versão gratuita para avaliação. A lista completa dos software analisados encontra-se no Anexo A.

Dentre os software analisados, os mais relevantes para este trabalho e seus respectivos pontos fortes são:

**D´Accord Guitar Chord Dictionary 2.0:** Personalização do modo de visualização do acorde no braço do instrumento, permitindo que canhotos (cordas invertidas) não precisem se adaptar a visualização de acordes executados por destros;

**FretboardDots 2.2:** Não utiliza cifras para identificar o acorde, com isso consegue contornar o problema da falta de padronização dos símbolos. Os acordes são identificados pelos intervalos que o compõe.

**WinChord 4.2:** Permite alterar os parâmetros do algoritmo de cálculo do acordes como, por exemplo, a afinação do instrumento, quantidade de dedos, e abertura dos dedos, possibilitando um melhor ajuste as necessidades e restrições do usuário

**TablEdit 2.62:** Permite a inclusão de símbolos na tablatura referente a mão direita do violonista.

**Power Tab 1.7:** Trabalha com padrões rítmicos para violão e guitarra, fazendo uma junção de figuras rítmicas (Rhythm Slashes) da notação clássica com a tablatura, como mostra a Figura 1.

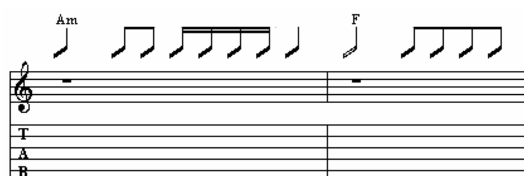



Figura 1.1 Tratamento do padrão rítmico pelo Power Tab

**WebLyrics 1.0.3.1:** Possui um interessante mecanismo para arpejar os acordes, associando os dedos do violinista as teclas do teclado numérico do computador.

**Guitar Trainer 2.4:** Associa a posição dos dedos no braço do instrumento à nota na partitura.

**Voicings 4.06:** Permite trabalhar separadamente as vozes dos acordes e sequenciando-as. É como se estivéssemos trabalhando o modo de arpejar, ou seja, informações de mão direita.

**Rhythm n Chords Lite 2.04:** Objetiva criar a parte de guitarra da composição através da criação de uma progressão de acordes associados a padrões rítmicos armazenados em bibliotecas de ritmos. Este software se baseia nos mesmos princípios que norteiam a proposta deste trabalho e será descrito detalhadamente para que se possa traçar as principais diferenças entre as duas propostas.

### 1.2.2 Rhythm n Chords Lite

Software desenvolvido pela MusicLab em um projeto de cooperação entre EUA e Rússia. Na verdade, é um *plug-in*[4] para os produtos da Cakewalk® e sua idéia principal é a separação da definição dos acordes da criação das estruturas rítmicas que toca.

A harmonia (os acordes) é definida através de uma interface (*Chord Chart View*) que imita a forma tradicional com que os guitarristas estão acostumados a trabalhar (pentagrama com as divisões dos compassos). Esta interface pode ser vista na Figura 1.2;

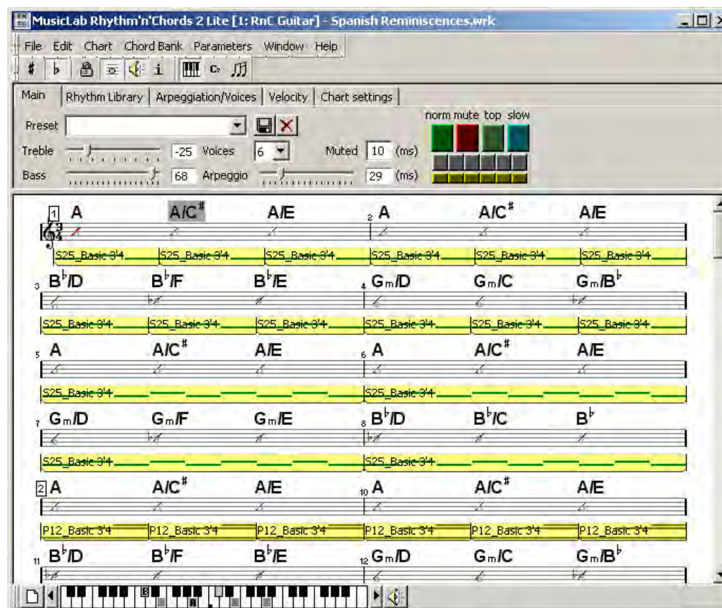



Figura 1.2: Interface para definição harmônica do plug-in Rythm ´n Chord

Os acordes podem ser inseridos usando símbolos padrões da notação americana organizados em um menu hierárquico ou escolhendo as notas em um teclado de piano virtual. Devido à integração com o Cakewalk, ainda é possível importar uma trilha MIDI com os acorde previamente definidos. Em teste com o software na versão Lite 2.04, a única forma possível foi através do menu de símbolos na notação americana sendo as outras formas descritas na documentação do fabricante.

O ritmo é definido através da associação de padrões rítmicos (pré-gravados) à trilha onde estão inseridos os acordes. Os padrões rítmicos podem ser selecionados de bibliotecas de ritmo, que acompanham o *plug-in*. São criados em frases rítmicas de 1, 2, 4 e 8 tempos e contem várias técnicas de guitarra como *strumming, muting, picking, plucking, bass and strum, slow strum, muted strum, glissando, slide, etc*

[4] Plug-in: Pedaço de software que estende a funcionalidade de um software previamente desenvolvido.

A representação visual do padrão rítmico é feita através de uma interface similar a *Piano-roll* [5](ROADS, 1996), e pode ser vista na vista na Figura 1.3. Cada uma das barras horizontais no interior da caixa representa um ataque (*stroke*) no violão e eles podem ser de vários tipos, identificados por sua coloração. Ataques normais são verdes, *muted* são vermelhos, ataques lentos são azuis, *tops* são verdes claros e os marrões são de cordas tocadas individualmente. A duração da nota produzida é representada pela extensão do traço. Para cada tipo de ataque existe uma configuração da velocidade do arpejo, da quantidade de vozes e do equilíbrio do volume sonoro entre as vozes (*voice balance*). O equilíbrio do volume das vozes, em especial, é um recurso que procura fidelizar o som gerado através do controle de amplitude de cada voz do acorde, ou seja, toma como referencia a terceira corda da guitarra e proporcionalmente a distancia do referencial (3º corda) é decrescido o volume.

O sistema faz a alternância automática entre os arpejos ascendentes e descendentes.

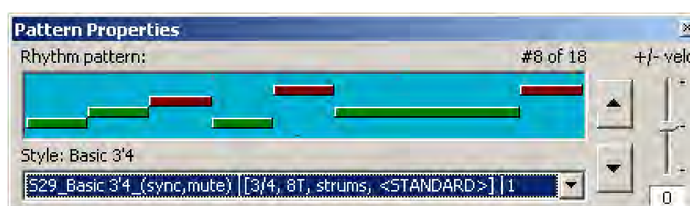



Figura 1.3: Representação gráfica de um padrão rítmico.

A parte resultante da junção do ritmo com a harmonia pode ser controlada por parâmetros de *velocity* (dinâmica e equilíbrio entre as vozes do acorde), *arpeggiation* (tempo entre a primeira e última nota do acorde) e polifonia (numero de vozes do acorde).

Um recurso interessante deste software é o tratamento que ele permite fazer para que a mudança de um acorde para o outro seja suave e sem sobressaltos. Isso é feito trabalhando as inversões dos acordes e assim, escolhendo a nota melódica do acorde (segundo documentação do fabricante). Também é possível modificar o baixo do acorde, o que o fabricante não considera uma inversão.

A idéia da separação da parte harmônica da rítmica é a mesma que norteia o desenvolvimento da proposta deste trabalho, onde a harmonia é considerada uma simples progressão de acordes e o ritmo uma seqüência de ataques sem notas definidas. Esta visão é diferente da notação musical tradicional onde as notas e as informações rítmicas estão combinadas.

O trabalho proposto difere da ferramenta apresentada nos seguintes pontos:

---

[5] Representação que deriva da época dos tocadores de piano onde as notas eram codificadas como cartões perfurados em rolo de papel. Tons individuais são assinalados verticalmente enquanto que o início do tempo e a duração dos eventos são codificados como pontos ou linhas horizontais, conforme a duração da nota.

- Definição harmônica com símbolos da notação brasileira de cifragem, permitindo configuração da mesma;
- Configuração do instrumento e do perfil do usuário;
- Possibilidade da construção de padrões rítmicos através de interface de clara compreensão para violonistas;
- Plataforma propícia para adição de inteligência artificial para sugestão de padrões rítmicos e computação afetiva.
- Uso independente de software de terceiros;
- Transição entre os acordes por similaridade dos desenhos.

### 1.2.3 Tecnologias Atuais Disponíveis aos Violonistas

Os aspectos rítmicos ligados ao violão possuem características próprias e que não são modeladas no principal protocolo de comunicação entre instrumentos digitais (MIDI).

Arpejadores e seqüenciadores são equipamentos (ou programas) facilmente encontrados e manipulados por músicos, principalmente os que se dedicam à criação de música eletrônica, entretanto existem algumas desvantagens na utilização destes equipamentos por violonistas:

A interface de utilização dos equipamentos é bem diferente da que os violonistas estão acostumados, pois estes equipamentos geralmente estão em teclados *workstations* ou em equipamentos especiais de interface pouco intuitivas.

Nos arpejadores mais simples, os programas de arpejo são estáticos, não permitindo a alteração, e mesmo que permitissem, seria de uma forma complexa demais para quem está acostumado a dedilhar as cordas para obter efeito similar.

Técnicas de mão de direita usadas por violonistas para conseguirem alteração na sonoridade e expressividade não são disponibilizadas por esses equipamentos.

## 1.3 Objetivos

O objetivo principal deste trabalho é a concepção e o desenvolvimento de um software que permita aos músicos (em especial violonistas) a criação de composições definindo separadamente os elementos musicais de uma execução em violão, ou seja, melodia, harmonia e ritmo.

Com esta visão, objetiva-se possibilitar a experimentação sonora de novos padrões rítmicos, fornecendo fragmentos musicais para composições e base tecnológica para sistemas de acompanhamento em tempo real.

### 1.3.1 Objetivos Específicos

- Reconhecer cifras na notação brasileira (ou personalizada) e conseqüente formação dos acordes;
- Calcular os desenhos de acorde baseados em instrumentos virtuais (configuráveis) e características dos usuários;
- Calcular a similaridade entre desenhos de acordes visando a escolha mais adequada na transição entre dois desenhos;

- Desenhar uma interface gráfica para captura de padrões rítmicos, voltada a violonistas;
- Definir um protocolo de comunicação entre agentes musicais baseado no MIDI;
- Desenvolver pacotes de software voltados à programação musical em Java;

## 1.4 Contribuições

### 1.4.1 Ciência da Computação

Em 1876 Alexandre Graham Bell marcou a união da tecnologia com o som através da invenção do telefone, mas foi com a invenção do gramofone que as possibilidades de armazenamento e alteração do som se estabeleceram.

Já no meio musical foi Thaddeus Cahil que, por volta de 1906, mostrou o primeiro instrumento musical que produzia som por meios elétricos. Pouco menos de uma década depois (1915), Lee De Forest inventou o que seria a base para geração de sons eletrônicos, o oscilador.

Há menos de meio século(1957), o primeiro programa de computador para música foi criado no laboratórios da Bell (Nova Jersey), por Max Mathews. O sucesso do Music I deu origem a uma série de programas musicais do mesmo criador. Em 1976, com a difusão dos microcomputadores e a utilização de linguagens de programação de alto nível, a computação musical se estabeleceu como uma promissora linha de pesquisa.

A computação musical é uma área multidisciplinar onde conhecimentos de diversas áreas são aplicados na música. A Inteligência Artificial tem tido um papel crucial na história da computação musical desde seu começo nas décadas de 50, sendo os maiores esforços direcionados aos sistemas de composição e improvisação, marginalizando sistemas para performance que usam técnicas de I.A. para capturar, não somente aspectos técnicos, mas também aspectos afetivos implícitos na música. Em uma performance no violão, a expressividade está diretamente relacionada com a mão direita do violonista agindo sobre aspectos rítmicos e de dinâmica. O software desenvolvido no contexto deste trabalho por si só já poderia ser considerado uma contribuição tanto para a computação como para a música, pois valida as teorias formalizadas nessa proposta através do uso de I.A em sua concepção.

Muitas linguagens vem sendo usadas na programação de aplicações musicais, entretanto nota-se um crescimento do Java entre as principais linguagens para este fim. Comparando Java com linguagens especialmente desenvolvidas de programação musical como MAX/MSP, SuperCollider e Nyquist e KeyKit, pode-se afirmar que, por ser uma linguagem aberta, Java pode combinar música com outras funcionalidades da linguagem, como rede, gráficos, banco de dados.O fato de Java estar sendo amplamente utilizada na programação de aplicações musicais não significa que é a única ou a melhor linguagem para este tipo de programação. Entretanto, os programadores parecem ter maior facilidade em expressar-se através do Java seja qual for o domínio de conhecimento, inclusive a música. Os fatores de sucesso do Java como linguagem de programação para software musicais são os mesmos que fizeram de Java uma linguagem bem sucedida em áreas mais tradicionais, dentre os quais cita-se a robustez, portabilidade, facilidade de aprendizado, bom suporte pela indústria, fácil depuração, bem projetada etc. Há, contudo, algumas características que fazem de Java uma das linguagens mais usadas para desenvolvimento de software musicais como, por exemplo,

o fato de trazer contigo uma biblioteca para tratamento de som e mensagens MIDI (Java Sound).

As restrições do Java Sound (biblioteca nativa para manipulação de som que acompanha o Java) não desestimularam os programadores de aplicações musicais que, ponderando as vantagens da linguagem, começaram a desenvolver novas bibliotecas mais eficientes e completas. Essas bibliotecas serão discutidas em maior detalhe no Capítulo 4 destinado a implementação do protótipo, entretanto adianta-se que nenhuma das bibliotecas é brasileira ou foca no reconhecimento de cifras e geração de acordes para instrumentos virtuais, um dos objetivos deste trabalho e outra contribuição para a computação.

### 1.4.2 Na música

A música vem se mostrando um campo muito promissor para auxiliar na descobertas de técnicas computacionais devido as suas características temporais e outros problemas provenientes de execuções musicais. Mas a utilização das soluções encontradas para os problemas musicais em outras áreas não é a única justificativa para a pesquisa da computação musical.

Há muitos séculos a música é usada no entretenimento, mas somente após sua união com a tecnologia é que sua difusão foi consideravelmente aumentada. Essa união é praticamente irreversível, visto a praticidade e qualidade que a tecnologia oferece aos profissionais da música, desde a concepção até a distribuição da mesma. A facilidade propiciada pela tecnologia democratiza a produção musical.

A maior contribuição para a música, deste trabalho, é a própria ferramenta, que pode auxiliar no ensino do uso do instrumento e aspectos rítmicos, bem como permitir que músicos que não toquem violão gravem uma parte de violão, em sua composição. A experimentação sonora através da ferramenta, pode também contribuir para que novas sonoridades ou estilos musicais sejam criados.

# 2 FUNDAMENTAÇÃO TEÓRICA

Considerando-se a multidisciplinaridade desta pesquisa, faz-se necessário uma breve explicação dos conceitos computacionais e musicais envolvidos em seu contexto. Entretanto os temas abordados já estão bem fundamentados na comunidade cientifica, logo, este texto tem apenas a finalidade de contextualizar o leitor.

## 2.1 Conceituação de Agentes

A abordagem de agentes tem sido usada para tratar problemas não convencionais onde elementos aparentemente distintos trabalhando em conjunto produzem um resultado complexo demais para poder ser programado facilmente de forma centralizada. Estes sistemas normalmente possuem características ditas inteligentes visto a imprevisibilidade de suas ações, uma vez que cada elemento possui seu papel individual e a habilidade de interação é comum a todos estes elementos (agentes).

Um agente é uma entidade real ou virtual, capaz de agir em um ambiente, de se comunicar com outros agentes, que é movida por um conjunto de inclinações (sejam objetivos individuais a atingir ou uma função de satisfação a otimizar); que possui recursos próprios; dispõe (eventualmente) de uma representação parcial deste ambiente; que possui competência e oferece serviços; que pode eventualmente se reproduzir e cujo comportamento tende a atingir seus objetivos utilizando as competências e os recursos que dispõe e levando em conta os resultados de suas funções de percepção e comunicação, bem como suas representações internas (REZENDE, 2003)(FERBE, 1991).

A definição acima é muito abrangente e genérica, tanto quanto sugeriu Russell e Norvig ao apresentar o conceito de agente como um paradigma integrador das técnicas de I.A (RUSSELL;NORVIG, 1995). Eles definem de maneira muito mais sucinta um agente como tudo o que pode ser considerado capaz de perceber seu ambiente através de sensores e de agir através de atuadores, podendo estes serem agentes humanos, robóticos e de software. Esta é a visão adotada no contexto desse trabalho.

Uma característica comum a esses sistemas é a utilização de uma metáfora de inteligência baseada no comportamento humano: como ele pensa, raciocina, toma decisões, planeja, percebe, se comunica e aprende.

### 2.1.1 Propriedades dos Agentes

Segundo Hunhs e Singh, alguns conceitos que caracterizam os agentes(HUHNS, 1998), são:

**Autonomia de decisão:** Capacidade de analisar uma situação, gerar alternativas de atuação e escolher a situação que melhor atende seus objetivos. Quando um agente se

baseia no conhecimento anterior de seu projetista e não em suas próprias percepções, dizemos que o agente não tem autonomia (RUSSELL, 2004).

Existem vários modelos de decisão que conferem ao agente a característica de autonomia. O mais conhecido é baseado na teoria da racionalidade limitada de agentes (SIMON, 1987), segundo o qual um agente sabe identificar as alternativas de solução, avaliá-las e ordená-las de acordo com um conjunto de critérios e, por fim, escolher a melhor (REZENDE, 2003). Esse comportamento é observado no Agente Mão Esquerda no ato da escolha do desenho do acorde, como será visto no Capítulo 3.

**Autonomia de Execução:** Capacidade de operar no ambiente sem intervenção de outro agente (geralmente humano). Esta capacidade está planejada para o Agente Solo e, principalmente, no Agente Mão Direita.

**Competência para decidir:** Capacidade de configurar sua atuação sem intervenção externa; Prevista para os Agentes Mão-Esquerda e Mão-Direita.

**Existência de agenda própria:** Lista de objetivos que concretizam suas metas. A agenda de objetivos, assim como as restrições, preferências e modo de atuação (aversão ou não ao risco) de um agente podem ser configuráveis. Prevista para o Agente Mão-Esquerda no ato do cálculo do desenho do acorde.

**Reatividade:** Capacidade de reagir às mudanças do ambiente a partir do reconhecimento de um contexto conhecido; Todos os agentes do sistema são reativos.

**Adaptabilidade:** Capacidade do agente de adaptar seu processo de decisão frente a situações desconhecidas; Os agentes do sistema não possuem essa característica.

**Mobilidade:** Capacidade do agente de mover-se e ser executados em outras plataformas; Possível devido às opções técnicas tomadas, porém não é necessário devido aos requisitos do sistema, logo os agentes do sistema não possuem essa característica.

**Personalidade:** Capacidade do agente de personificar-se, utilizando recursos que lembrem características humanas como a emoção ou o mau humor. Quantidade de dedos nas mãos, abertura máxima dos dedos e até quantidade de mãos são características humanas que podem ser configuradas no sistema. A inclusão da noção de emoção no Agente Mão-Direita é a meta final deste trabalho, previsto para trabalhos futuros.

**Interatividade com o usuário:** Capacidade de interagir com usuários e, considerando os possíveis mal-entendidos, reagir às falhas de comunicação de maneira aceitável. Presente nos Agentes Mão-esquerda, Mão-direta e Caixa de Som.

**Ambiente de atuação:** Caracteriza o local onde o agente vai atuar, isto é, em ambientes fechados (*desktop*) ou abertos (internet). Os agentes do sistema têm capacidade de atuar nos dois ambientes, porém os testes se detiveram no ambiente *Desktop*.

**Comunicabilidade:** Capacidade de interagir com outros agentes computacionais para a obtenção de suas metas. Os agentes do sistema se comunicam entre eles usando protocolo proprietário baseado no protocolo MIDI.

Além dessas características, os agentes podem ser classificados por sua cognição, pelo seu foco (similaridades físicas ou comportamentais com humanos) e por sua atuação (isolada ou social). (REZENDE, 2003).

### 2.1.2 Sistemas Multiagentes

Os sistemas multiagentes focam na resolução de um problema através de uma comunidade de agentes cooperando ou competindo (por vezes, os dois casos), entretanto, o que diferencia esta proposta de uma Resolução Distribuída de Problemas – RDP, é o fato do problema não ser conhecido previamente(REZENDE, 2003). Esses sistemas surgiram quando a metáfora da inteligência passa a ser o comportamento social, onde um conjunto de entidades inteligentes executa ações de modo coordenado no seio de uma sociedade objetivando um comportamento global coerente.

Existem questões sutis a serem observadas no ambiente de tarefas antes de se determinar se as tarefas devem ser feitas por um único agente ou por uma comunidade (RUSSELL, 2004). Muitas vezes, herdamos do mundo real a visão de uma única entidade realizando uma determinada tarefa e tomamos isto como a única forma de realizar esta atividade, como no exemplo deste trabalho, o violonista. Russel questiona e explica: *"Um agente A tem de tratar um objeto B como um agente ou ele pode ser tratado apenas como um objeto que tem um comportamento estocástico? A distinção fundamental é saber se o comportamento de B é ou não mais bem descrito como a maximização de uma medida de desempenho cujo valor depende do comportamento do agente A".*

Mas por que distribuir a inteligência ou criar uma inteligência coletiva? Ferber sugere várias respostas com um ponto em comum: lidar com a crescente complexidade dos sistemas computacionais (FERBER, 1999)

Um agente que atua isoladamente em um ambiente tem, em princípio, um controle total sobre os resultados de suas ações. Caso o agente, por outro lado, esteja imerso num ambiente onde co-existam outros agentes, certamente irá ocorrer uma interferência social entre estes agentes, onde processos de coordenação e negociação podem ser usados. (REZENDE, 2003).

Um grupo de agentes pode adotar ou ser submetido a diferentes formas de organização. Três níveis de organização foram descritas por Ferber (FERBER, 1999):

**Nível micro-social:** Interesse nas interações entre agentes e nas várias formas de conexões existentes entre um pequeno número de agentes. Este é o nível que concentra grande parte dos estudos da IA e também esse trabalho.

**Nível dos grupos:** interesse nas estruturas intermediárias usadas na composição de organizações mais complexas.

**Nível das populações ou sociedades globais:** Interesse na dinâmica de um grande número de agentes, estrutura geral do sistema e sua evolução. Pesquisa relacionada à vida artificial.

Essas organizações podem ser estáticas ou dinâmicas, se observadas através da ocorrência das interações sociais. Em alguns casos, padrões de interação que se repetem muitas vezes são capturadas em estruturas pré-estabelecidas, ou seja, estáticas. Seguindo este raciocínio, duas classes de modelos para as interações sociais são apresentadas (CONTE; CASTELFRANCHI, 1992):

**Modelos estáticos:** Os agentes têm um problema prévio a resolver. As interações sociais são limitadas por uma organização pré-existente, que guia os agentes para atingir o objetivo para qual o sistema foi concebido;

**Modelos dinâmicos:** Interações sociais e organizações são estabelecidas de forma que os agentes consigam atingir seus próprios objetivos. Este modelo pode ainda ser classificado como baseado na utilidade (o mundo social é um domínio da inferência dos agentes) e baseado na complementaridade (capacidade complementares entre os agentes).

Ainda sobre a organização dos agentes, Fox, Barbuceanu, Gruninger e Lin (FOX et al. 1998) afirmam que uma organização consiste de várias divisões e subdivisões, um conjunto de agentes alocados nestas divisões, um conjunto de papeis que os agentes assumem e um conjunto de metas. Os papéis são protótipos de funções a serem desempenhadas pelos agentes na organização.

O sistema multiagentes proposto atualmente não está distribuído em várias máquinas por uma questão relativa a facilidade no uso do mesmo por músicos, entretanto o problema é resolvido de forma distribuída entre os agentes e, se for desejável no futuro, podem ser fisicamente distribuídos.

## 2.2 Referencial Teórico de Música

As seções subseqüentes introduzem conceitos musicais diretamente relacionados a este trabalho, ou seja, uma breve descrição de notações musicais populares, cifragem de acordes e uma explicação sobre ritmo.

### 2.2.1 Notação Musical Popular

É comum encontrar nos principais meios de comunicação (web, revistas, tv) métodos que prometem ensinar violão em algumas poucas lições Sem entrar no mérito da eficácia destes métodos, é fato que são consumidos e que eles popularizam conceitos básicos da execução musical no violão. A notação musical adotada em muitos desses métodos é a tablatura.

A tablatura é uma forma de notação em que a nota a ser tocada (ou o acorde) é indicada pela posição dos dedos do executante no braço do instrumento. Originalmente era usada para alaúdes, mas ainda está em uso para violão e outros instrumentos trasteados. A Figura 2.1 mostra um exemplo de tablatura escrita para um instrumento trasteado com provavelmente com quatro cordas (quantidade de linhas horizontais) onde somente as notas referentes as cordas soltas (representada pelo número 0) e a terceira casa (número 3) são tocadas. Este tipo de tablatura é mais comumente encontrado fora do Brasil, em especial nos Estados Unidos.

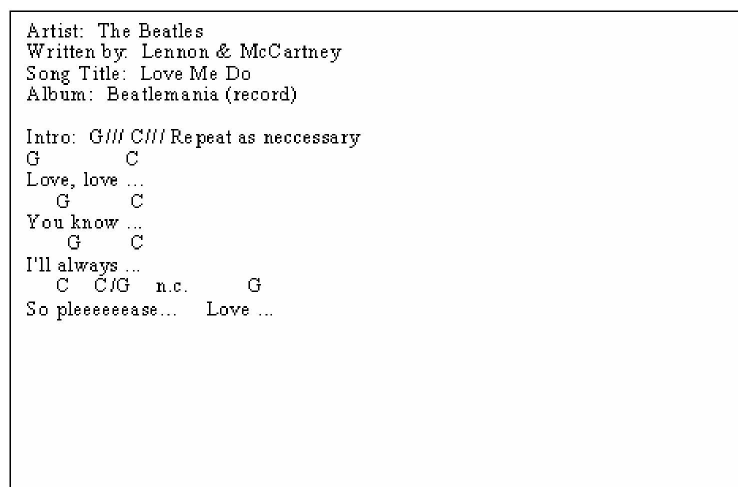
```
           Tuning : E A D G  Time Signature: 4/4


|0---|--0-|0---|--0-|0---|--0-|0---|0---|0---|--0-|0---|
|--0-|----|--0-|----|--0-|----|--0-|--0-|--0-|----|--0-|
|----|3---|----|3---|----|3---|----|----|----|3---|----|
|----|----|----|----|----|----|----|----|----|----|----|
```

Figura 2.1: Exemplo de tablatura em notação comum nos EUA.

Ainda na música popular, é usada a notação harmônica simples, na qual a seqüência harmônica é assinalada pelas abreviaturas dos nomes dos acordes (cifras).

Os diversos tipos de notação gráfica para a música contemporânea retornam ao caráter indeterminado das notações mais antigas, dando ao intérprete participação na própria elaboração da obra. Caminho inverso pode ser feito se o interprete insere anotações no registro da obra, evoluindo a notação e adaptando-a a sua própria cultura. Este fato explica a aceitável divergência entre notações (e seus símbolos).



```
Artist:  The Beatles
Written by:  Lennon & McCartney
Song Title:  Love Me Do
Album:  Beatlemania (record)

Intro:  G/// C/// Repeat as neccessary
G          C
Love, love ...
   G       C
You know ...
    G      C
I'll always ...
   C   C/G   n.c.       G
So pleeeeeease...    Love ...
```

Figura 2.2: Exemplo de registro musical popular normalmente encontrada no Brasil.

No Brasil, é comum se referir a letras de músicas cifradas como tablaturas, mesmo que erradamente. Nestes documentos aparecem por vezes as diagramações dos acordes. Um exemplo é visto na Figura 2.

### 2.2.2 Cifragem de Acordes

Cifras nada mais são que símbolos usados para representar acordes. Esses símbolos não são mundialmente padronizados, mas isto não chega a ser um grande problema para músicos mais experientes, o que não é verdade para novatos ou autodidatas. Basicamente os símbolos em uma cifra identificam a nota fundamental, os intervalos que devem ser adicionados ou evitados, alterações e inversões. No capítulo 4, destinado a implementação do sistema, descreve-se o funcionamento de um reconhecedor de cifras e conseqüente geração do acorde em dois pacotes de software destinados a programação musical, frutos desta pesquisa.

Ao utilizar notações baseadas na simples definição da seqüência harmônica, ou seja, aquelas que usam cifras para representar os acordes, uma ferramenta pode ser necessária, principalmente entre iniciantes: o dicionário de acordes. O dicionário de acordes mostra como um determinado acorde pode ser executado no violão (ou outro instrumento), ou seja, mostra o(s) desenho(s) do acorde, tal qual representado na Figura 2.3. É comum encontrarmos estes desenhos em notações populares que não somente registram as notas, mas também no método de produzi-las (ZAHAR, 1982).

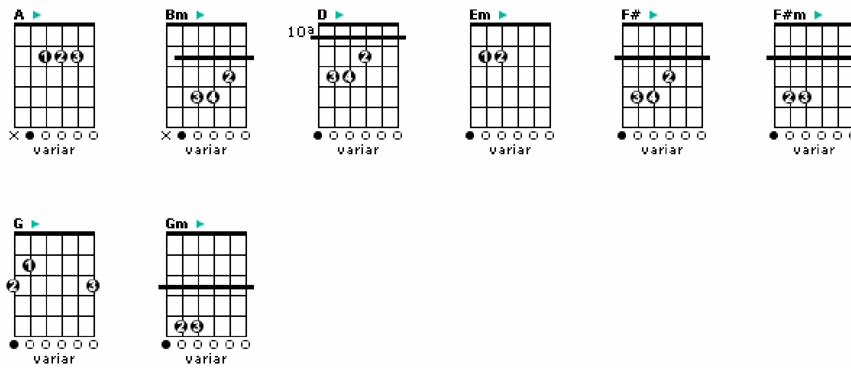



Figura 2.3: Diagramações de Acordes (desenhos dos acordes)

O desenho do acorde indica os trastes e as cordas que devem ser ponteadas pelo violonista, mas não a ordem em que as notas devem ser tocadas e nem o intervalo de tempo entre elas. Essas informações dependem do padrão rítmico (batida) adotado.Verifica-se então a primeira interdependência entre desenhos de acorde e padrões rítmicos, ou seja, mão esquerda e mão direita.

Apesar de possíveis diferenças na simbologia usada na cifragem, a diagramação dos acordes é uma informação relativamente estática, ou seja, é possível dizer que a cifra "C" representa um dó maior e é executado de uma determinada maneira no violão (desenho do acorde) dada sua afinação. Logo, toda vez que a cifra "C" aparecer, o aprendiz irá executar no violão a digitação referente ao acorde de Dó Maior que, muitas vezes, ele memoriza e utiliza em várias outras músicas.

### 2.2.3 Aspectos do Ritmo no Violão

Da mesma forma que o aprendiz repete o desenho do acorde aprendido em uma música em qualquer outra, ele repete a padrão rítmico em músicas que julgue ser similares. Entretanto, sem o acompanhamento de um tutor, é difícil o aluno ter a certeza de que o ritmo que ele está tentando executar está correto, pois a noção de tempo necessária no ritmo, muitas vezes, não está bem fundamentada no aprendiz, fazendo com que o ritmo soe mecanizado ou mesmo inapropriado. Esses blocos rítmicos que se repetem são popularmente conhecidos como batidas e sub-classificados em dedilhados[6] e rasgueados[7].Assumir que o ritmo está simplesmente relacionado com a mão direita do violonista é uma visão limitada da realidade, mas satisfatória em um primeiro momento.

---

[6] *Dedilhado: T*écnica de mão direita que usa as pontas dos dedos para tocar nas cordas em seqüência. Este tipo de técnica torna a sonoridade da música mais suave, pois os sons produzidos são percebidos individualmente e em seqüência.

[7] Rasgueado: Técnica de origem espanhola que e usa os dedos, no lado da unha, para bater nas cordas em um movimento rápido de abertura de mão (começando pelo dedo mínimo e finalizado com o dedo

Um aspecto é constantemente desconsiderado por software e métodos de ensino de violão: a ligação existente entre a digitação do acorde e o ritmo (batida) empregada. Ensinando o padrão rítmico e a digitação dos acordes de maneiras independentes, pode-se levar o aprendiz a cometer erros em uma execução musical como, por exemplo, tocar a nota de uma corda que ficou solta e não pertencente ao acorde. Isto ocorre principalmente porque o ser humano aprende tentando adaptar os esquemas mentais já existentes em uma situação similar, ou seja, se em uma determinada música ele aprende uma digitação de "**G7**" então, ele tende a repeti-la em outra música que tem outro padrão ritmo e pode não ser a digitação mais apropriada.

Conseguir representar o ritmo de uma forma amigável e de fácil compreensão ao usuário é o primeiro passo para permitir que o mesmo crie e experimente novos ritmos. Espera-se com o uso do computador nesta "representação rítmica" facilitar o entendimento e simulação sonora do ritmo.

No contexto deste trabalho, um padrão ritmo deve possuir as seguintes propriedades:

**Quantidade de notas desejadas na digitação dos acordes (polifonia):** Já na escolha dos acordes é possível visualizar a quantidade de notas simultâneas que devem ser executadas. Duplicações, dobramentos, supressões, triplicação etc. são artifícios utilizados para adequar um acorde às restrições de um determinado instrumento e de uso muito comum no caso do violão. Portanto, é necessário indicar com quantas notas simultâneas seu padrão rítmico vai trabalhar. No caso do violão de seis cordas, estes valores vão de um (melodia monofônica) até seis (total de cordas);

**Duração do padrão rítmico:** Dada pela quantidade de tempos e figura de tempo;

**Notas do acorde:** Notas classificadas pela altura considerando oitavas (mais grave para mais aguda): Como a linha harmônica é independente do padrão rítmico, não é possível saber as notas no momento da definição do padrão, logo, as alturas das notas indicam as cordas que deverão ser tocadas.

**Duração das notas:** Linhas horizontais de tempo (total dos compassos) são definidas para cada nota. No momento que o usuário julgar que a nota deva ser tocada, uma marca é feita na linha do tempo, sendo que o comprimento da marca indica duração da nota.

**Propriedades da execução da nota:** Em cada nota, determina-se as propriedades de execução da mesma, tais como: intensidade, balanço, direção do arpejo e outros efeitos;

---

indicador) usando um dedo por vez para tocar todas as cordas seqüencialmente. Neste tipo de técnica, as unhas do violonista servem para garantir um ataque preciso.

### 2.2.4 Definindo Ritmo

Ritmo é a forma pela qual as diferentes durações dos sons e pausas (mais breves ou mais longas) se distribuem no tempo. Podemos criar combinações rítmicas variadas, simples ou de extrema complexidade (TOM DA MATA, 2005).

As durações das notas e das pausas são representadas pelos valores rítmicos ou figuras: da semibreve (de maior duração) à semifusa (menor duração).

A Figura 2.5 mostra as figuras de tempo e seu valor comparativo, por exemplo, uma semínima vale metade de uma mínima (TOM DA MATA, 2005).

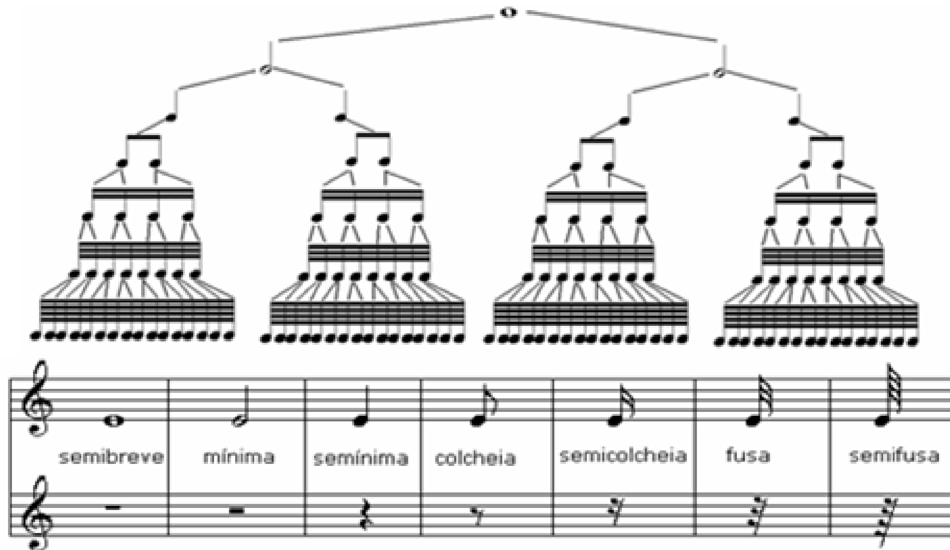



Figura 2.4: Figuras de tempo usada pela notação musical tradcional.

O estudo do ritmo é feito por todos os tipos de músicos, porém neste grupo destacam-se os percursionistas e bateristas devido às características de seus instrumentos. A figura 8 mostra um exemplo de partitura escrita para esses músicos onde somente informações rítmicas são relevantes (BATERA, 2005).



*Figura 2.5: Partitura de bateria*

Não existe uma notação clássica separada para bateristas, entretanto criou-se uma notação baseada na notação clássica. A Figura 2.6 mostra o que significam os símbolos usados na partitura (Figura 8).

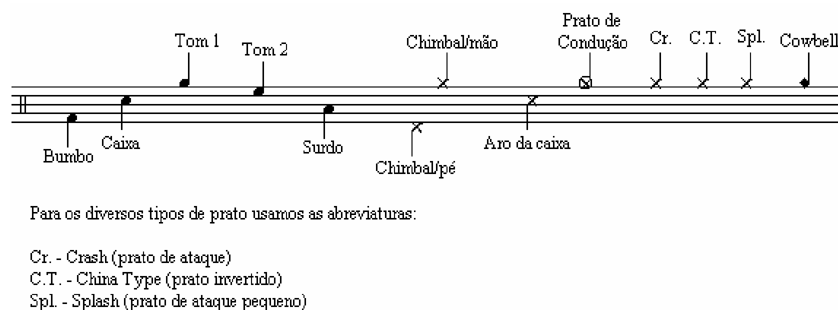



Figura 2.6: Notação para partituras de bateria

A seguir serão apresentados alguns termos e explicações relacionadas ao conceito de ritmo (TOM DA MATA, 2005)**:**

**Pulso**

As combinações rítmicas (células rítmicas) de uma música têm sempre como referência, explícita ou implícita, uma pulsação rítmica constante e regular. Esta sucessão de pulsações regulares são os tempos.

**Compasso – métrica**

Esta série de pulsações, isto é, de tempos de mesma duração, podem ser agrupados dois a dois, três a três, quatro a quatro. Estes são os agrupamentos mais comuns na música ocidental seja popular, folclórica, ou na chamada música clássica. Entretanto, em outras culturas, como nas músicas tradicionais do Oriente e da África, são comuns outros agrupamentos. Estes agrupamentos de tempos são o que chamamos de compassos. O primeiro tempo é sempre um pouco mais forte ou mais acentuado.

Quando os compassos são agrupados de dois em dois tempos, dizemos que a música está no compasso binário. De três em três, ternário. De quatro em quatro, quaternário.Por exemplo: marcha, baião, tango e samba são binários. A maior parte das cantigas de roda também. Já o compasso da valsa é ternário.

**Representação dos Compassos**

Logo no início da música, na primeira pauta, encontramos dois números sobrepostos. O numerador indica quantos tempos teremos em cada compasso. O denominador indica qual a figura, o valor rítmico adotado que vale um tempo, isto é, qual dos valores (apresentados na Figura 4) equivalerá a um tempo naquela música. A isto chamamos unidade de tempo. Tendo a unidade de tempo, podemos deduzir a unidade de compasso, isto é, qual a figura que, sozinha, demorará (soará) durante todos os tempos do compasso.

**Divisão dos Compassos e Subdivisão dos Tempos**

Numa música cada pulsação, ou tempo, tem exatamente a mesma duração. Se num tempo temos mais de uma nota, a duração deste tempo terá que ser subdividida entre

estas notas. Também podemos ter notas que duram mais de um tempo ou mesmo que duram uma quantidade de tempos que supera a quantidade de tempos do compasso. Para aumentar o valor de uma nota usamos:

**Ponto de Aumento**

Um ponto colocado à direita da nota e que equivale à metade do valor da nota.

**Ligadura**

Uma linha curva que liga duas notas seguidas na mesma altura, ou seja, duas notas que têm o mesmo som e, portanto, ficam no mesmo lugar na pauta. Neste caso, somente a primeira é emitida respeitando-se o seu valor rítmico e prolongando-se o seu som durante o tempo do valor rítmico seguinte. Desta forma, a nota deve soar pela duração dos valores somados.

**Deslocando as acentuações naturais**

Muitas vezes nas músicas temos um efeito de deslocamento da acentuação natural, ou seja, o tempo forte, primeiro tempo, é preenchido por pausa (silêncio) ou então temos um prolongamento do som anterior. Convém lembrar que todo tempo tem uma parte forte e outra fraca. A parte forte de um tempo é exatamente o momento em que a marcação do tempo é feita. O resto da duração do tempo constitui a parte fraca. Portanto, este deslocamento pode ser feito em qualquer um dos tempos do compasso.

**Síncope**

Quando uma nota é executada em tempo fraco ou parte fraca de tempo e se prolonga ao tempo forte ou parte forte do tempo seguinte. A síncope é regular quando as notas que a formam têm a mesma duração. É chamada de irregular quando suas notas têm durações diferentes.

**Contratempo**

Quando a nota soa em tempo fraco, ou parte fraca de tempo, sendo antecedida, isto é, tendo no tempo forte ou na parte forte do tempo, uma pausa.

Tanto a síncope quanto o contratempo produzem um efeito de deslocamento das acentuações naturais. São muito utilizadas na nossa música, como no samba e bossa nova.

**Andamento**

Andamento é o que vai determinar a velocidade do pulso, dos tempos. Algumas vezes, dois intérpretes gravam a mesma música e um canta um pouco mais lento e o outro, um pouco mais rápido. Assim, o ritmo e o andamento são os componentes básicos da marcação.

O metrônomo, aparelho inventado por Loulié em 1710, regula a quantidade de pulsos por minuto (bpm) ajudando a precisar a duração exata dos tempos.No decorrer de uma música o andamento pode ser alterado em algum trecho para dar mais expressão.

## 2.3 Resumo do Capítulo

Este Capítulo procurou introduzir conceitos que serão necessários para a compreensão do trabalho proposto. Referente a computação foram apresentados conceitos vinculados a resolução de problemas por uma comunidade de agentes e suas propriedades. Já no

campo musical, foram apresentadas de forma sucinta algumas notações musicais alternativas, cifragem de acordes e conceitos de ritmo..

# 3 CONCEPÇÃO DO SISTEMA

Neste capítulo trataremos do processo de desenvolvimento da aplicação e as idéias que nortearam seu desenvolvimento, dando foco para nos requisitos, análise preliminar e algumas decisões de projeto. Muito dos requisitos identificados são passíveis de reutilização, portanto optou-se por criar bibliotecas que pudessem ser utilizadas separadamente deste sistema. Ainda, teremos no Capítulo 4 apresentaremos o protótipo e seus resultados.O paradigma adotado foi o orientado a objetos com modelo de ciclo de vida em espiral.

Para a concepção idealização e desenvolvimento do software objeto deste trabalho, foram seguidas as seguintes etapas:

**Identificação dos requisitos desejáveis:** Identificação das funcionalidades desejáveis do software. É bom salientar que nem todos os requisitos foram necessariamente construídos em função de decisões de projeto e/ou restrições técnicas;

**Análise do sistema:** Construção de modelos que visam o esclarecimento e completo entendimento do domínio do problema em estudo.

**Projeto:** Adaptação dos modelos da análise sob a ótica da tecnologia selecionada para a implementação do protótipo, ajustando as necessidades levantadas a questões técnicas.

**Implementação/Codificação:** Tradução dos modelos de projeto para a linguagem de programação definida também em projeto. Testes individuais também são previstos nesta fase.

**Testes e avaliações:** Testes de integração e avaliação do protótipo.

Neste Capítulo daremos foco para nos requisitos, análise preliminar e algumas decisões de projeto. Muito dos requisitos identificados são passíveis de reutilização, portanto optou-se por criar bibliotecas que pudessem ser utilizadas separadamente deste sistema. Ainda, teremos no Capítulo 4 apresentaremos o protótipo e seus resultados.O paradigma adotado foi o orientado a objetos com modelo de ciclo de vida em espiral.

## *3.1* Requisitos Identificados

O requisito básico que norteia o desenvolvimento desta aplicação é a separação dos elementos musicais (harmonia, melodia e ritmo) em uma execução no violão. É comum haver uma separação destes elementos musicais buscando a simplicidade do entendimento da música e dos software musicais. (WEST et. al. 1991)

### 3.1.1 Definição da harmonia

Cabe ao usuário definir a seqüência dos acordes que compõe a linha harmônica de sua música. Ainda é possível que várias harmonias sobreponham-se na mesma composição.

A definição da seqüência harmônica é feita através das cifras, onde o usuário entra, seqüencialmente, a cifra referente ao acorde. Blocos com uma seqüência de acordes denominados *Partes* podem ser criados possibilitando ao usuário a inclusão da mesma seqüência harmônica em vários locais da composição. Exemplo: supondo as partes A e B, a composição poderia ser A + B + A ou A + A + B;

Não existe nenhuma referencia ao tempo na definição da harmonia, somente uma seqüência simples de acordes.

### 3.1.2 Personalização dos Símbolos da Cifragem

Como não estão mundialmente padronizados, certos símbolos usados na cifragem podem divergir em função da origem da música. Logo, deve ser possível adaptar os símbolos a cultura do usuário.

### 3.1.3 Reconhecimento das Cifras e Cálculo do Acorde

Uma vez definida a notação de cifragem, o sistema deve ser capaz de reconhecer as cifras e gerar um acorde resultante, caso a cifra seja válida. Caso existam problemas com a cifra, o sistema deve explicar a causa do erro e sugerir correções (não implementado nessa versão).

O acorde calculado deve possuir a descrição do papel (intervalo) de cada nota que compõe o mesmo.

### 3.1.4 Cálculo do Desenho do Acorde para Instrumentos Virtuais Configuráveis

O desenho do acorde é uma representação de como é executado o acorde em um determinado instrumento, portanto para que o desenho do acorde possa ser calculado o instrumento e afinação do mesmo devem ser definidos.

E desejável a visualização do desenho do acorde em uma interface gráfica similar ao instrumento real.

O sistema em questão tem como instrumento base o violão de seis cordas com a afinação padrão (Mi(6)- Lá – Ré – Sol – Si – Mi(1)), mas tanto o instrumento quanto sua afinação podem ser modificados.

### 3.1.5 Definição da Melodia

Da mesma forma que na linha harmônica, o usuário pode determinar a linha melódica de toda a música ou simplesmente um solo. Ele pode fazer isto importando um arquivo musical (MIDI) ou escrevendo as notas diretamente na partitura (não implementado nessa versão).

### 3.1.6 Definição do ritmo

Como já mencionado anteriormente, o ritmo deve ser definido independente da harmonia. Talvez isso não seja um pensamento natural para músicos que estão acostumados a fazê-lo de forma unificada. De qualquer forma, o ritmo é o principal dado a ser tratado pelo sistema e portando mecanismos de captura destes dados devem

ser oferecidos ao usuário. Dois dispositivos foram pensados: interface de comunicação com instrumentos MIDI (não implementado nessa versão) e uma interface gráfica para usuários sem o instrumento.

Blocos com informações rítmicas denominados padrões rítmicos devem ser vinculados à informação harmônica para que a música possa soar. Cada estilo de música tem um ou mais padrões, e estes, organizados em bibliotecas de ritmo, podem ser usados para compor um padrão rítmico mais complexo (não implementado nessa versão).

Arquivos MIDI podem ser usados como fonte para extração de padrões rítmicos.

### 3.1.7 Controle da execução da música;

Controlar o andamento é permitir ao usuário iniciar, parar, paralisar, avançar e retroceder na música (implementado parcialmente nessa versão).

### 3.1.8 Controlar volume da música (*mixado*);

A música que é ouvida pelo usuário é uma mistura de vários padrões rítmicos utilizando como base a(s) linha(s) harmônica(s) e a(s) linha(s) melódica(s).

Este requisito permite amplificar ou atenuar gradualmente o volume de todos estes canais em conjunto.

### 3.1.9 Possibilitar escolha do sintetizador usado para geração das notas;

A qualidade do som gerada pelo sistema está vinculada a alguns componentes de hardware e software, como por exemplo, a placa de som, o sintetizador, a própria arquitetura do computador(Mac/PC) e seu Sistema Operacional (Win/Linux/MacOS). Dentre estes, destaca-se o sintetizador.

O sintetizador é um equipamento que permite gerar ou modificar características de um som com o intuito de criar novas sonoridades ou até mesmo, através de determinadas técnicas, recriar um som acústico. Já existem sintetizadores virtuais, ou seja, software que simulam o hardware.

Este requisito permite que o usuário escolha qual sintetizador quer usar para gerar os sons.

### 3.1.10 Ajuste individual dos controles de determinado instrumento;

Cada padrão rítmico, bem como a linha melódica, pode ser representado por um instrumento ou um tipo de violão que proporciona ao ouvinte uma sonoridade mais próxima da realidade.

Cada instrumento usado nesta composição deve estar em um canal independente, ou seja, devemos ser capazes de controlar individualmente cada uma dos seguintes atributos:

- Instrumento: Deve ser possível alterar o instrumento que gera uma determinada sonoridade (timbre) de um padrão rítmico ou linha melódica;
- Efeitos: Controle no volume, *stereo (pan),* eco (*delay*), reverberação (*reverb*) e outros relativos a cada sintetizador proporcionam a possibilidade de aprimorar a sonoridade ao ponto exato (ou próximo) que o compositor definiu (não implementado nessa versão).

- Controle de solo, mudo (*mute*) e desligamento de um “canal”;

### 3.1.11 Persistir dados para futuro uso (inclusive MIDI);

Visando a continuação da composição previamente iniciada sem a perda do que já foi feito, deve-se disponibilizar ao usuário a opção de salvar: os padrões rítmicos, a seqüência harmônica e melódica, a música em formato MIDI e todos os itens juntos em um único projeto.

## 3.2 Abordagem Multiagente

O desenvolvimento de sistemas interativos musicais acompanhou a evolução das técnicas de Inteligência Artificial e Engenharia de Software. Nas décadas de 60 e 70 os sistemas eram freqüentemente implementados através de algoritmos iterativos (CONGER, 1988). Recentemente, a comunidade começou a usar em aplicações musicais técnicas como: sistemas multiagente, vida artificial e algoritmos genéticos (BILOTTA et al. 2000).

A idéia básica deste trabalho é desenvolver uma comunidade de agentes que represente as entidades envolvidas em uma performance musical de violão. A organização do sistema baseia-se em um modelo dinâmico, ou seja, apesar de ter um problema prévio a ser resolvido, a decisões dos agentes só podem ser tomadas imediatamente antes da execução baseada em uma série de parâmetros.

Quatro agentes foram identificados e modeladas no contexto desse trabalho. São agentes reativos, com autonomia de decisão (uma vez os dados iniciais fornecidos), com agenda própria que pode ser configurada, comunicativos, imóveis, interativos e que atuam tanto em ambientes fechados como abertos. São eles:

**Mão Esquerda (ME):** Responsável pela execução dos acordes, ou seja, deve possuir o conhecimento de formação de acordes dada a afinação do instrumento bem como a interpretação das cifras que representam os acordes. Ainda, é responsabilidade desta entidade escolher qual o melhor desenho de acorde baseado na habilidade e restrições do usuário.

**Agente Mão Direita (MD):** Responsável pelo ritmo impresso na música e muito da expressividade do músico, por isso é comum omitir informações rítmicas nas notações musicais populares; Isto, ao mesmo tempo em que flexibiliza a execução, confunde os iniciantes. As informações de um padrão rítmico basicamente consistem em dizer quais cordas que serão tocadas em um determinado momento. Esta informação em conjunto com as notas definidas na linha harmônica do Agente de Mão Esquerda, indica as mensagens MIDI que irão propiciar a geração do som pelas “Caixas de Som”.

**Agente Caixa de Som (CS):** Permite que os usuários simplesmente escutem a composição, sem nenhuma (ou muito pouca) interferência na composição. Configurações de síntese, controle de andamento, volume, transposição de tom e outros parâmetros relativos à execução e sonoridade da música podem ser configurados.

Existe ainda um quarto agente que pode aparecer nesta interação, o Solista. Este é responsável por reproduzir uma melodia, em qualquer que seja o instrumento. No nosso escopo de trabalho é um agente que não influi diretamente nos resultado e, portanto, não será descrito em detalhes.

A seguir descreveremos com maiores detalhes a arquitetura do sistema, os agentes, seus papéis, responsabilidades e interações.

### 3.2.1 Arquitetura

Na arquitetura proposta, os agentes se situam no contexto de uma composição, ou seja, para cada nova composição um novo grupo de agentes é instanciado. Para se obter um resultado sonoro, é necessário ao menos um Agente ME, um MD. Opcionalmente um CS se for desejado trabalhar os parâmetros da execução.

A distribuição dos agentes pode se dar em diferentes máquinas. Entretanto, neste tipo de ferramenta, onde a música objetiva gerar material musical rapidamente e da forma mais prática possível, optou-se por deixá-los rodando em uma única máquina ainda que o protocolo de comunicação e aspectos técnicos possibilitem uma separação em um estágio futuro.

A comunicação entre os Agentes ME e MD é bidirecional e pode ser múltipla, ou seja, um agente ME pode conversar com diversos agentes MD, bem como um agente MD pode conversar com diversos agentes ME. Não foram modeladas interações entre agentes do mesmo tipo. Os Agentes CS interagem como todos os outros tipos de agentes da mesma composição. A Figura 10 mostra um exemplo de interação entre os diversos tipos de agentes no contexto de uma composição. Neste exemplo, a harmonia estabelecida no Agente ME é executada por dois ritmos sobrepostos, cada um representado por um Agente MD. A caixa de som (CS) reproduz a sonoridade obtida desta interação.

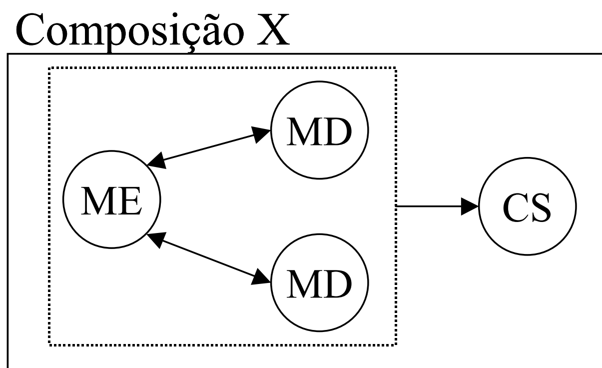



Figura 3.1: Arquitetura de uma composição.

O fluxo das mensagens inicia no agente ME quando ele carrega as informações harmônicas e tenta reconhecer os símbolos dos acordes. O processo de escolha do melhor desenho de acorde é baseado na facilidade de execução, similaridade com o acorde anterior, características do instrumento (afinação), características do instrumentista e também informações rítmicas.

Para que uma harmonia fique completa e apta a execução é necessário que seja associada a ela um ou mais padrão(oes) rítmico(s). Esse processo é feito pelo no Agente MD. Feito isso, é possível solicitar a execução da música, como mostra a Figura 3.2.

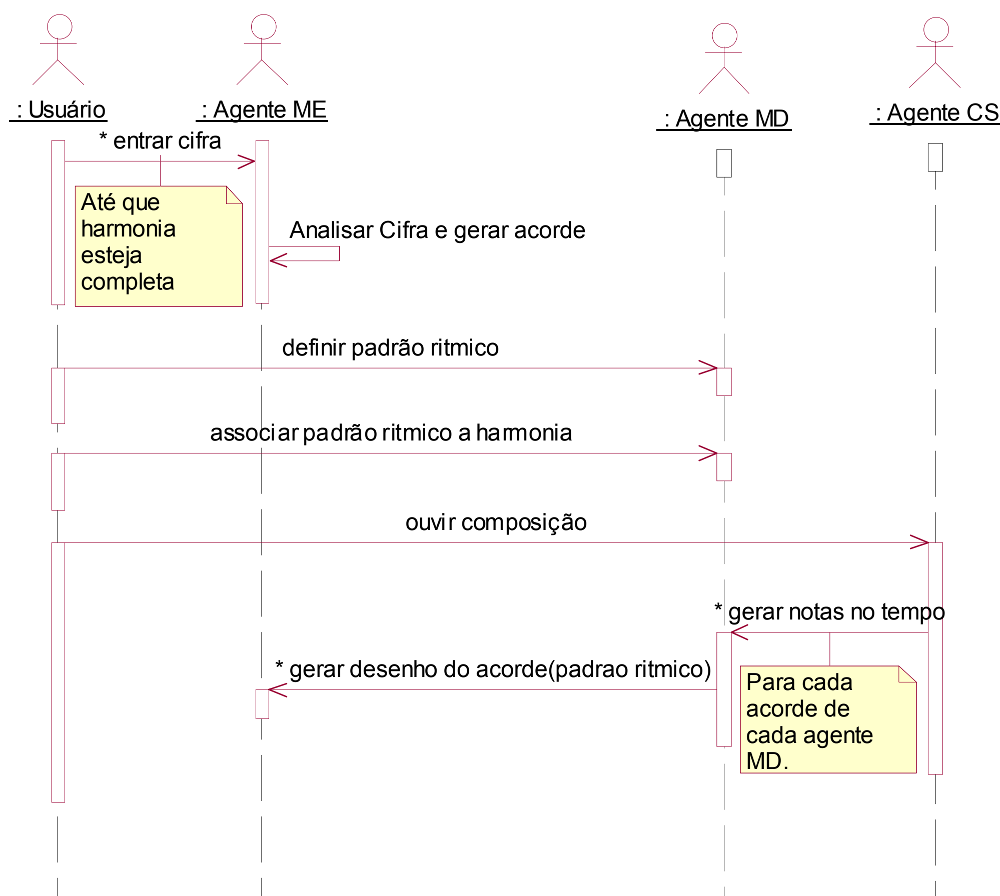



Figura 3.2: Diagrama de Interação entre os agentes do sistema.

A diagrama de interações representado na Figura 3.2, baseado na UML, mostra o usuário e os Agentes como atores capazes de prestar serviços mediante as requisições (flechas indicando a troca de mensagens). Esta figura representa somente o fluxo normal do caso de uso “tocar composição”.

### 3.2.2 Agente Mão Esquerda (ME)

Apesar do foco do trabalho proposta estar no aspecto rítmico (Agente MD), o Agente ME agrega algumas das tarefas mais trabalhosas e cognitivas em suas responsabilidades. Para ajudar a programação destas tarefas, optou-se por criar pacotes de software que pudessem ser utilizados tanto pelo agente ME, como também por outros trabalhos futuros do grupo. Estes pacotes serão visto no Capítulo 4.

As principais tarefas do Agente ME são:

- Reconhecimento e validação das cifras do acorde baseado em uma notação que possa ser personalizada;
- Cálculo do acorde informando ao usuário as notas e intervalos que compõem aquele acorde;
- Cálculo do desenho (ou representação) do acorde levando-se em conta propriedades do instrumento virtual utilizado e características do instrumentista;

  - Propriedades dos instrumentos virtuais: Quantidade de cordas, afinação, quantidade de trastes;
  - Propriedades do instrumentista: Quantidade de dedos da mão esquerda, quantidade de dedos da mão direita, abertura máxima de dedos (medida em trastes).
- Escolha do melhor desenho de acorde considerando-se: facilidade na transição do desenho de acorde anterior, facilidade na execução do desenho de acorde (quantidade de dedos, proximidade da cabeça do violão, abertura de dedos, uso de pestana), e o padrão rítmico. O primeiro desenho de acorde é escolhido somente pela proximidade em relação cabeça do violão, garantindo que soe em uma região mais agradável.
- A escolha do melhor desenho do acorde será detalhado na descrição do pacote de "Formação de Acordes", no próximo Capítulo. Ressalta-se, entretanto, que o algoritmo padrão não considera o padrão rítmico como uma variável e precisou ser estendido para tal. Fundamentalmente dois aspectos do padrão rítmico são considerados:
- Polifonia: Quantidade de notas polifônicas do padrão rítmico (definição obrigatória em dedilhados);
- Inexistência de cordas soltas em arpejos: Quanto o número de notas simultâneas (início no mesmo tempo) supera o número de dedos da mão direita, conclui-se que as cordas não poderão ser *pinsadas*, logo um arpejo rápido (às vezes com palheta) é necessário. Nestes casos as cordas do desenho do acorde precisam ser contíguas, pois não daria tempo de pular alguma corda. Ainda, a polifonia é opcional nestas situações.

O diagrama da Figura 3.3 mostra como o Agente ME obtém as informações necessárias para efetuar o processamento das funcionalidades acima citadas.

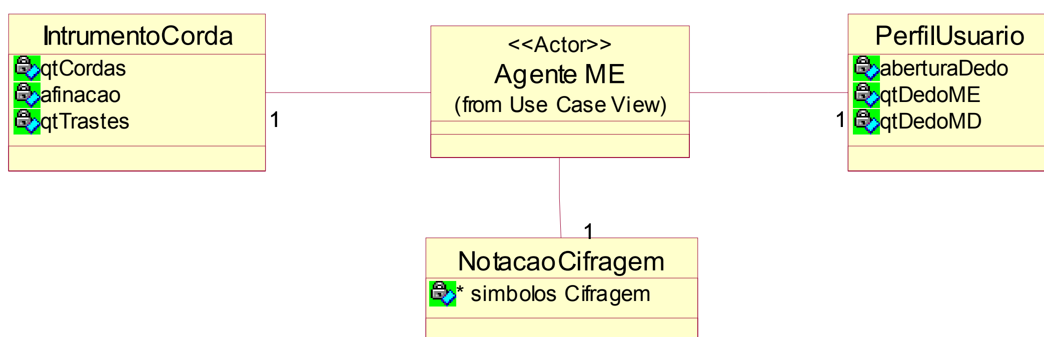



Figura 3.3: Dependências do Agente ME.

Somente informações harmônicas são tratadas por pelo Agente ME. A harmonia definida não deve ter nenhuma informação de tempo, pois se considera esta propriedade referente ao ritmo, portanto, tratado no agente MD. Logo, formula de compasso, figura de tempo, e o próprio compasso são informações desprezadas por este agente.

Além dos acordes, o usuário pode querer registrar algumas técnicas e efeitos próprios da mão esquerda. Ainda que não implementado no protótipo, são elas:

**Bend:** Alteração proposital na afinação, obtida ao se puxar à corda de um violão para cima. Apesar de ser mais comum na melodia (Agentes Solistas) também pode ser usado na harmonia.

**Vibrato:** Consiste em movimentos rápidos dos dedos da mão esquerda sobre a corda criando uma variação da altura da nota. A intensidade e velocidade podem ser controladas.

**Hammer-Ons**: Ligado ascendente; técnica do violão e da guitarra que consiste em tocar uma nota de maneira convencional e a nota mais aguda seguinte apenas com a mão esquerda, 'martelando' a corda contra a escala.

**Pull-Offs**: Ligado descendente; técnica do violão e da guitarra que consiste em tocar uma nota de maneira convencional e a nota mais grave seguinte apenas com a mão esquerda.

**Trill:** Rápida alternância entre duas notas usando sucessivos *hammer-ons e pull-offs*.

**Harmônicos:** Som gerado pela oscilação de corpo vibrante em freqüências múltiplas (x2, x3, x4, etc) da freqüência fundamental. No violão os harmônicos podem ser gerados de várias maneiras, a mais comum é com toque suave com o dedo da mão esquerda na corda em uma posição equivalente a uma divisão inteira da corda (1/2, 1/4/1/8 etc). Existem vário tipos: natural, artificial, *tapped*, *pitch* etc.

**Slide:** É o deslizar o(s) dedo(s) da mão esquerda sobre os trastes da guitarra/violão enquanto a corda está vibrando. Possui alguns tipos:

**Legato Slide** –Toque a primeira nota e deslize para a segunda que não é tocada.

**Shift Slide** - Toque a primeira nota e deslize para a segunda que é tocada.

**Slide Into** - Legato slide de uma nota começando de um ponto pré-determinado e parando na nota indicada.

**Slide Out Of** - Legato slide de uma nota terminando em um ponto pré-determinado.

### 3.2.3 Agente Mão Direita (MD)

Permite a definição de um padrão rítmico para violão através de uma interface especialmente desenvolvida para isso.Após obter notas com o agente ME, as envia para o agente CS com as informações de tempo pertinentes. Além da interface, foi modelada a captura dos padrões rítmicos através da extração dos mesmos de arquivos MIDI ou diretamente de um instrumento MIDI.

Teoricamente, um agente MD pode estar vinculado a diversos ME. Restringimos no protótipo que um Agente MD deve estar vinculado a somente um agente ME. O agente MD está ligado diretamente ao agente CS em uma cardinalidade de N para N.

É responsabilidade do Agente MD:

- Permitir a definição de padrões rítmicos para violão, considerando:
    - Polifonia do padrão: Quantidade de cordas necessárias para executar o padrão, conseqüentemente, quantidade de vozes do acorde.
    - Quantidade de tempos: Quantidade de tempos de todo o padrão.
    - Figura de Tempo: Duração de cada um dos tempos do padrão.

- Associação do padrão rítmico à harmonia do agente ME ao qual está vinculado.
- Identificar notas com ataque junto para propiciar tratamento diferenciado, como a velocidade de arpejo, por exemplo.
- Permitir audição da composição com o ritmo definido;

Abaixo cita-se características, técnicas e efeitos que podem ser descritos como responsabilidade da mão direita. Somente os 4 primeiros foram implementados nesta versão do protótipo.

**Direção do Arpejo:** Ascendente ou descente. Indica se as cordas começarão a ser excitadas de baixo pra cima (arpejo descendente) ou de cima pra baixa (ascendente).

***Delay* do Arpejo:** Tempo total dado em função do somatório do tempo entre cada nota do acorde em arpejo. Quanto maior, mais lento o arpejo.

**Swing (balanço):** Variação pra mais ou menos de até 50% do valor da figura de tempo em relação ao ataque na nota. O objetivo é tornar o padrão mais humano e menos mecânico.

**Dinâmica:** Determina a amplitude da nota. Útil para determinar a acentuação do compasso ou o tempo forte.

**Palm Mute:** Abafa o som da nota com um leve tocar nas cordas, próximo a ponte do violão, com a palma da mão direita (destros). Essa ação corta o som das cordas que estão vibrando criando um efeito percusivo.

**Pickstrokes:** Toca a nota da direção indicada, ou seja, de cima para baixo ou de baixo para cima; Sem muito efeito sonoro, mas pode ser útil para fins didáticos.

**Tap (tapping):** Batidas nos trastes do instrumento com os dedos da mão direita, geralmente o indicador ou o médio.

**Tremolo Picking:** Consiste em tocar a mesma nota várias vezes e muito rápido, soando com se tivesse tocando uma única vez (corte na amplitude).

**Staccato:** Indica uma nota de duração muito curta, não relacionada a figura de ritmo da partitura.

Optou-se por não utilizar a noção de compasso visando uma simplicidade para usuários iniciantes ou leigos de teoria musical, portanto o padrão rítmico se repete para cada acorde da harmonia, podendo ser criados tantos padrões quanto necessário.

O Agente MD atualmente é reativo. Sua proposta, entretanto, projeta no futuro a capacidade de dotá-lo com inteligência para que o mesmo sugira padrões rítmicos baseados no estilo musical e até mesmas características emocionais. Estas idéias são detalhadas nos trabalhos futuros.

### 3.2.4 Agente Solista (SL)

Este agente não estabelece comunicação com os agentes Mão-Esquerda (ME) e Mão-Direita (MD). Foi projetado somente para ler arquivos MIDI e enviar notas para o Agente Caixa de Som (CS). O arquivo MIDI deve conter somente informações melódicas e não necessariamente soarão com timbre de guitarra. Na verdade, não passa de um tocador MIDI que envia as mensagens musicais para o Agente Caixa de Som.

No futuro, este agente irá representar o agente humano interagindo com o sistema através de um instrumento MIDI, por isso a importância de defini-lo neste momento. Ainda, é possível que a melodia influa na decisão dos desenhos de acorde, buscando inversões em que a nota fundamental esteja presente na melodia.

### 3.2.5 Agente Caixa de Som

Sintetiza e *mixa* as notas geradas pelos agentes ME, MD e SL. É usado quando se deseja ouvir a música fazendo pequenas alterações nos agentes, tais como volume, timbre, mudo, solo. Na caixa de som, todos os agentes vão tocar juntos e em sincronia.

É o único agente que possui uma interface gráfica e roda sobre o computador cliente. Todos os outros agentes são configurados através de sua interface e podem rodar em outras máquinas.

São tarefas do Agente CS:

- Servir de interface para configuração dos outros agentes e criação da composição;
- Controlar a execução da música;
- Mostrar a composição em uma notação gráfica;
- Exportar a composição para arquivo MIDI;
- Permitir o controle dos parâmetros de execução de cada agente da composição;

Pode-se ter diversos agentes CS na composição e com isso experimentar diversas configurações dos parâmetros de execução.

## 3.3 Esquema de comunicação

Usando uma rede TCP/IP como meio de propagação, os agentes recebem mensagens que informam além do remetente e o destinatário, o evento MIDI a ser executado no tempo indicado.

A decisão de se criar um protocolo proprietário de comunicação advém da necessidade de velocidade na comunicação prevendo futuro uso em tempo real da aplicação, como um sistema de performance. Aproveitando o protocolo padrão de comunicação entre instrumentos digitais – MIDI, criou-se uma camada superior que encapsula a mensagem MIDI.

**Formato das mensagens:** (<Agente Remetente>, <Agente Destinatário>, <Mensagem MIDI>, <Tempo de Execução>, <Identificador Seqüencial>).

Exemplo de mensagem MIDI: *ShortMessage(Note On/Canal, Nota, Velocidade).*

O tempo de execução determina quando a mensagem vai ser enviada ao dispositivo MIDI. É contado em milissegundos a partir da mensagem anterior, ou seja, é o intervalo entre a mensagem anterior e a atual. A primeira mensagem tem o tempo de execução igual a zero, pois é enviada assim que o usuário solicita a execução da música.

O problema de latência é tratado adicionando-se um buffer de mensagens cujo tamanho varia em função da velocidade média da transmissão, como pode ser visto na Figura 13.

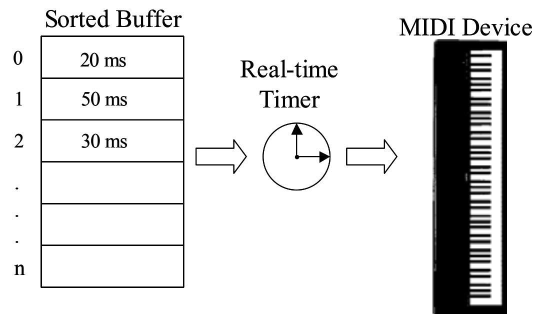



Figura 3.4: Dispositivo para tratar a latência das mensagens MIDI

O timer retira a mensagem da fila (identificador seqüencial) e programa a si mesmo para fazê-lo novamente no tempo especificado da próxima mensagem.

Veremos no Capítulo destinado a implementação que por questões técnicas e funcionais, não foi necessária à implementação do sistema descentralizado, logo o tratador de latência, apesar de implementado, não está sendo utilizado. Tal fato também influenciou na escolha do método de troca de mensagens entre os agentes, sendo escolhido *Sockets* sobre TCP/IP.

## 3.4 Escolha das tecnologias envolvidas na construção

### 3.4.1 Linguagem de Programação

Por que Java é bom para música? Esta pergunta foi feita aos pesquisadores e programadores que atuam na área da Computação Musical através de listas freqüentadas por pessoas de várias partes do mundo. Ficou evidente nas respostas um entusiasmo, talvez um pouco exagerado, dos apaixonados por Java como uma linguagem de programação genérica, mas poucos souberam responder o que efetivamente qualificava Java como uma linguagem propícia ao desenvolvimento de aplicações musicais.

Florian Bomers, engenheiro de software da Sun Microsystems e um dos responsáveis pelo desenvolvimento do Java Sound, explica que dois pontos de vista devem ser analisados nesta pergunta: Java como linguagem e Java como plataforma. Como linguagem, Bomers diferencia o Java das outras linguagens pelo fato dela trazer consigo o Java Sound, uma biblioteca padrão e extensível que garante o funcionamento da aplicação em qualquer plataforma Java, evitando que o programador tenha que tomar a difícil decisão de escolher uma outra biblioteca de manipulação sonora de baixo nível, como acontece em outras linguagens genéricas.

Apesar da Java Sound atender boa parte das aplicações musicais, o próprio Bormers reconhece suas restrições, sugerindo que a pergunta seja: "Por que Java não é boa para música?". Ele mesmo responde:*"Java não é pior que outras linguagens, exceto por especificidades como acesso direto ao processador"e continua, "Não é preciso ser melhor em processamento de som para ser uma plataforma melhor para a música".* Enaltecendo assim, as qualidades do Java como: independência de plataforma, modernidade, bibliotecas extensas e rápida execução (quando código dinamicamente otimizado e compilado). Reconhece ainda que Java possui uma latência alta se comparada com linguagens específicas para manipulação de som e seu uso deve ser

pensado em função das necessidades da aplicação, onde portabilidade deve pesar mais que desempenho.

Phil Burk, proprietário da Softsynth e um dos pesquisadores mais atuantes da Computação Musical, exalta como uma das mais importantes características do Java a possibilidade de desenvolver uma aplicação musical dinâmica em forma de *Applet* e distribuí-la facilmente.Ressalta ainda a possibilidade de combinar música com rede, gráficos e outras funcionalidades nativas do Java por ser uma linguagem aberta.

O fato de Java estar sendo amplamente utilizada na programação de aplicações musicais não significa que é a única ou a melhor linguagem para este tipo de programação. Entretanto, os programadores parecem ter maior facilidade em expressar-se através do Java seja qual for o domínio de conhecimento, inclusive a música.

As restrições do Java Sound (biblioteca nativa para manipulação de som/MIDI que acompanha o Java) não desestimularam os programadores de aplicações musicais, pelo contrário, eles parecem ter ponderado as vantagens da linguagem e, em um esforço conjunto, começaram a desenvolver novas bibliotecas mais eficientes e completas. Esta postura vez crescer o número de API disponíveis e, conseqüentemente, a dificuldade na escolha.

### 3.4.2 Bibliotecas Musicais para Java

Foram analisadas 14 pacotes que agregam ao Java a capacidade de trabalhar com áudio e dados musicais. São elas: Java Sound (JAVASOUND, 2004), JMSL (JMSL, 2003), jMusic (SORENSEN, 2003), Wire /Wire Provider (GERRIT, 2003), JavaMIDI (MARSANYI, 2003), NoSuch MIDI (NOSUCH MIDI, 2003), MIDI Share (MIDI SHARE, 2003), MIDI Kit (MCNABB, 2003), jFugue (JFUGUE, 2003), Tritonus (TRITONUS, 2003), Jsyn (JSYNTHLIB, 2003), jScore (JSCORE, 2003), Jass (DOEL, 2005) e Xemo (PROJECT XEMO, 2003).

Procurou-se ordenar a apresentação das API´s por sua relevância, funcionalidade e data de criação. Todas as informações que constam na análise foram retiradas da documentação fornecida pelo fabricante, o que nem sempre foi suficiente. Experimentos também foram realizados a fim de testar os códigos que constam nos exemplos; Cada API teve, ao menos, sua função descrita. Entretanto, para as API´s que se julgou de maior relevância, complexidade e completude, foi detalhado o principio de funcionamento e as classes mais importantes. Este estudo pode ser visto no Anexo B.

Chegou-se a conclusão que as melhores API ´s para trabalho com som (de maneira geral) são o JSyn, jMusic e Java Sound. Para trabalhar com MIDI, as melhores API´s são jMusic, JMSL e Java Sound. O destaque desta análise foi o jMusic que se mostrou completa, estável e de fácil uso em todas as categorias e acabou sendo escolhida para ajudar na implementação do trabalho proposto.

A seguir é apresentado um quadro comparativo entre as API´s analisadas dentro das classificações sugeridas.

### 3.4.3 Processamento de Áudio

Esta categoria caracteriza-se pela capacidade das API's em processarem dados de áudio, de forma genérica. As funcionalidades desejáveis podem ser vistas na Tabela 3.1.

–

Tabela 3.1: Comparação entre API's que processam áudio.

| Critério\API | **Java Soun d** | **jMusi c** | **MIDI Kit** | **JSy n** | **Titonu s** |
|---|---|---|---|---|---|
| **Gravação e reprodução de áudio;** | X | X | X | X | X |
| **Conversão e suporte a diferentes formatos de áudio;** | X | X | | X | X |
| **Manipulação de *samples* (amostrar);** | | X | | X | |
| **Manipulação e/ou processamento de dados de áudio;** | X | X | | | X |

### 3.4.4 Síntese de Áudio

Esta categoria caracteriza-se pela capacidade das API's em gera/modifica sons baseado em algoritmos de síntese, de forma genérica. Os funcionalidades desejáveis podem ser vistas na Tabela 3.2.

Tabela 3.2: Comparação entre API's que sintetizam áudio..

| Critério\API | **JSy n** | **Jas s** | **JMus ic** |
|---|---|---|---|
| **Geração de áudio a partir da conexão de unidades geradoras (osciladores, envelopes, amplificadores etc.)** | X | X | |
| **Geração de áudio modificando parâmetros físicos das ondas sonoras;** | | X | |
| **Uso de instrumentos virtuais com parâmetros de áudio pré-estabelecidos e devidamente encapsulados;** | X | | X |
| **Suporte a diversos tipos de síntese, como *wavetable*, aditiva, subtrativa, FM etc.** | X | X | X |

Vale lembrar que o Jsyn integra o pacote do JMSL.

### 3.4.5 Sequenciamento MIDI

Esta categoria caracteriza-se pela capacidade das API's em escalonar eventos MIDI no tempo. As funcionalidades desejáveis podem ser vistas na Tabela 3.3.

Tabela 3.3: Comparação entre API's que sequenciam eventos MIDI..

| Critério\API | **Java sound** | **JMus ic** | **JMS L** | **MIDI Share** | **Trito nus** |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| **Possuir seqüenciador virtual;** | X | | | X | |
| **Possuir mecanismo que permita programar / escalonar eventos MIDI para serem executadas em determinado momento de tempo;** | | X | X | | X |
| **Possuir mecanismo de sincronização;** | X | | X | X | X |
| **Leitura de gravação de arquivos MIDI;** | X | X | X | X | X |
| **Controles da execução da música (andamento, posicionamento no tempo, dinâmica)** | X | X | X | X | X |
| **Controle dos canais (dinâmica, mute, solo, timbre)** | X | X | X | X | X |

### 3.4.6 Comunicação MIDI com dispositivos externos

Muitas Api's para comunicação com dispositivos externos forma criadas devido as restrições do Java Sound em faze-lo. Atualmente este problema já foi parcialmente resolvido, mas a API's continuam sendo usadas.

Esta categoria caracteriza-se pela capacidade das API's em enviar eventos MIDI a dispositivos externos. As funcionalidades desejáveis podem ser vistas na Tabela 3.4.

Tabela 3.4: Comparação entre API's que enviam eventos s MIDI a dispositivos externos.

| Critério\API | **Java MIDI** | **MIDI Share** | **No Such** | **Wire** | **Wire Prov.** | **jMus.** | **MIDI Kit.** |
|---|---|---|---|---|---|---|---|
| **Capacidade para trocar mensagens MIDI com dispositivos de hardware externo;** | X | X | X | X | X | X | X |
| **Portabilidade (pelo menos dois S.O´s)** | X | | | | | X | |
| **Suporte para Applets** | | X | X | | | X | X |
| **Comunicação via rede** | | | | | | | X |
| **Necessidade do Java Sound** | | | | X | X | | |

* O Midi Share também está incluído no pacote do JMSL;

### 3.4.7 Síntese MIDI

A diferença da síntese MIDI em relação a síntese de áudio, na classificação adotada por este trabalho, está na entrada do processo. Enquanto a síntese MIDI consistem em basicamente gerar os sons considerando somente os dados MIDI, a síntese de áudio permite também o uso de áudio como entrada, modificando-os se desejado.

Esta categoria caracteriza-se pela capacidade das API's em gerar sons a partir de eventos MIDI. As funcionalidades desejáveis podem ser vistas na Tabela 3.5.

Tabela 3.5: Comparação entre API's que geram sons a partir de eventos MIDI.

| Critério\API | **Java Sound** | **JMusic** | **Tritonus** |
|---|---|---|---|
| **Possuir sintetizador virtual;** | X | X | X |
| **Possibilitar redirecionamento de mensagens MIDI para sintetizadores em hardware;** | | X | X |
| **Uso de soundbanks ou banco de timbres;** | X | | X |
| **Controle dos parâmetros de sínteses via API (qualidade de áudio);** | | X | X |
| **Persistência dos dados em MIDI e/ou áudio** | X | X | X |

### 3.4.8 Componentes Musicais Gráficos

Esta categoria caracteriza-se pela capacidade das API's em exibir informações musicais, em diversas notações, através das interfaces gráficas. As funcionalidades desejáveis podem ser vistas na Tabela 3.6.

Tabela 3.6: Comparação entre API's que exibem informações musicais em interfaces gráficas.

| Critério\API | **JScore** | **Xemo** | **JMusic** |
|---|---|---|---|
| **Exibição de dedos MIDI em notação musical clássica** | X | X | X |
| **Exibição de dados MIDI em notação musical alternativa** | | | X |
| **Funcionalidades de execução musical integradas** | X | | X |

### 3.4.9 Representação Musical

Esta categoria caracteriza-se pela capacidade das API's em trabalhar com outros formatos de arquivos usados para armazenar informações musicais. As funcionalidades desejáveis podem ser vistas na Tabela 3.7.

Tabela 3.7: Comparação entre API's que trabalham com formatos de arquivos musicais.

| Critério\API | JFugue | Xemo |
|---|---|---|
| **Permite codificação de dados musicais em formatos ortodoxos, diferentes do convencional MIDI** | X | X |
| **MusicXML** | | X |

### 3.4.10 Programação Musical (composição)

Uma API musical pode ser usada em software destinados para diversos fins, entretanto possuem maior vocação para alguma atividade musical. Esta categoria é composta pela API's criadas para trabalhar com composição musical. As funcionalidades desejáveis podem ser vistas na Tabela 3.8.

Tabela 3.8: Comparação entre API's criadas para serem usadas na composição musical.

| Critério\API | jMusic | JMSL | Jsyn |
|---|---|---|---|
| **Criação/edição gráfica de elementos musicais;** | X | X* | |
| **Criação/edição por código de elementos musicais;** | X | X | X |
| **Performances interativas;** | X | X | X |
| **Gravação do material gerado;** | X | X | X |
| **Suporte a MIDI** | X | X | X |
| **Síntese em tempo real;** | X | X | X |

*MusicShape Editor

## 3.5 Resumo do Capítulo

Visto os requisitos do sistema e, principalmente, visando a inclusão de futura de mecanismos capazes de tomar decisões ditas inteligentes, optou-se por uma arquitetura multiagentes para conceber o ambiente, no qual quatro agentes foram definidos: Agente Mão Esquerda (harmonia), Agente Mão Direta (ritmo e controles de expressividade), Agente Solista (melodia) e Agente Caixa de Som (controle de execução).

Para a construção do protótipo, foi escolhida a linguagem de programação Java e as API's de programação musical jMusic e Java Sound. Para a comunicação entre os agentes optou-se pelo RMI, porém as mensagens são formatadas de acorde com um protocolo de comunicação proprietário que encasula mensagens MIDI. Ainda, decidiu-se construir duas bibliotecas para programação musical em Java que será disponibilizada para a comunidade e são descritas no próximo Capítulo.

# 4 BIBLIOTECAS PARA PROGRAMAÇÃO MUSICAL - DESENVOLVIMENTO FOCANDO REUSO

Como visto no Capítulo 3, existem várias API´s em Java disponíveis para programação musical, entretanto nenhuma delas oferece suporte ao reconhecimento de cifras usualmente encontrados no Brasil, formação de acordes e geração de desenhos/representações desses acordes em instrumentos musicais configuráveis.

Essas características podem ser desejáveis em outros sistemas musicais, inclusive nos futuros sistemas do grupo, logo se optou por criar API´s para que o trabalho produzido possa ser reutilizado por toda a comunidade desenvolvedora de software musicais. Esta iniciativa vem contribuir para o desenvolvimento da área no Brasil.

O desenvolvimento de pacotes de software focados para reuso não é igual ao desenvolvimento de aplicações computacionais. Cuidados extras nos modelos, documentação e na escalabilidade devem ser tomados. Pacotes também podem ser vistos como uma forma de organização das classes do sistema e, neste sentido, alguns outros pacotes foram criados, porém estes não foram projetados para serem distribuídos e não serão citados ou descritos nesta seção.

No contexto musical, dois pacotes foram criados: br.inf.ufrgs.lcm.formacaoAcorde e br.inf.ufrgs.lcm.representacaoAcorde. Os nomes dos pacotes foram criados baseados na convenção sugerida pela Sun. Abaixo descreve-se os pacotes e como as principais funcionalidades foram implementadas. No Anexo C, distribuído em CD-Rom devido seu volume, está a documentação das API´s.

## 4.1 Pacote de Formação de Acordes (br.ufrgs.inf.lcm.formacaoAcorde);

A principal funcionalidade deste pacote é a conversão da cifra em um acorde, com suas notas e intervalos.

Acorde é uma combinação de sons simultâneos ou sucessivos quando arpejados; Cifras são símbolos criados para representar acordes, sendo compostas de letras, números e sinais. Acordes possuem três (tríades) ou quatro (tétrades) sons simultâneos, já os que possuem cinco ou mais sons são tétrades com notas acrescentadas (CHEDIAK, 1984).

Os símbolos que representam os acordes variam de cultura para cultura e estão intimamente relacionados com o estilo musical, sendo o idioma um dos fatores envolvidos nesta questão, não havendo ainda um padrão estabelecido, o que pode dificultar a leitura da música.

A falta de padrão não chega a ser um grande problema para músicos mais experientes e, talvez por isso, é constantemente desconsiderada nos software de performance musicais

(IPS) atuais. Este problema atinge os músicos iniciantes que, em busca de músicas que os agradem, acabam por deparar-se com cifras desconhecidas.

É importante lembrar que as cifras definem simplesmente as notas que serão tocadas, ficando a cargo do executor a escolha do momento em que cada nota será tocada (ritmo).

Diferentes cifras podem representar o mesmo acorde. Por exemplo, o acorde “dó maior com sétima maior” (**C7M**) tem a mesma formação do acorde "mi menor com baixo em dó” (**Em/C**) e sua cifra **C7M** pode ser escrita como **Cmaj7**, dependendo da notação utilizada, neste caso a americana.

Alguns símbolos e regras da cifragem são contraditórios, o que dificulta uma formalização computacional. Acompanhe o exemplo: o acorde **C5** é formado por simplesmente duas notas: Dó (I) e Sol (V). A cifragem não tem o papel de indicar duplicações, triplicações, supressões e outros artifícios utilizados devido à restrição do instrumento. Entretanto por vezes se faz, como no caso do chamado acorde força **C5**, onde o Sol (V) deve ser duplicado ou dobrado, sendo assim, o acorde é formado pelas notas; Dó, Sol e novamente Sol. Esta regra não é válida para intervalos de sétima por exemplo, onde o intervalo é inserido sem que nenhum outro seja retirado.

As principais classes deste pacote são descritas a seguir:

### 4.1.1 CifraValida

Representa a cifra já validada segundo a notação vigente e pronta para se transformar em um acorde (classe Acorde). Um objeto da classe CifraValida armazena , entre outras coisas, a nota fundamental e os intervalos descritos na cifra. Com estas informações é possível calcular o acorde.

A opção de se definir um objeto intermediário entre o texto da cifra e o acorde (objeto da classe Acorde) deu-se devido à necessidade de rastreamento do processamento da cifra, isto é, é desejável que o usuário saiba porque sua cifra não está correta e qual seria o correto. Este conhecimento não é do acorde, que tem suas próprias regras de criação.

Apesar de possível, não é recomendado que os programadores instanciem objetos da classe CifraValida diretamente, ou seja, a forma recomendada para se obter este objetos é através da classe CifraValidaMaker.

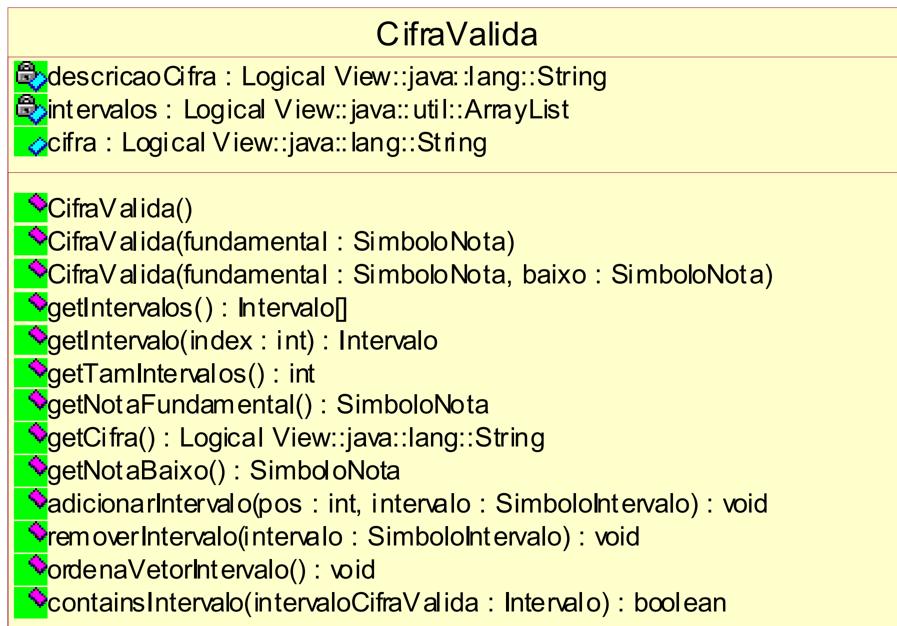



Figura 4.1 : Classe CifraValida

A Figura 4.1 mostra uma simplificação da classe CifraValida. Para lista completa dos métodos e sua descrição detalhada consulte a especificação da API no Anexo C.

### 4.1.2 CifraValidaMaker

Gera objetos da classe CifraValida baseada em uma entrada textual da cifra e a notação de cifragem vigente (classe NotacaoCifra). É a principal classe deste pacote, onde foram implementados os autômatos que reconhecem uma palavra como cifra.

O sufixo Maker indica que esta classe funciona como uma fabrica (*Factory Design Pattern*) de objetos da classe CifraValida e somente um método público foi implementado, como mostra a Figura 4.2.

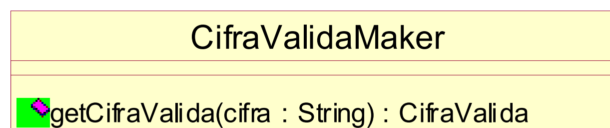



Figura 4.2 : Classe CifraValidaMaker

Para a construção desta classe foi necessário primeiramente formalizar a linguagem com os símbolos mais usados na cifragem brasileira. Isto não quer dizer que o sistema só funcionará com esta notação, mas que esta notação será a padrão para o sistema, podendo ser modificada ou mesmo substituída por outra a critério do usuário. Isto garante que as cifras exibidas nas tablaturas sejam familiares ao usuário, facilitando a leitura da música.

Os símbolos utilizados e suas funções podem ser observados na Tabela 9.

Tabela 4.1: Símbolos definidos na notação brasileira de cifragem

| Símbolo | Descrição | Símbolo | Descrição |
|---|---|---|---|
| A | Nota Lá | ° | Diminuto |
| B | Nota Si | 2 | Segunda Maior |
| C | Nota Dó | b2 | Segunda Menor |
| D | Nota Ré | 4 | Quarta Justa |
| E | Nota Mi | #4 | Quarta Aumentada |
| F | Nota Fá | 5 | Quinta Justa |
| G | Nota Sol | #5 | Quinta Aumentada |
| # | Alteração Sustenido | b5 | Quinta Diminuta |
| b | Alteração Bemol | 6 | Sexta Maior |
| add | Adição de Intervalo | 7 | Sétima Menor |
| ( | Início de Alteração de Nota | 7M | Sétima Maior |
| ) | Fim de Alteração de Nota | 9 | Nona Maior |
| / | Inversão no Acorde | b9 | Nona Menor |
| ^ | Junção de intervalos | 11 | Décima Primeira Justa |
| m | Intervalo maior | #11 | Décima Primeira Aumentada |
| m | Intervalo menor | 13 | Décima Terceira Maior |
| sus | Suspensão | | |

Na linguagem indicada na Tabela 4.1, os parênteses são usados quando existe uma nota alterada (# | b) ou quando houver intervalos que não compõem a estrutura básica da tétrade (9, 11, 13). Definiu-se ainda que os acordes diminutos são somente as tétrades, ou seja, o acorde C° é formado pelos intervalos 1+b3+b5+b7. Se quisermos apenas a tríade 1+b3+b5, então a cifra seria **Cm(b5).**

O símbolo "add" indica a adição de um intervalo que não é o intervalo seguinte em relação ao último intervalo do acorde. Ex: **C** = 1+3+5; **C(add 9)** = 1+3+5+9.

O símbolo "^" significando a junção de intervalos não é um símbolo musical conhecido. Na verdade este símbolo foi criado por restrições técnicas do autômato, visto que o mesmo usa o espaço em branco como sinal de término de uma cifra, ou seja, se desejarmos validar o acorde **C7M(9 11)** teríamos problemas com o espaço entre os intervalos de nona(9) e décima primeira(11). Para isto optou-se pelo uso do caractere "^", ficando o acorde com a seguinte aparência **C7M(9^11)** . Para contornar esta restrição, pode-se criar uma interface gráfica para a entrada das cifras que sobreponha os intervalos 9 e 11, simulando uma fração, como ocorre nas tablaturas. O resultado seria:

$$C7M\begin{pmatrix} 9 \\ 11 \end{pmatrix}.$$

As regras de formação dos acordes em conjunto com a linguagem definida tornam possível a construção de um autômato finito para validação de cifras, representado pela 5-upla D1 = (Σ, Q, δ, q0, F) onde:

a) Σ é o alfabeto de símbolos de entrada, Σ = nota ∪ alt ∪ sus ∪ var

nota = {A, B, C, D, E, F, G}

alt = {#, b}

var = {º, m, 5}

susN = {sus2, sus4, sus9, sus11}

num = {2, 4, 6, 5, 7, 9, 11, 13}

numA = {2,4,6}

numB={9, 11, 13}

numC = {2, 4, 9, 11}

numD = {2, 4, 5, 6}

b) Q é conjunto de estados possíveis, Q = {S0.. S69}

c) δ é a função parcial de transição $\delta : Q \times \Sigma \rightarrow Q$

d) q0 é o estado inicial, q0 = S0

e) F é o conjunto de estados finais, F = {S2, S3, S4, S5, S6, S7, S9, S16, S18, S19, S20, S21, S22, S31, S32, S33, S39, S40, S48, S65, S68, S69}

Graficamente, D1 está representado parcialmente pela Figura 4.3.

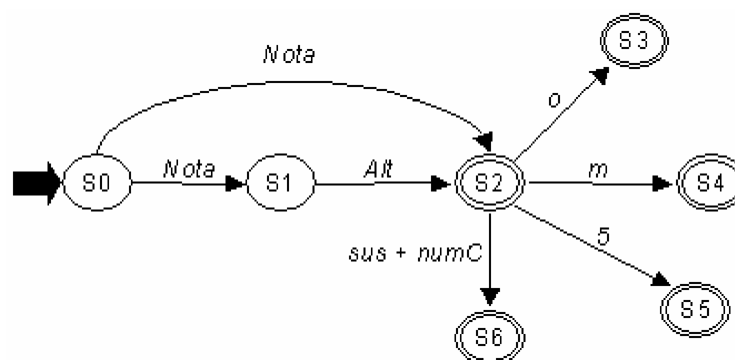



Figura 4.3 : Autômato finito inicial.

Na Figura 4.3, observa-se o autômato inicial capaz de validar acordes maiores, menores, diminutos, força (duplicação da quinta justa) e suspensos. Os estados finais S2, S3, S4 e S5 levam a sub-autômatos que tratam e validam qualquer cifra. Como por exemplo, o autômato representado na Figura 4.4, que faz o tratamento das tétrades (acordes com o intervalo de sétima) já validados pelo autômato inicial (estado final S3).

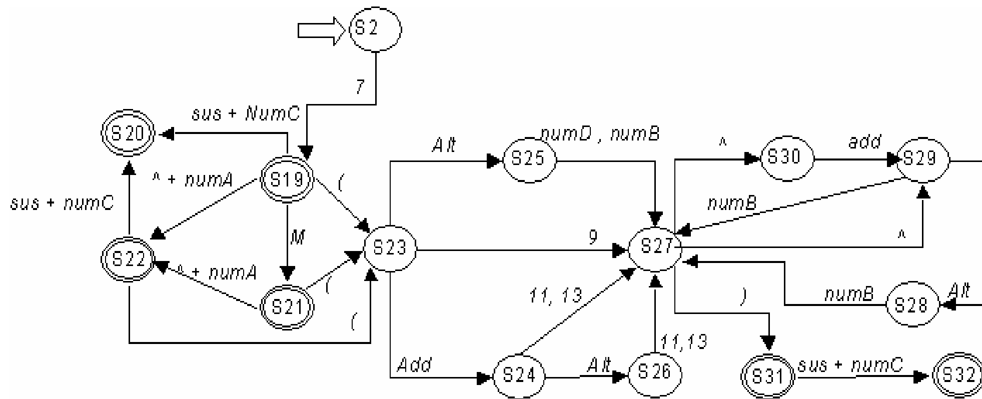



Figura 4.4: Autômato para tratamento de tétrades (acorde com sétima) a partir do estado S2

Ainda partindo do estado S2, o autômato da Figura 4.5 se encarrega de validar as cifras dos acordes com intervalos de adicionados de 2, 4, 5,6 e 7. Os outros intervalos devem estar entre parênteses.

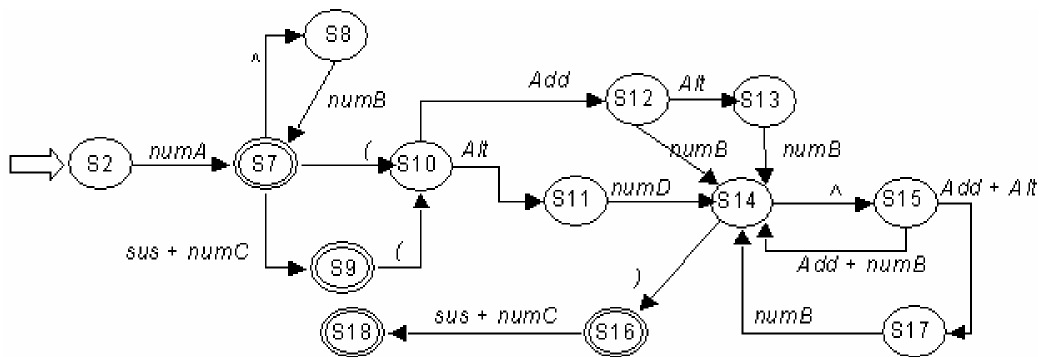



Figura 4.5: Autômato para tratamento de acorde com os intervalos de segunda, quarta ou sextas.

A seguir, os autômatos que validam os acordes diminutos, menores, força e suspensos, respectivamente (ver Figura 19).

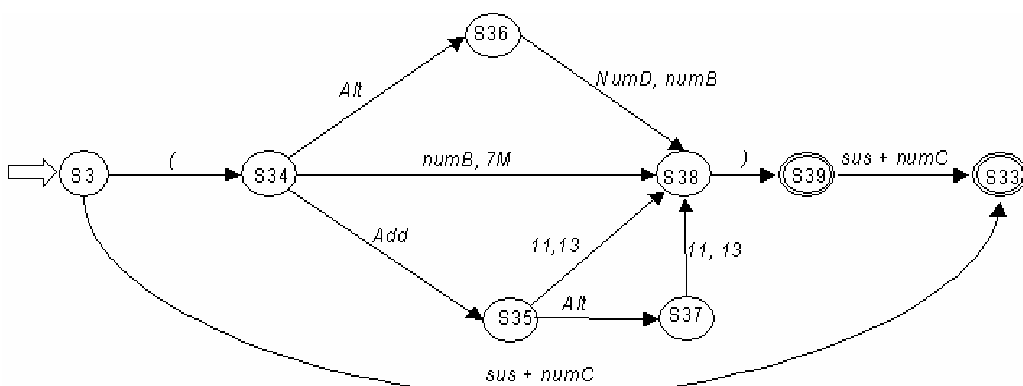



Figura 4.6: Autômato para tratamento de tétrades diminutas.

O sub-autômato representado na Figura 4.6 valida acordes com inclusões e intervalos e suspensões dos acorde diminutos, ou seja, a partir do estado S3. Exemplo: **G#dim(add11).**

Os autômatos da Figura 4.7 e Figura 4.8 tratam os acordes menores, sendo o segundo para tétrades menores.

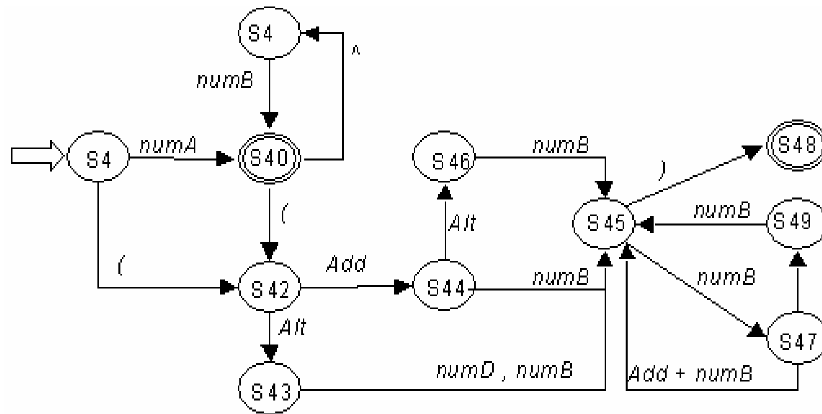



Figura 4.7: Autômato para tratar acordes menores.

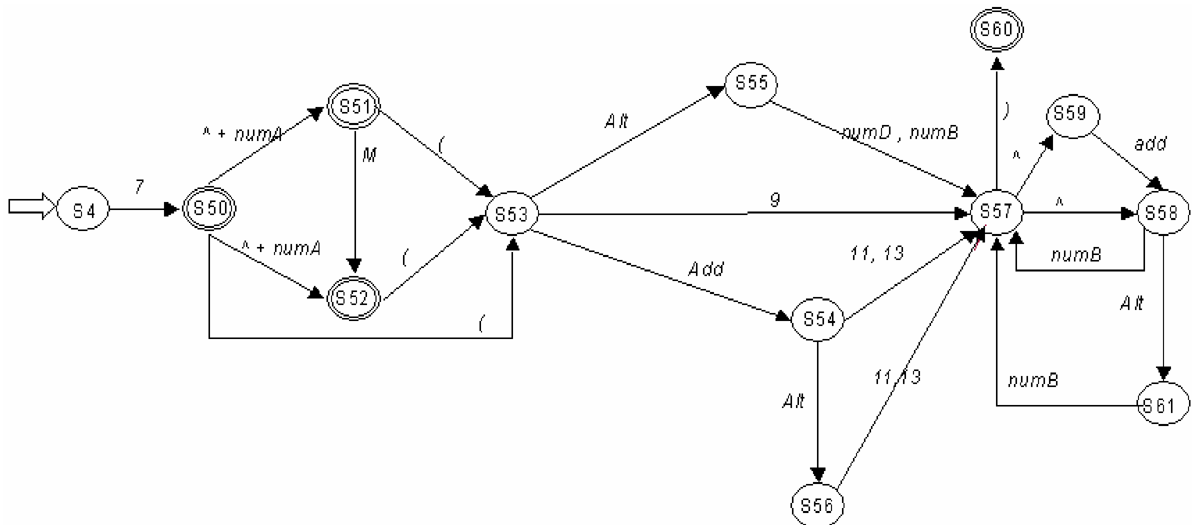



Figura 4.8: Autômato para tratamento de acordes menores com intervalo de sétima.

O autômato da Figura 4.9 trata alteração da quinta dobrada em acordes força, por exemplo **A5(b5).**

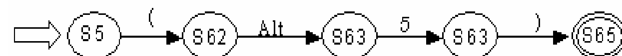



Figura 4.9: Autômato para tratar acordes força (intervalo dobrado de quinta)

O autômato da Figura 4.10 inicia a partir de todos os outros estados finais, exceto S5, S65, S68 e S69.

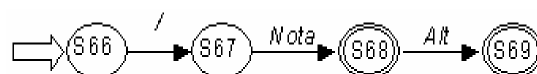



Figura 4.10: Autômato que trata alteração no baixo do acorde e inversões.

### 4.1.3 Acorde

Um objeto da classe acorde é um conjunto de notas que compõem o acorde. Essas notas, juntamente com os intervalos que as representam, são encapsuladas na classe NotaAcorde. Logo, um acorde é um conjunto de objetos da classe NotaAcorde, como mostra a Figura 4.11.

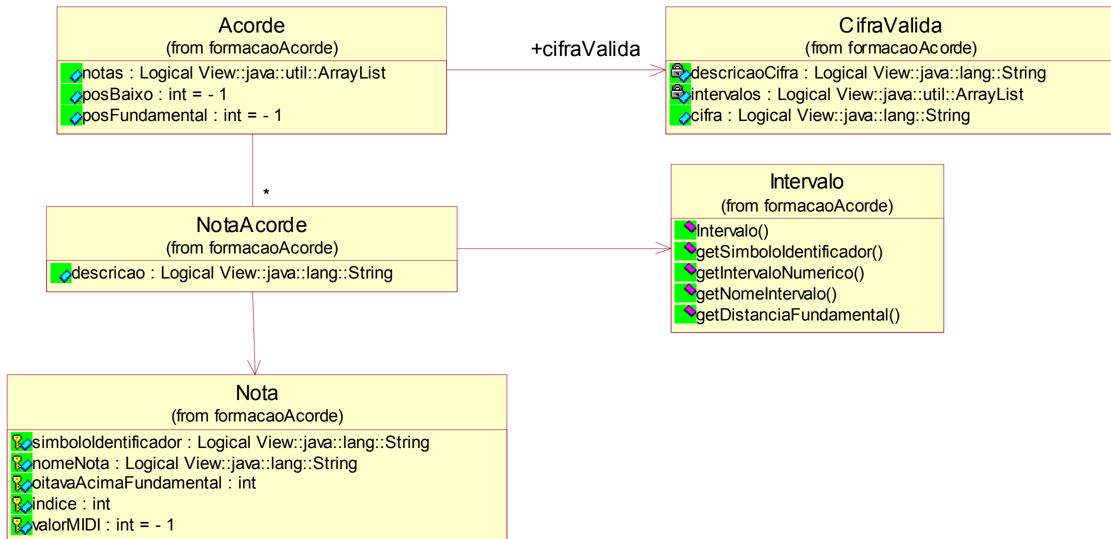



Figura 4.11: Classes associadas a classe Acorde

Na Figura 4.12 é possível observar detalhes da classe acorde, como por exemplo o atributo posBaixo que é usado quando o acorde possui uma inversão.

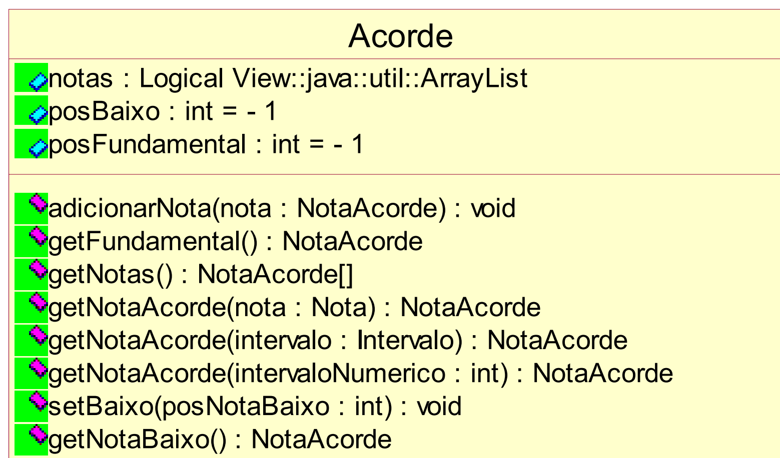



Figura 4.12: Classe Acorde

### 4.1.4 AcordeMaker

Gera um objeto da classe Acorde tendo como entrada um objeto da classe CifraValida. Se identificada e validada a cifra que representa o acorde, então é conhecido: a nota fundamental, intervalos e possível inversão do acorde. Com base nestas informações o acorde é montando (suas notas são definidas) considerando as regras de formação de

acordes, que são muito mais formais e estão padronizadas. Exemplo: Uma tríade menor é sempre composta dos intervalos 1 + b3 + 5;

Na Figura 4.13 é possível observar a simplicidade da classe AcordeMaker que também é uma fábrica de objetos, tal qual CifraValidaMaker, porém estes do tipo Acorde.

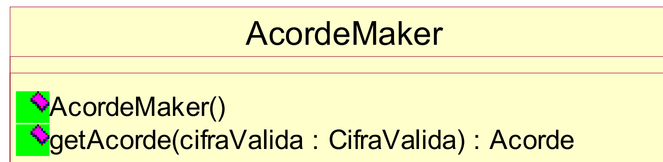



Figura 4.13: Classe AcordeMaker.

Um tratamento diferenciado foi dado aos acordes que possuem o baixo em outra nota que não seja a Fundamental. Exemplo: **C/G**. Este acorde possui uma inversão, isto é a nota Sol (**G**) representada no baixo já compõe a estrutura básica do acorde, logo, simplesmente defini-se que o Sol será a nota mais grave. Isto é diferente de um **Am/F#,** pois o **F#** não está entre as notas do Am, ou seja, precisa ser adicionado, mesmo que não esteja representado o intervalo de sexta maior (**F#**) na cifra.

## 4.2 Pacote de Geração das Representações dos Acordes (br.ufrgs.inf.lcm.representacaoAcorde)

Este pacote abrange todo processamento relativo às representações ou desenhos dos acordes, envolvendo classes de instrumentos, manipulação e geração destas representações. Esta última pode ser considerada a principal funcionalidade do pacote.

Quando se fala em sistemas que trabalham com representações de acordes, a abordagem mais comum é a indexação da imagem do acorde no instrumento tendo como índice as cifras musicais. Apesar de ter uma rápida recuperação da informação e relativa simplicidade na implementação, este tipo de abordagem é inflexível, ficando restrita às cifras e ao instrumento para o qual foi projetada. Outra forma possível é o cálculo da representação, como será visto neste pacote.

A geração da representação que será executada no instrumento se dará em duas fases. Na primeira, a representação é preenchida com as posições das notas no violão (corda e traste) e um conjunto de representações básicas é retornada. Definiu-se como representação básica aquela que não possua notas dobradas, duplicadas ou suprimidas em função das restrições do instrumento. Esta divisão em fases foi uma decisão de projeto que ponderou o tempo de processamento necessário para se obter a melhor representação completa (onde há dobramentos), visto que para se chegar a melhor representação, todas as representações teriam que ser integralmente calculadas. Trabalhando por fases, conseguimos resultados aceitáveis usando como base de comparação as representações básicas obtidas na primeira fase e processando (calculando dobramentos) somente da representação escolhida.

A Figura 4.14 mostra os componentes e suas funções no processo de geração das representações básicas de um acorde, relativo a fase 1 do processo global acima mencionado. O início do processo ocorre com a entrada do texto que representa uma cifra. Um “Autômato Finito” (classe formacaoAcorde.CifraValidaMaker) com as regras

de formação de acorde busca os símbolos que deve reconhecer na "Notação Proposta" (classe NotacaoCifra). Após a validação do texto como uma cifra, gera-se um objeto com os intervalos e notas extraídos da cifra (classe CifraValida). Este objeto será então processado (classe AcordeMaker) e os intervalos darão lugar as notas(NotaAcorde) em um novo objeto representando o acorde (Acorde). Para se obter as representações básicas (RepresentacaoAcordeMaker) de um acorde é necessário conhecer o instrumento (InstrumentoCorda / ViolaoPlugin) e as preferências do músico (RepresentacaoAcordeProperties).

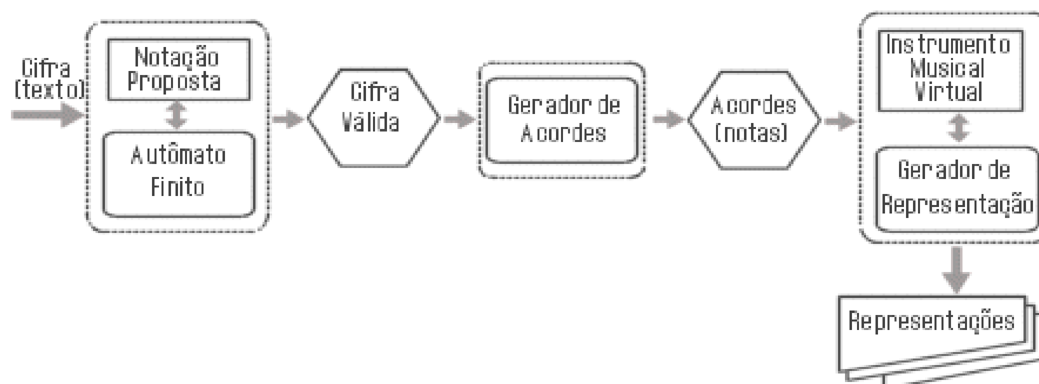



Figura 4.14: Processo de geração das representações básicas de acordes.

Como exemplo deste processo foi implementado um dicionário de acordes de representações básicas que pode ver visto na Figura 4.15. Neste exemplo o texto "**G#m7(b5)**" foi passado como uma cifra, validado e convertido em uma cifra válida. A cifra válida foi então processada e gerou-se o acorde, que teve suas representações básicas calculadas.

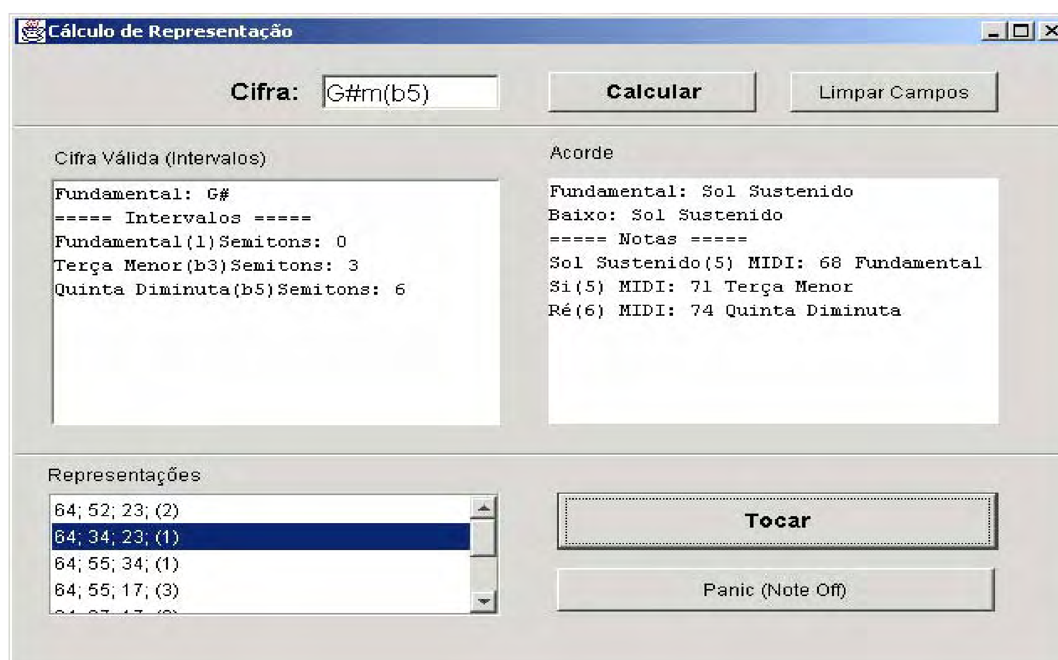



Figura 4.15: Dicionário de acordes de representações básicas.

Um acorde não possui mais notas do que a capacidade polifônica do instrumento, no caso do violão, 6 notas simultâneas (assumindo o violão padrão de seis cordas). Isto quer dizer que as representações básicas podem ter espaço sobrando para repetição de certas notas adaptando-a as necessidades, gostos e habilidade do músico, características do estilo musical e padrão rítmico.

Omissões, duplicações, dobramentos, triplicações e inversões das notas dos acordes, são pós-tratamentos feitos em cima de um conjunto de representações básicas, considerando as restrições físicas do instrumento e configurações do usuário. Por exemplo, o usuário pode definir que, a nota que representa o intervalo de terça, jamais deve ser dobrada em suas representações. Estes processamentos são relativos a segunda fase do processo de geração de representações, bem como o cálculo da sugestão dos dedos de cada posição da representação, e são descritos na classe ProcessadorRepresentacaoInstrumentoCorda,

As principais classes do pacote serão descritas seguir. Para um maior detalhamento das classes, seus métodos e atributos, consultem a especificação da API's no Anexo C.

### 4.2.1 Instrumento

A classe abstrata Instrumento define o que um instrumento musical precisa ter para ser incorporado ao sistema. Este hierarquia foi definida visando a expansão do sistema por novos instrumentos que possam vir a ser criados no futuro. A Figura 4.16 mostra as associações da classe Instrumento.

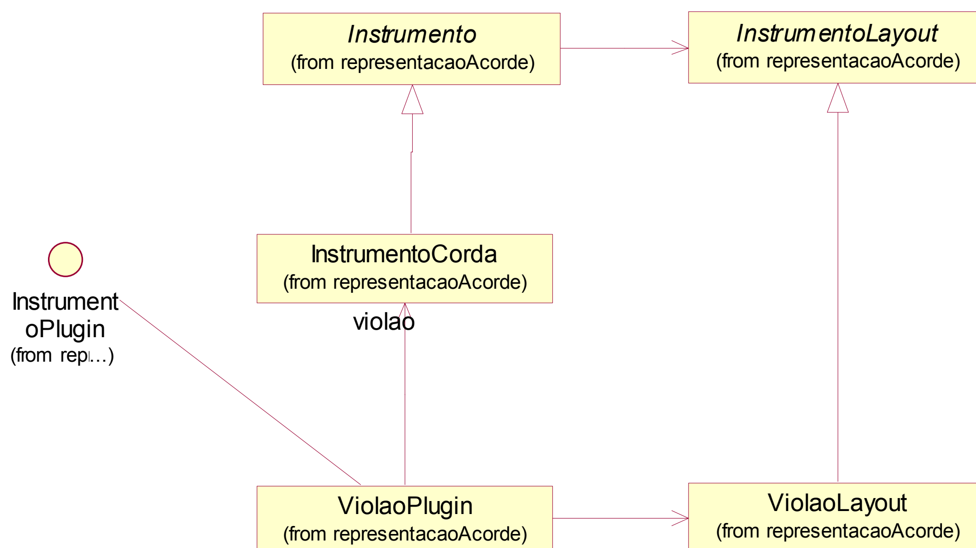



Figura 4.16: Ligações da classe Instrumento

Analisando a Figura 29 podemos verificar que existe uma ligação entre a classe Instrumento e InstrumentoLayout. A classe InstrumentoLayout tem o propósito representar graficamente o instrumento em questão, permitindo, por exemplo, uma visualização do desenho do acorde no próprio instrumento. Nesta primeira versão, esta funcionalidade não foi implementada.

### 4.2.2 InstrumentoCorda

O principal instrumento em estudo neste trabalho é o violão, entretanto notou-se que os algoritmos criados são genéricos para qualquer instrumento trasteado de corda. Baseado nisto, optou-se por generalizar as características dos instrumentos de corda na superclasse InstrumentoCorda.

Nesta classe encontram-se alguns atributos que influenciam diretamente no processamento e geração das representações dos acordes. Note que estas representações são específicas para a instancia do instrumento de corda criado. Na Figura 4.17 é possível observar os detalhes do comportamento e estado da classe InstrumentoCorda.

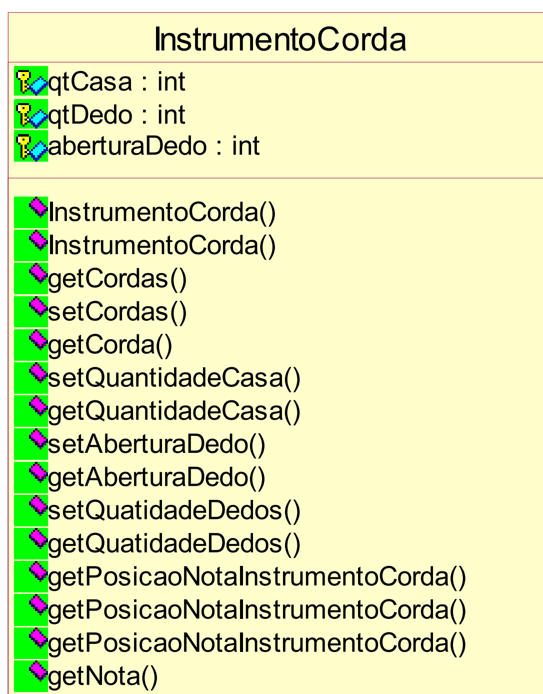



Figura 4.17: Classe InstrumentoCorda

### 4.2.3 InstrumentoPlugin

Interface que permite a inclusão de um novo instrumento no sistema durante sua execução. Para que um instrumento possa ser incorporado ao sistema é necessário que a nova classe criada, além de ser filha da superclasse Instrumento, implemente a interface InstrumentoPlugin. Note que não há a necessidade de se implementar este interface se o instrumento vier junto com o pacote, ou seja, este recurso se aplica somente em casos onde o usuário deseja obter (ou programar) um instrumento que não compõe o pacote

original ou que seja adicionado ao sistema em tempo de execução. A carga do "*plugin*"é feita no sistema através da interface InstrumentoManager.

Para demonstrar a utilização da interface, o pacote traz a classe ViolaoPlugin, que retorna um objeto do tipo InstrumentoCorda com as características de um violão de 6 cordas. São elas: 12 trastes[8], 4 dedos para execução de acordes, abertura máxima de 4 casas e afinação tradicional (Mi, Lá, Ré, Sol, Si, Mi). As configurações podem ser alteradas em tempo de execução. A Figura 4.18 representa no diagrama o que foi dito.

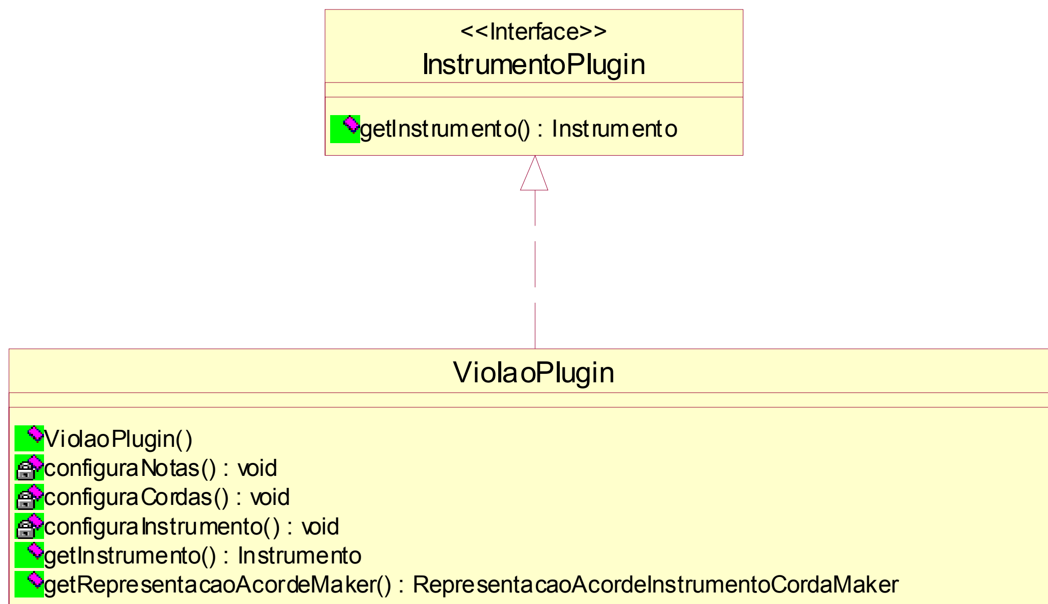



Figura 4.18 : Criando um violão.

### 4.2.4 RepresentacaoAcorde (RepresentacaoAcordeInstrumentoCorda)

Representa a forma com que um acorde deve ser executado em um determinado instrumento.

A forma mais conhecida entre os músicos de se representar um acorde, independentemente do instrumento, é a notação clássica onde as notas e os intervalos são descritos em um pentagrama. Sem os devidos conhecimentos teóricos musicais a compreensão da notação clássica não é trivial, pois esta, apesar de ser eficaz, não é intuitiva (amigável).

A representação dos acordes sobre a imagem do instrumento permitiu aos aprendizes uma visão mais intuitiva de como os acordes devem ser executados, entretanto, não se

---

[8] Normalmente, o violão possui 19 trastes, entretanto o software esta se limitando a 12 trastes por ser esta a região do instrumento mais utilizada para execução de acompanhamentos com acordes.

pode esquecer que acordes são genéricos a todos os instrumentos harmônicos. Recursos multimídia podem ser usados para se obter uma notação mais rica (ROADS, 1996).

Um acorde pode ser executado de diferentes formas no mesmo instrumento. Isto ocorre porque a maior parte dos instrumentos possui um registro superior a uma oitava, fazendo com que as notas se repitam. As cifras não determinam qual a altura das notas que compõem os acordes, ficando isto a cargo das restrições do instrumento e opções do instrumentista.

A representação do acorde leva em consideração, para seu cálculo, as particularidades de cada instrumento musical. No caso do violão, os principais parâmetros do algoritmo que gera as representações são a quantidade de trastes e cordas, a afinação das cordas, a abertura máxima dos dedos do violonista, o uso de pestana e quantidade dos dedos disponíveis para execução. Como visto anteriormente, estes parâmetros estão definidos na classe InstrumentoCorda e, considerando que a representação do acorde é específica para cada tipo de instrumento, é necessário que haja uma classe que expresse as particularidades de uma representação para um instrumento de corda. Esta classe é a RepresentacaoAcordeInstrumentoCorda.

A Figura 4.19 mostra as particularidades de uma representação de acorde para um instrumento de corda. Chama-se a atenção para a multiplicidade do relacionamento entre as classes RepresentacaoAcordeInstrumentoCorda e Pestana. Note que mais de uma pestana foi autorizada em uma mesma representação e esta situação, apesar de possível, torna a representação muito difícil de executada. Mediante este fato, não foi implementada nesta versão esta possibilidade.

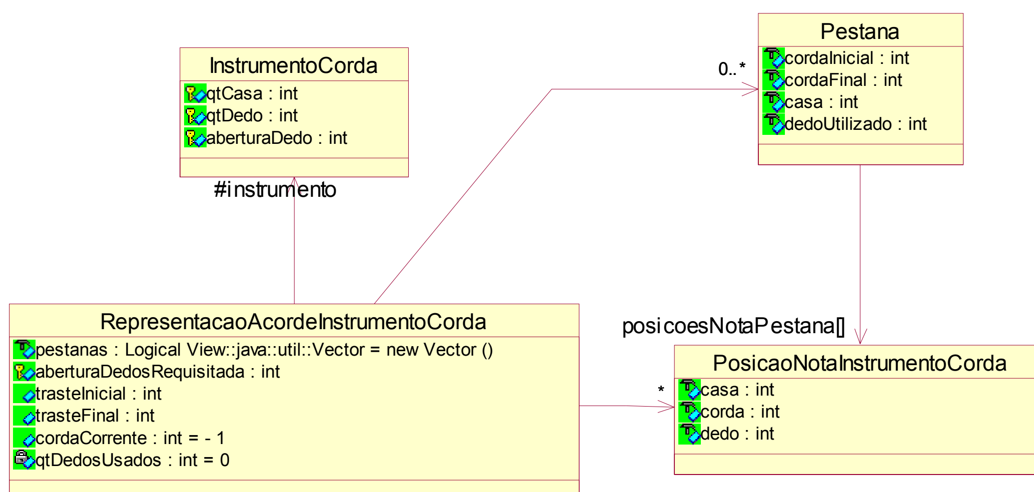



Figura 4.19: Particularidades de uma representação de acorde para um instrumento de corda.

Talvez o termo mais conhecido na música para o que foi definido como representação de acorde seja desenho do acorde (*shape*). Entretanto, optou-se por manter o nome "representação de acorde" considerando que este nome seria mais genérico e faria menção a forma com que o acorde seria mostrado no instrumento, ou seja, sua representação (possivelmente gráfica) em um determinado instrumento.

Dois conceitos serão importantes quando falarmos na escolha da próxima representação de um acorde: similaridade dos desenhos dos acordes e a facilidade de execução destes

desenhos. Como exemplo, suponha uma música com a progressão harmônica de C - G (Dó e Sol maior). O primeiro passo é saber como se executa o Dó Maior no violão e são várias as possibilidades. A Figura 4.20 mostra uma das formas mais comuns também lida como 53-42-30-21-10 [9].

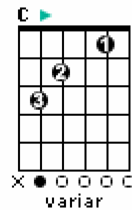



Figura 4.20: Representação do acorde Dó Maior.

O próximo acorde que devemos saber como executar é o Sol Maior (G) e, novamente, temos diversos desenhos do sol no violão como mostra a Figura 4.21. Qual deles devemos escolher?

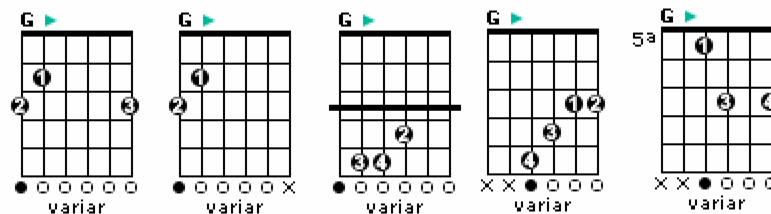



Figura 4.21: Algumas possibilidades do acorde Sol Maior no violão.

Uma das respostas é: escolhe-se a representação que exige menos esforço do instrumentista na passagem de um acode para o outro, ou seja, a digitação mais próxima da digitação anterior. Nova pergunta: Como se calcula a similaridade entre as representações?

O algoritmo usado para calcular a similaridade das representações será explicado no detalhamento da classe ProcessadorRepresentacaoAcordeInstrumentoCordaMaker, entretanto adianta-se que as variáveis envolvidas são: quantidade de notas simultâneas, quantidade de notas iguais e distancia entre as posições das notas. Submetido a este exemplo o sistema escolheu a seguinte representação para o acorde Sol: 63 – 52 – 40 – 23 – 13.

Uma outra variável envolvida na escolha da próxima representação é a facilidade de execução do desenho e isto é calculado com base na abertura de dedos necessária para

[9] Lê-se: quinta corda-terceira casa, quarta corda-segunda casa, terceira corda solta, segunda corda – primeira casa e primeira corda solta.

execução, quantidade de cordas soltas e proximidade da cabeça do violão. Os detalhes serão mostrados na descrição da classe ProcessadorRepresentacaoAcordeInstrumentoCordaMaker.

### 4.2.5 RepresentacaoAcordeInstrumentoCordaMaker

Gera os objetos da classe RepresentacaoAcordeInstrumentoCorda relativos a um determinado InstrumentoCorda. A classe RepresentacaoAcordeInstrumentoCordaMaker implementa a interface RepresentacaoAcordeMaker, que define as características e comportamento básico de um gerador de representações de acorde.

Como dito anteriormente, para cada tipo de instrumento passível de executar acordes (instrumentos harmônicos) é necessário ter uma implementação da interface RepresentacaoAcordeMaker, isto porque existem peculiaridades e especificidades na execução de cada instrumento. Por exemplo, o uso de pestanas não se justifica em um teclado musical.

A Figura 4.22 mostra algumas das especificidades de um gerador de representações para instrumentos de cordas trasteados[10].

---

[10] **Trastes**: Série de filetes de metal, madeira ou tripa atravessados no braço de certos instrumentos de corda para tornar mais fácil o ponteio das cordas. Exemplo de instrumentos trasteados são o violão, guitarra, cavaquinho, banjo etc.

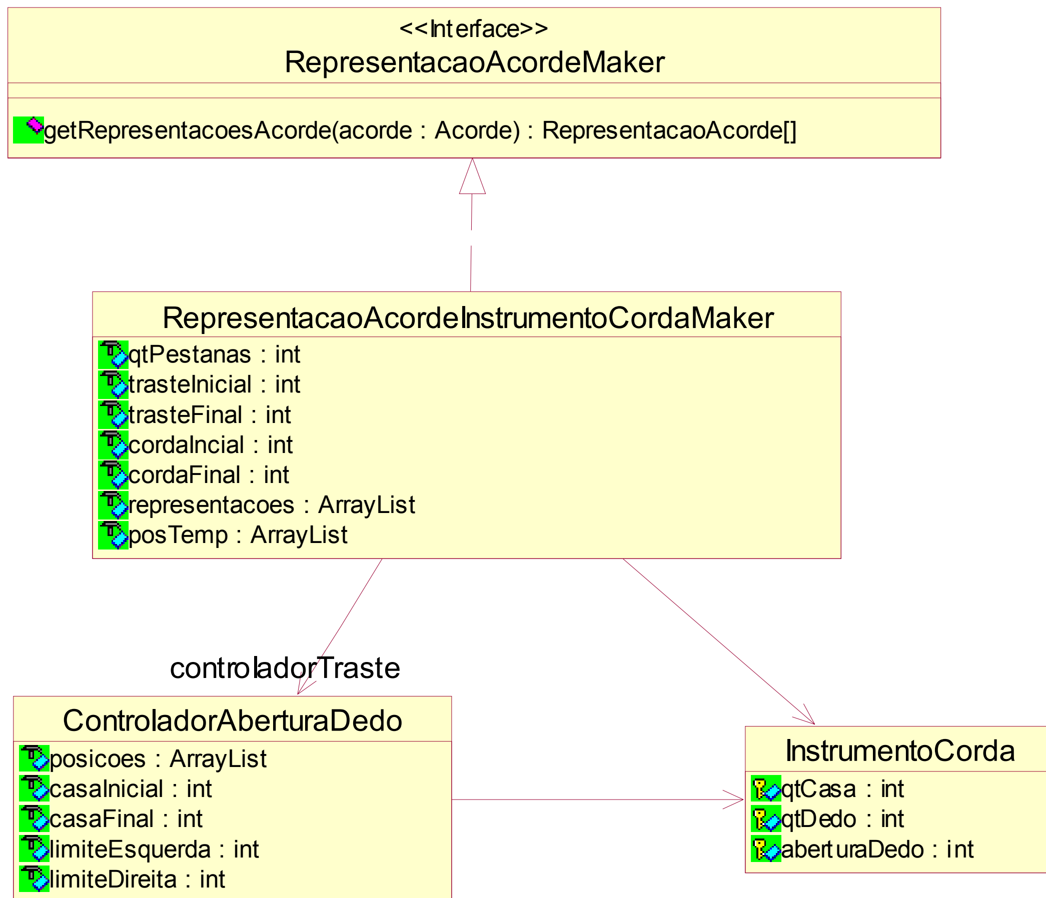



Figura 4.22 : Implementação de um gerador de representações de acordes para instrumentos de corda.

Observe que o RepresentacaoAcordeInstrumentoCordaMaker possui uma dependência do ControladorAberturaDedo. Esta classe é um mecanismo criado para impedir que representações de acordes calculadas extrapolem a capacidade de execução do músico em relação à quantidade de dedos usados e a abertura máxima dos dedos. Além disso, esta classe restringe o espaço da busca da nota em uma corda, otimizando o algoritmo que monta a representação.

A Figura 4.23 mostra a classe RepresentacaoAcordeInstrumentoCordaMaker, onde 3 métodos merecem destaque:

**GetRepresentacoesAcorde(Acorde)**: Método definido na interface RepresentacaoAcordeMaker que retorna um vetor de objetos RepresentacaoAcorde das representações básicas relativas ao acorde passado.

**GetRepresentacaoAcordeInstrumentoCordaFacil(Acorde, qtNotasSimultaneas)**: Retorna a representação mais fácil de executada com um número de notas simultâneas e acorde passados como paramentos.

GetRepresentacaoAcordeInstrumentoCordaSimilar(RepresentacaoAcordeIsntrumentoCorda, Acorde, qtNotasSimultaneas): Retorna a RepresentacaoAcordeIsntrumentoCorda do acorde passado mais similar a representação anterior.

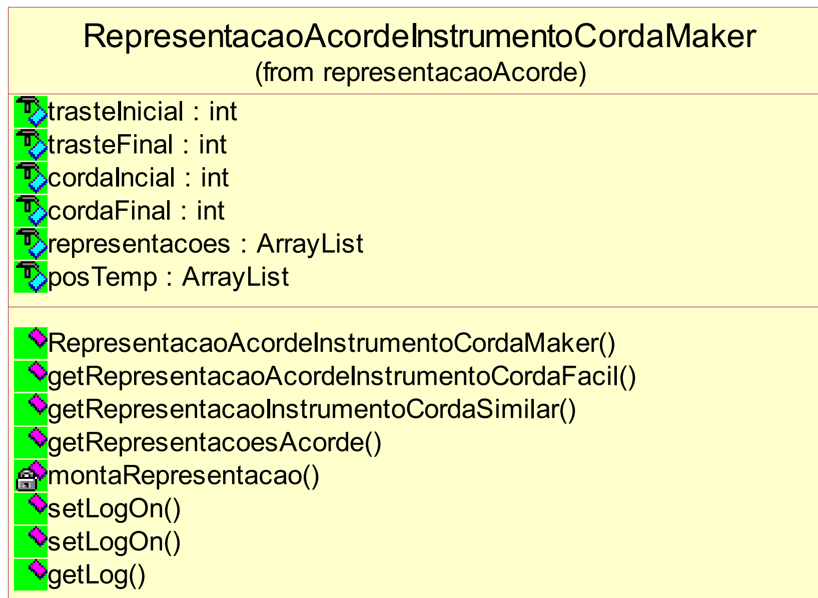



Figura 4.23: Classe RepresentacaoAcordeInstrumentoCordaMaker.

O Algoritmo 1 mostra a lógica superficial do cálculo das representações básicas relativo ao método getRepresentacoesAcorde.

*Contador Cordas = Quantidade de cordas do Instrumento;*
*Quantidade Notas = Quantidade de notas do Acorde;*

*Enquanto Contador Cordas >= Quantidade Notas faça*
  *Busca a nota do baixo na corda (Contador Cordas) do instrumento;*
*Se existir a nota então*
  *Adicionar a posição da nota em uma nova representação;*
  *Remover a nota achada da lista de notas do acorde;*
*Procurar as notas restantes nas cordas inferiores a corda do baixo;*
*Fim se*

*Contador Cordas –1;*
*Fim Enquanto*

Algoritmo 1: Cálculo das representações básicas

As representações processadas são ordenadas pela abertura dos dedos (decrescente) e pela quantidade de dedos necessária para execução do acorde (crescente) e pela proximidade da cabeça do violão (crescente). Desta forma, a primeira posição da lista será a representação mais fácil de ser executada. Este princípio é usado nos métodos getRepresentacaoAcordeInstrumentoCordaFacil e getRepresentacaoAcordeInstrumentoCordaSimilar e serão descritos a seguir.

O Algoritmo 2 mostra como é feita a escolha da representação mais fácil do acorde implementado no método getRepresentacaoAcordeInstrumentoCordaFacil.

*Quant. Notas simultâneas = número de notas simultâneas definidas pelo usuário*

*Cont. Posição = 0;*
*Calcular representações básicas do acorde;*

*Enquanto não achar uma representação completa com(Quant. Notas simultâneas) E (Cont Posição < quantidade elementos da lista) Faça*
*Processar a entrada(Cont. Posição ) das representações básicas (ordenadas por facilidade);*
*Se existe alguma representação na lista com (Quant. Notas simultâneas)*
*Retorna a representação;*
*Senão*
*Cont. Posição + 1;*
*Fim Se*
*Fim enquanto*

Algoritmo 2: Algoritmo para achar a representação de acorde mais fácil de ser executado.

O Algoritmo 3 mostra como é feita a escolha da representação mais similar a representação do acorde anterior da música. Este algoritmo foi implementado no método getRepresentacaoAcordeInstrumentoCordaSimilar.

*Calcular representações básicas do Acorde*
*posEscolhida = Solicitar ao Processador que identifique a representação básica mais similar a representação do acorde prévio e armazenar o índice em posEscolhida;*
*Enquanto (posEscolhida < numero de representações)*
*Processar a representação(posEscolhida );*
*Se o processamento retornou alguma representação com mesmo número de notas simultâneas então*
*Retornar representação*
*Senão*
*posEscolhida +1;*
*Fim se*
*Fim Enquanto*

Algoritmo 3: Calculo da representação de acorde similar.

### 4.2.6 ProcessadorRepresentacaoAcordeInstrumentoCorda

Esta classe tem o objetivo de executar operações/processamentos sobre objetos do tipo RepresentacaoAcordeInstrumentoCorda.

Na Figura 4.24 é possível ver os processamentos implementados até o momento, dentre os quais 3 são destacados:

**processarRepresentacaoAcorde():** Retorna uma lista de todas as representações completas possíveis, com os dobramentos configurados pelo usuário, a partir de uma representação básica.

**getRepresentacaoAcordeSimilar():** Escolhe dentre um conjunto de representações básicas, qual é a mais similar a representação do acorde anterior da música.

**indicarDedos():** calcula quais dedos devem ser usados na execução do acorde.

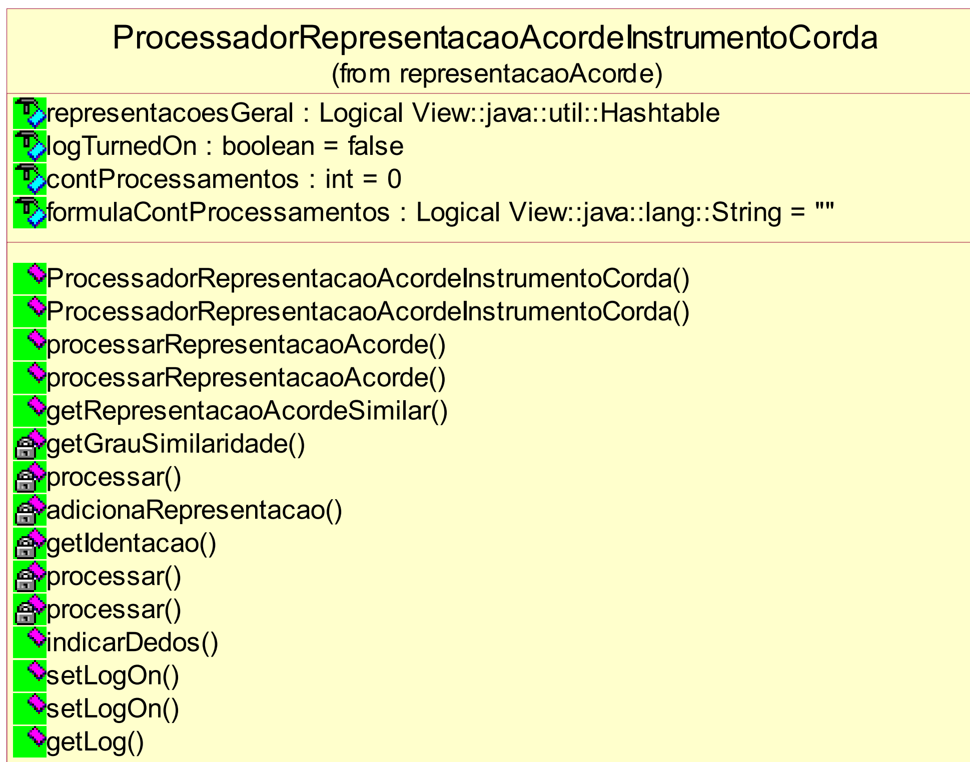



Figura 4.24: Classe ProcessadorRepresentacaoAcordeInstrumentoCorda.

Processando as representações em fases chega-se a um resultado mais rápido, porém o processo fica mais difícil de ser compreendido. Recapitulando: Uma cifra representa um acorde (um acorde pode ter mais de uma cifra) e este acorde possui diversas maneiras de ser executado no sistema. Dividimos estas “maneiras de executar o acorde” (representação do acorde) em duas categorias: representações básicas (somente com uma instância de cada nota do acorde) e representações completas ou processadas, onde podem existir dobramentos, duplicações e outros processamentos pré-configurados pelo usuário. A supressão é feita quando não é possível calcular nenhuma representação básica, portando na Fase 1.

As representações básicas, apesar de serem apropriadas para execução, talvez não sejam as mais adequadas para isto em virtude de características do estilo musical, opções do músico e as possibilidades que o instrumento oferece. Estas são processadas na primeira fase e nos dá uma boa idéia de qual representação escolher, antes dos processamentos mais demorados. Uma representação básica gera várias representações completas.

Os processamentos que podem ser efetuados pelo método "processarAcordeInstrumentoCorda()" estão expressos na classe RepresentacaoAcodeProperties e são passíveis configuração pelo usuário. São eles:

O dobramento da nota fundamental: Repete a nota fundamental;

**A duplicação da nota fundamental:** Repete a nota fundamental em uníssono[11];

**A triplicação da nota fundamental:** Permite repetir a nota fundamental até 3 vezes;

**O dobramento da terça:** Repete a nota representada pelo intervalo de terça (maior ou menor);

**O dobramento da quinta justa:** Repete a nota representada pelo intervalo de quinta justa;

**A duplicação da quinta justa:** Repete a nota representada pelo intervalo de quinta justa em uníssono;

**A supressão da quinta justa:** Omite o intervalo de quinta em função de um outro intervalo de maior prioridade; Esta opção só é considerada quando não é possível formar o desenho do acorde com todos os intervalos descritos na cifra, então a quinta é omitida para dar lugar a um intervalo mais importante.

**A consideração das oitavas das notas (escala diatônica):** Diferencia intervalos, por exemplo, de nova e segunda. Apesar de serem a mesma nota, teoricamente estão em oitavas separadas. Isto é constantemente desconsiderado no violão.

**Indicar o dedo que deve tocar cada nota:** Permite que o sistema sugira quais dedos usar na execução do acorde.

**Indicar possível pestana**: Autoriza o sistema a calcular representações com pestanas.

**Calcular as inversões:** Permite ao sistema calcular inversões do acorde mesmo que esta não esteja expressa na cifra.

No contexto deste trabalho, a duplicação de notas é a repetição da mesma nota (mesma oitava), e o dobramento a mesma nota, porém podendo ser em oitavas diferentes. Logo, considera-se a duplicação um caso do dobramento.

O processamento de uma representação completa, apesar de trabalhar com um número finito de possibilidades, é demorada se comparada com o cálculo da representação básica. A Figura 4.25 mostra como é feito o processamento dos dobramentos nas representações básicas. No exemplo, 3 processamentos estão configurados para serem executados e são representados por p1, p2 e p3. Note que a representação básica original "R" deve ser testada nos 3 processamentos e seus produtos testados pelos processamentos restantes.

[11] Mesma nota, na mesma oitáva.

Cada vez que um processamento conseguir gerar uma nova representação, esta deve ser adicionada, entretanto representações iguais devem ser descartadas e isso sobrecarrega o sistema.

Os processamentos são verificados em cima das cordas disponíveis da representação, e a ordem é relevante. Neste sentido, a opção de permitir o cálculo da inversão é considerada para definir se a busca começa a partir da nota baixo ou da corda mais grave do instrumento. Além desta verificação, é necessário verificar se o músico terá dedos suficientes para execução, se há necessidade do uso e se é possível a pestana. Ainda, deve-se verificar se serão sugeridos os dedos que devem ser usados.

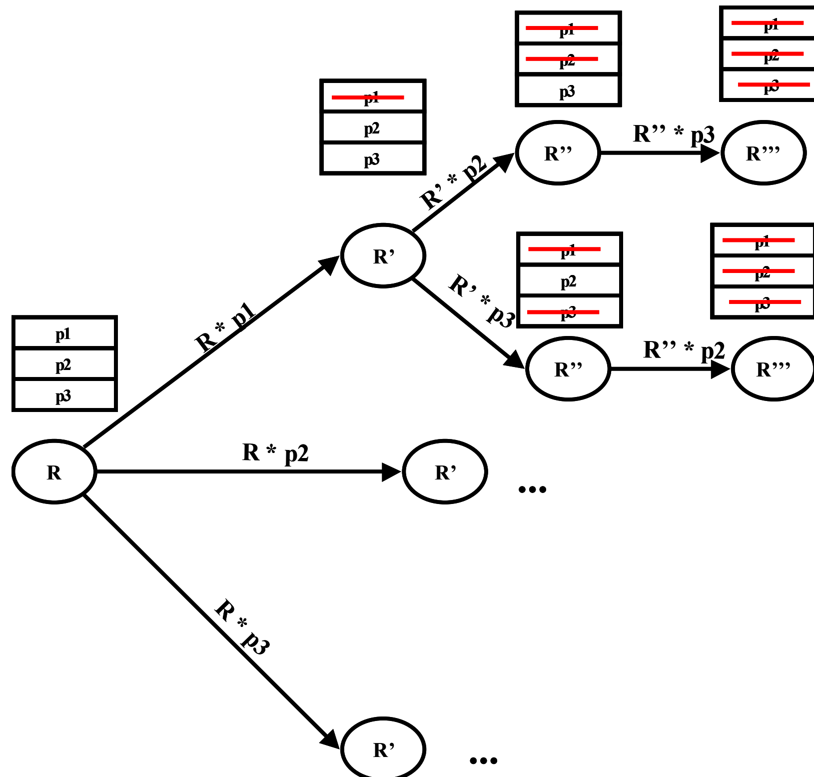



Figura 4.25: Processamento de dobramentos em reapresentações básicas.

Devido ao tempo de processamento necessário para se calcular todas as representações completas possíveis de um acorde, foi definida o método de cálculo da representação por fases. A quantidade de operações necessárias para se processar as representações básicas é dada pela Equação 1, onde n é a quantidade de processamentos a serem feitos.

$$F(n) = n * f(n-1) + 1$$

Equação 1

O Gráfico 1 mostra a quantidade de operações necessárias, conseqüentemente o tempo de processamento, para se calcular as representações básicas em função dos processamentos agendados. Só para ilustrar, em uma máquina Atlhon XP 1.8 e 256 Mb ram, foi preciso 16 ms para se calcular as 13 representações básicas de um Dó Maior e outros 62 ms para se processar uma das representações básicas (a mais fácil de ser executada pelo instrumentista). Esta representação básica processada gerou 5 representações completas. Se fossemos processar todas as representações de Dó Maior

para escolhermos a mais fácil, demoraríamos 263 ms, ou seja, 3 vezes mais. O acorde escolhido para este exemplo foi aleatório e pode existir uma variação entre acordes diferentes com mais ou menos representações básicas.

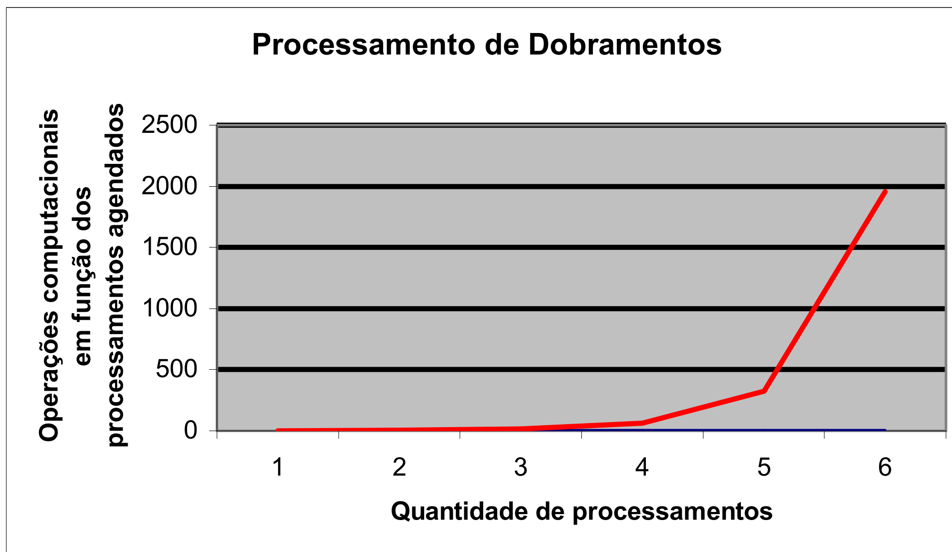



Gráfico 1: Gráfico dos Processamento dos Dobramentos

Como já foi visto, uma das formas de escolha da representação é por similaridade. A classe RepresentacaoAcordeInstrumentoCordaMaker, através do método getRepresentacaoAcordeInstrumentoCordaSimilar, usa a classe ProcessadorAcordeInstrumentoCorda para calcular a similaridade entre as representações. O objetivo final é escolher dentre um conjunto de representações básicas aquela que mais se assemelha a uma representação completa e então processá-la.

O fator de similaridade varia de 0 a 1, onde 1 é um desenho de acorde inteiramente igual a sua referencia. Acompanhe o exemplo: Dado um **Am** com o desenho 50-42-32-21-10 como acorde inicial e um **G** como o acorde seguinte na seqüência harmônica, o algoritmo irá primeiramente calcular as representações básicas do **G**. São elas: *63- 40- 20; 63- 40- 34; 63- 52- 40; 510- 49- 110; 63- 34- 23; 63- 52- 23; 45- 37- 17; 45- 34- 23; 63- 55- 34; 510- 37- 17; 510- 49- 37; 30- 23- 17; 63- 40- 17; 45- 23- 17; 63- 23- 17; 63- 37- 17; 63- 55- 17;*

A seguir, irá calcular o grau de similaridade de cada representação básica em relação a representação completa e retornará a mais similar. Neste caso, a escolha foi a representação básica 63-40-20 com 52% de similaridade. Este valor é o somatório da similaridade de cada posição da representação básica em relação a representação completa. Para cada posição da representação básica, a função de similaridade é calculada para as posições da representação completa que esteja na mesma corda, uma acima e uma abaixo, observando sempre a abertura máxima de dedos do músico. Os valores são alocados em uma tabela, como o exemplo visto na Tabela 4.2.

Tabela 4.2 : Cálculo de Similaridade

| | **-** | **50** | **42** | **32** | **21** | **10** |
|---|---|---|---|---|---|---|
| **63** | **-** | **.31** | | | | |

| **40** | | **.31** | **.62** | **.32** | | |
|---|---|---|---|---|---|---|
| **20** | | | | **.31** | **.62** | **.31** |

Quando não há interseção na coluna dos maiores valores de cada linha, é feita a média direta dos maiores valores de cada linha, neste caso, 0,31 + 0,62 + 62 = 0,52. Entretanto, há casos em que os maiores valores caem na mesma coluna e, sendo assim, o maior valor prevalece como regra geral. Uma exceção a essa regra é quando o menor valor é, o único da linha, logo, deve permanecer.

A função que calcula a similaridade da posição é dada por f(x) e f(y), sendo respectivamente para o calculo da similaridade entre posições que estejam na mesma corda e posições que estejam em cordas diferentes (acima ou abaixo). A fatorCasa é uma  é a diferença entre os trastes das posições quando não existem cordas soltas. Exemplo: 53 e 47, neste caso, 7 – 3 = 4 é o fatorCasa. Quando há cordas soltas, então o fatorCasa é dado pela diferença das médias dos trastes representação, desprezando cordas soltas, com peso de 0.5. Exemplo: 53-40-32, média: (3 + 2)/2 = 2.5 (arredonda para 2) e 63-52-13, média (3 + 2+ 3)/3 = 2.6 (arredonda pra 3). Diferença: 3-2 = 1 * 0.5 = 0.5;

*f(x) = 1.0-((1.0/instrumento.getAberturaDedo())*(fatorCasa))*

*f(y) = 0.5-((0.5/instrumento.getAberturaDedo())*(fatorCasa);*

Uma vez achado a representação básica mais similar, processam-se então as duplicações. Caso não seja possível um desenho de acorde com a polifonia especificada pelo usuário, processa-se a próxima representação básica mais similar.

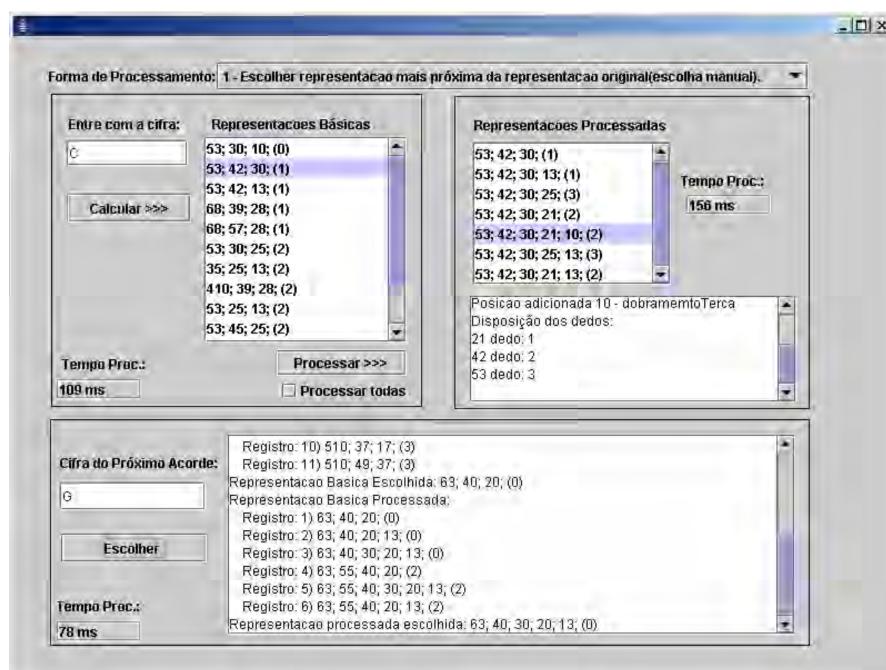



Figura 4.26: Simulador de teste da escolha das representações dos acordes.

A Figura 4.26 mostra um simulador construído para demonstrar o funcionamento da escolha do desenho do acorde baseado na similaridade entre eles. Observe que o acorde

inicial é um Dó Maior (C). Foi escolhida a representação básica 53;42;30 para ser processada as duplicações e, após processamento, foi definida a representação 53-42-30-21-10 como o desenho de acorde inicial. Suponha esta representação como a anterior em uma seqüência onde o Sol Maior (G) é o próximo acorde. O sistema, após processar todas a representações básicas, escolhe a 63-40-20 como a mais similar e processa-a, chegando à representação 63-40-30-20-13. Repare que ambas as representações possui a mesma polifonia, ou seja, 5 notas simultâneas.

A escolha por fases agiliza o processo, mas não garante que a representação tocada seja a mais fácil ou a que melhor soe, logo é comum haver divergência entre os desenhos de acordes que humanos escolheriam e os sugeridos pelo sistema. Os resultados serão apresentados no Capítulo 6.

## 4.3 Resumo do Capítulo

Os pacotes foram construídos visando o máximo de reuso e, por vezes, optou-se por soluções que extrapolam o contexto desta dissertação. Ainda, houve situações onde o contrário foi percebido, ou seja, generalizamos no pacote algoritmos que precisaram ser reescritos para o trabalho em questão. Um exemplo disto é o algoritmo de cálculo da representação do acorde, que precisou ser estendido para usar o padrão rítmico como variável.

Os pacotes formacaoAcorde e representacaoAcorde possuem código aberto e estão disponíveis para a comunidade desenvolvedora de software musicas em Java, Respectivamente, visam auxiliar no reconhecimento de cifras como acorde e no cálculo dos desenhos desses acordes.

# 5 CONSTRUÇÃO DO PROTÓTIPO

Este Capítulo visa apresentar a aplicação construída e seus detalhes de implementação.

Como dito anteriormente, foi utilizada a linguagem de programação Java na versão SDK 1.4.2. Para auxiliar no tratamento de áudio e MIDI duas bibliotecas foram utilizadas, o Java Sound (acompanha o SDK) e o jMusic 1.4.2. Ainda, foram construídas duas novas API´s como produto desta dissertação que também foram utilizadas na programação musical, br.ufrgs.in.lcm.formacaoAcorde e br.ufrgs.in.lcm.representacaoAcorde (Capítulo 4). A comunicação entre os agentes é feita através da tecnologia proprietária da Sun, RMI.

Para o desenvolvimento da interface do software, uma biblioteca que auxilia no desenvolvimento de interfaces gráficas foi construída e utilizada. Começaremos a explicação do protótipo por ela.

## *5.1* Interface Gráfica – Pacote Webfake

O desenvolvimento de uma interface gráfica em Java, normalmente, é mais trabalhoso e demorado quando comparada a outras linguagens. Este fato deve-se a flexibilidade dos pacotes gráficos Swing e AWT, que se por um lado potencializam a linguagem, por outro força o programador a escrever muitas linhas de código para atingir um objetivo relativamente simples, baixando sua produtividade.

Normalmente, há um conjunto de tarefas comuns no desenvolvimento de interfaces em Java. Um exemplo destas tarefas é a organização dos painéis (JPanel) em modelos para a construção de interface, ou seja, de que forma estão dispostos os painéis e que componentes gráficos os compõem. Esse modelo tende a ser padrão para todo o sistema. Exemplo: Podemos assumir que um menu de opções das interfaces Windows está localizado na parte superior da janela.

A definição de um modelo de interface toma tempo de projeto e desenvolvimento. Muitas vezes, não há necessidade de especificar este modelo visto que o grupo de desenvolvimento pode desejar a padronização de um modelo de interface visando a identidade visual de seus software. Buscando ganhar tempo de desenvolvimento destinado a definição do modelo de interface, este pacote, com um modelo genérico de interface, foi criado para auxiliar no desenvolvimento de interfaces Java.

O pacote implementa funcionalidades nativas, tais como: controle básico de navegação (voltar, avançar), controle de navegação dos *links*, delimitação de contexto navegacional, personalização da aparência de interface, implementação de menus comandos e barra de status.

A principal classe deste pacote é um painel de contexto que é estendido a cada tela do sistema. Este painel carrega informações gráficas, navegacionais e formas de tratar os eventos (comandos). A exibição deste painel se dá através de uma janela envoltória (frame externo) que contém uma barra de status e um menu lateral, ambos relativos ao painel de contexto.

A seguir serão apresentadas as principais classes que compões o pacote e a forma de sua utilização.

### 5.1.1 Conjunto de Classes

Os relacionamentos entre as principais classes do pacote desenvolvido, podem ser vistos na Figura 5.1. Observa-se que a classe principal deste diagrama é o ContextPanel, estando esse vinculado a Links e Commands que representam, respectivamente, as telas para qual se pode ir a partir do contexto em questão e as ações que podem ocorrer.

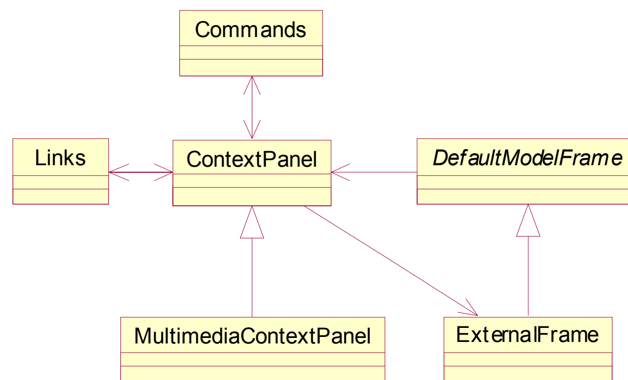



Figura 5.1: Diagrama de classes simplificado

Todas as telas do sistema devem ser subclasses da classe ContextPanel.

O ContextPanel estende a classe JPanel do pacote Swing, tal qual o DefaultModelFrame estende a classe Frame, também do pacote Swing. A relação entre a ContextPanel e DefaultModelFrame é a mesma existente entre suas superclasses, ou seja, um Frame exibe o Panel.

A classe abstrata DefaultModelFrame define o modelo básico de interface, composto de um painel lateral com *links*, comandos do contexto e comandos de sistema, uma barra de status e o painel de contexto. A cada troca de contexto, operações de vídeo são processadas a fim de manter a correta exibição da interface. Esse processamento é implementado também pela classe DefaultModelFrame.

A subclasse concreta ExternalModelFrame implementa os métodos definidos por sua superclasse (DefaultModelFrame) e a navegação entre contexto. O processo de união de um ContextPanel e um DefaultModelFrame também é processamento da ExternalFrame.

A seguir, as principais classes do pacote são descritas.

**DefaultModelFrame:** A classe abstrata DefaultModelFrame estende a classe swing.JFrame e possui a mesma função que esta. O ideal é que exista somente uma tela

principal, pois além da economia de memória, fica mais fácil gerenciar a interface. O frame é inicialmente dividido em 3 componentes:

- Status Bar: Exibe ajuda ou mensagens de pouca relevância ao usuário;
- Painel conteúdo: Varia de acordo com o contexto e é responsável pelas informações exibidas;
- Painel Opções: Exibe as opções permitidas ao usuário, podendo ser "Links"ou "Commands";

**ExternalFrame**: Classe concreta do DefaultModelFrame, a ExternalFrame além de implementar as operações definidas por sua superclasse, também gerencia a navegação entre os diversos contextos exibindo, ao usuário, o caminho percorrido até chegar ao local onde ele se encontra no sistema. As operações de "voltar" e "avançar" também são implementadas nesta classe.

**OutsideFrame**: Esta biblioteca foi projeta para possuir uma única janela no sistema onde os conteúdos vão sendo trocados a medida do necessário. Após algumas aplicações de testes, foi constatada a necessidade de possuir uma segunda janela vinculada a principal (estilo janela de dialogo) que pudesse servir de entrada, saída ou troca de dados com a janela principal. Esses conceitos foram implementados na OutsideFrame, uma subclasse do DefaultModelFrame e, portanto, tão poderosa quanto.

**ContextPanel**: É a parte da ExternalFrame que está em constante mudança em função do contexto. No painel de conteúdo, definem-se os "*links*", "comandos" ou qualquer outra função necessária, no contexto que o painel representa.

O ContextPanel é carregado em tempo de execução e está associado a um dispositivo que controla sua manutenção ou despejo da memória, o ClassFactory.

**ClassFactory:** É responsável pela gerencia dos objetos de interface (ExternalFrame e ContextPanel) instanciados em tempo de execução, provendo um controle sobre o desperdício de memória. Com seus métodos estáticos é possível registrar ou instanciar objetos gráficos sem a preocupação de saber se já existem ou não.

**Combiner**: A tela que é exibida ao usuário. É uma combinação de dois objetos: ExternalFrame e ContextPanel; Para que estes elementos pudessem ser mesclados de uma forma transparente e eficiente esta classe foi criada neste trabalho; Com seus métodos estáticos, esta classe cria / modifica a tela que o usuário verá.

**Links:** São os responsáveis pela navegação do usuário no sistema. Com eles é possível mudar o contexto para um menu ou qualquer outra tela que seja relevante no contexto em que o usuário se encontra. Alojados no painel de opções da tela principal, devem ser definidos durante o projeto do software e incluídos na construção dos painéis de conteúdos.

**Commands**: Guardam as ações relativas a cada um dos contextos. Exemplo: Se é necessário pegar uma informação em uma JTextBox e processá-la, esta ação será codificada em um Command e atribuída a um contexto.

**ObjectPass**: Esta classe funciona como um "guarda volumes" entre contextos que não podem se enxergar.

**MultimidiaContextPanel**: Subclasse da ContextPanel que incorpora recursos multimídia para controle de mídias temporais.

### 5.1.2 Modo de Funcionamento

Ao se optar pela utilização deste pacote de interface para auxiliar na construção do sistema é necessário, em tempo de projeto, definir a estrutura de navegação, ou seja, como os contextos estão interligados e como será a comunicação entre eles.

A Figura 5.2 mostra uma aplicação composta por três contextos: Tela1, Tela2, Tela3. Todos os contextos se inter-relacionam, ou seja, a partir de qualquer um deles é possível chegar nos outros dois. Estas relações são indicadas pelos *links* mostrados no painel lateral.

No contexto "Tela 2" existe ainda um comando auto-explicativo chamado "Limpar Campo".

Tanto os *links* como os comandos possuem mensagens de ajuda que são exibidas na barra de status quando o usuário passa o mouse sobre a opção desejada.

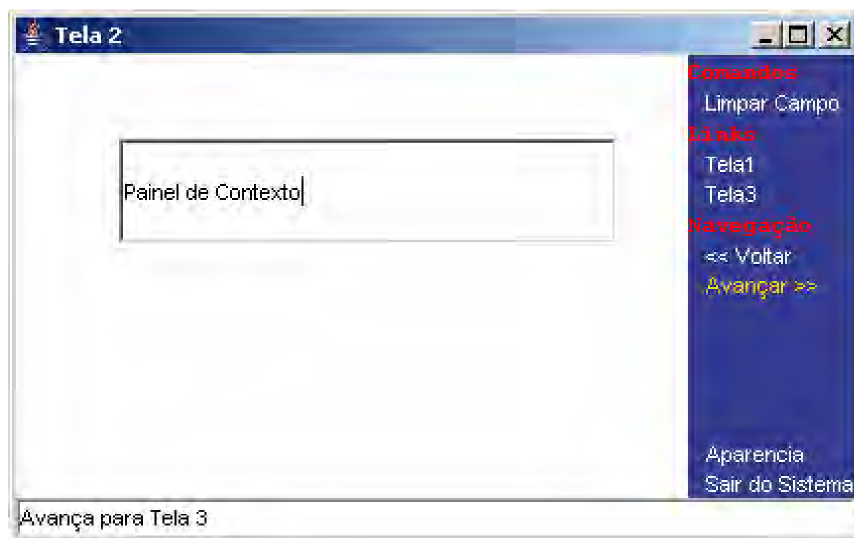



Figura 5.2: Aplicação Exemplo

Abaixo dos *links*, é possível ver os comandos de "voltar" e "avançar". Estes comandos aparecem à medida que o usuário navega no sistema, tal qual um *browser* faz com as páginas navegadas na web.

Na parte inferior do painel lateral se vê os comandos de sistema "Aparência" e "Sair do sistema". Esses comandos ficam ativos independentes do contexto que está sendo exibido.

No centro, em fundo branco, fica o painel de conteúdo (instancia do ContextPanel). Todo painel de conteúdo tem um título que é exibido na barra de título da janela.

Para que essa pequena aplicação de exemplo fosse feita, somente 3 classes precisaram ser criadas: Tela1, Tela 2 e Tela 3. Todas estendem o ContextPanel, ou seja, são suas filhas.

Em cada uma dessas classes foram definidos os *links* e os comandos pertinentes as funcionalidades dos contextos que representam. Veja o trecho de código abaixo:

```
1.  public class Tela1 extends MultimediaContextPanel {
2.  JTextField txCampo = new JTextField();

3.  public Tela1() {
4.  super("Tela 1");
5.  configuraTela();
6.  }

7.  private void configuraTela(){
8.  addLink("Tela2","Vai para Tela2","Tela2");
9.  addLink("Tela3","Vai para Tela3","Tela3");

10. addCommand("Limpar Campo", " teste", new java.awt.event.MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
11. txCampo.setText("");
12. }
13. });
14. }
```

Observa-se na linha 1 a extensão da classe MultimídiaContexPanel (subclasse da ContextPanel). Na linha 4, informamos a classe pai o título deste contexto. As linhas 8 e 9 mostram como definir *links* através no método addLink(nomeLink, ajuda, nomeClasse). Já a linha 10 mostra como adicionar um comando.

## 5.2 Adaptações do Pacote de Interface ao Protótipo

Basicamente uma única classe precisou ser estendida, a PolvoExternalFrame. Esta classe ganhou uma estrutura de árvore que serve para organizar os agentes como mostra a Figura 5.3.

Figura 5.3: Estrutura de árvore que organiza os agentes do sistema.

Cada agente do sistema gera sua interface(ContextPanel) e repassa esse objeto ao Agente CS, que o exibe.

Ainda, foram adicionados comandos permanentes, como os de criação dos agentes ME, MD e CS. Esses comandos só ficam visíveis quando a composição já foi criada.

## 5.3 Componente para Desenho do Padrão Rítmico

Não existe um componente gráfico no Java para a captura de padrões rítmicos. Dessa forma, baseado no conceito de interface conhecida como *Piano-roll*(ROADS, 1996), foi desenvolvido um componente gráfico que estende as funcionalidades de um JTable e sua forma de renderização.  Este componente é responsável não somente pela captura e

exibição dos padrões rítmicos para violão, mas também pela interpretação desses dados musicais que serão utilizados pelo Agente MD.

A Figura 5.4 mostra o componente sem nenhum padrão rítmico definido. As linhas verticais representam as vozes do acorde, ou seja, neste caso o padrão rítmico vai trabalhar com acordes com uma polifonia de 4 notas, sendo a 4 a mais grave e a 1 a mais aguda (conforme indicação do rótulo). As colunas representam os tempos e são usadas para dar idéia de duração a um evento musical. Ao lado esquerdo do componente, uma barra de ferramentas exibe as opções do usuário, são elas: Inserir (*insert*) um evento musical, Apagar(*delete*), Dividir (*Split*), Mover(*Move*) e Selecionar. Os comandos de Zoom não foram implementados nesta versão.

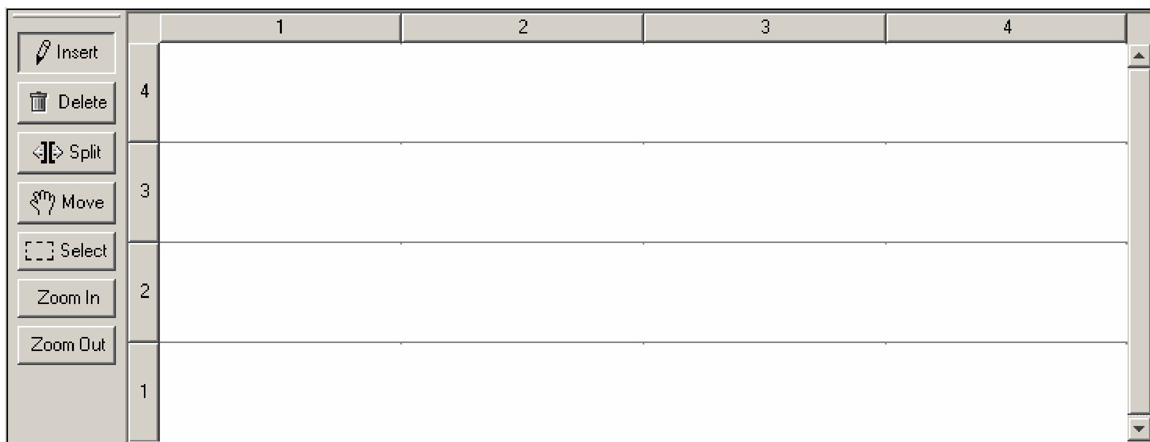



Figura 5.4: Componente para criação do padrão rítmico.

A Figura 5.5 mostra um padrão rítmico de 6 tempos e polifonia 5. Nele temos 8 eventos musicais. Na linha de rótulo 5 (primeira de cima pra baixo), está a nota mais grave. Neste momento o usuário não sabe qual nota é essa, até porque quem vai prover esta informação é o Agente ME quando ele calcular o desenho do acorde. No primeiro tempo da linha 5 se inicia um evento musical que só irá parar de soar no quarto tempo. Essa é sua duração. Mesmo pensamento se repete nos outros eventos musicais. Note, entretanto, os que eventos da linha 1 (a primeira de baixo pra cima), são dois e não um só, isso fica claro pela marcação da borda que separa os dois.

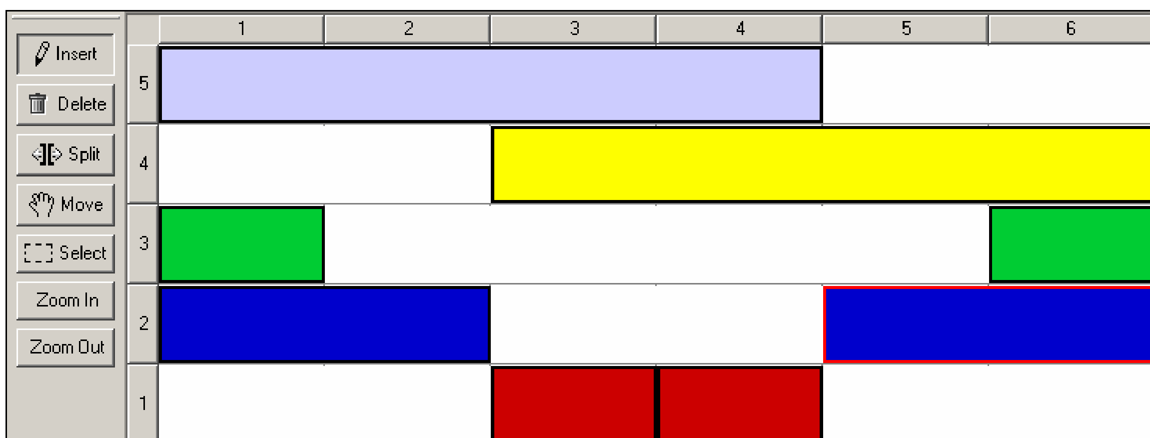



Figura 5.5: Exemplo de padrão rítmico.

## 5.4 Criando uma composição

No contexto desse trabalho, uma composição é o ambiente onde os agentes executam suas tarefas. Para este protótipo, optou-se por criar composições localmente, ou seja, todos os agentes rodam na mesma máquina. Essa decisão baseou-se no fato de que nem sempre uma estrutura de rede está disponível aos músicos no ato da criação de sua peça musical ou geração de material sonoro para a mesma.

A Figura 5.6 mostra a tela de criação da composição que possui somente dois atributos opcionais: nome e compositor. Após preencher os campos, clica-se na opção “criar composição” no lado direito da tela. Note ainda que na tela da composição existem contadores para cada tipo de agente que está vinculado a composição. A composição toda pode ser persistida em arquivo e recuperada para continuação do trabalho.

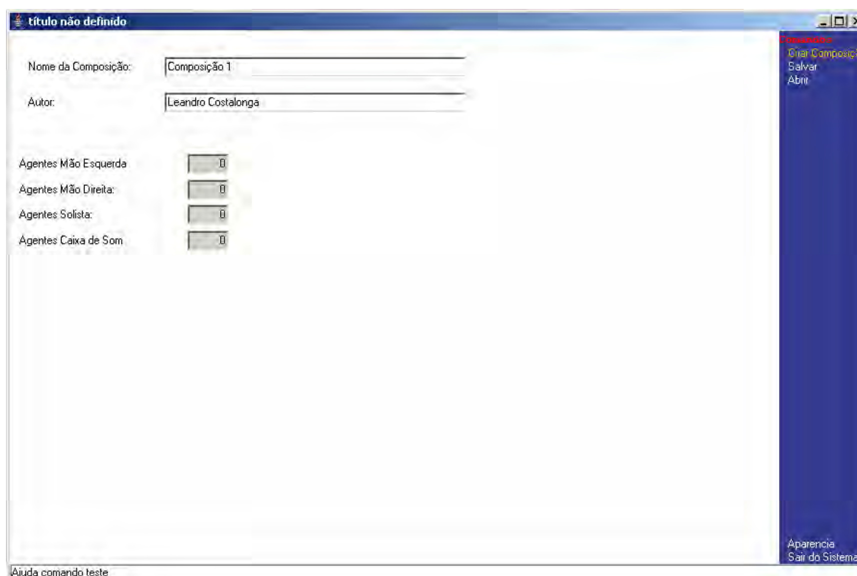



Figura 5.6: Tela de criação da composição.

## 5.5 Criando Agente ME e Definindo a Harmonia

Após a criação da composição, as opções para a criação de agentes ficam disponíveis embaixo da árvore que os mostra. Para criar um Agente ME, basta clicar na opção “Novo ME” e dar um nome ao Agente ou aceitar o nome sugerido pelo sistema.Após sua criação, ele aparece no primeiro nível da árvore. Basta clicar nele, para que sua interface seja exibida conforme mostra a Figura 5.8.

A harmonia é definida no Agente ME através da criação de partes harmônicas, que por sua vez, são criadas adicionando cifras às mesmas, deixando um espaço entre elas. Ao salvar a parte, todas suas cifras são analisadas e convertidas em intervalos musicais. Caso uma cifra não seja reconhecida, o usuário é informado do problema. Se o usuário tiver dúvida na escrita do acorde na notação de cifragem vigente ou das notas que compõem o acorde, ele pode solicitar informações sobre a cifra como mostra a Figura 5.7 para um acorde sol diminuto.

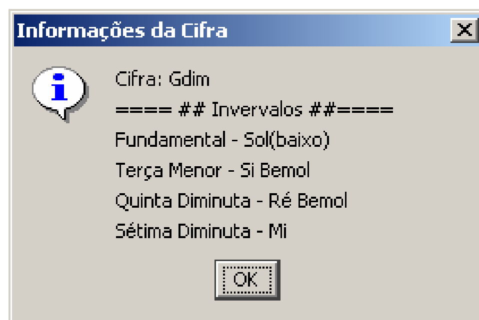



Figura 5.7: Informações de um Gdim

No exemplo da Figura 5.8, somente uma parte foi criada e adicionada à composição. Essa parte intitulada “Parte1” pode aparecer repetidamente na composição, sem a necessidade de se informar novamente as cifras.

O campo BPM indica a velocidade com que será tocada a previsão da parte. É importante ressaltar que este BMP é relativo somente a opção “Ouvir previsão da Parte”, quando os acordes são tocados seqüencialmente sem considerar nenhuma informação rítmica.

Não é necessário gravar todas as partes na harmonia, alias, estas somente serão gravadas quando o botão “Inserir parte na posição X” for acionado. A posição referida é ao lugar (na ordem) onde será inserida a parte. Qualquer alteração na parte é refletida na composição da harmonia.

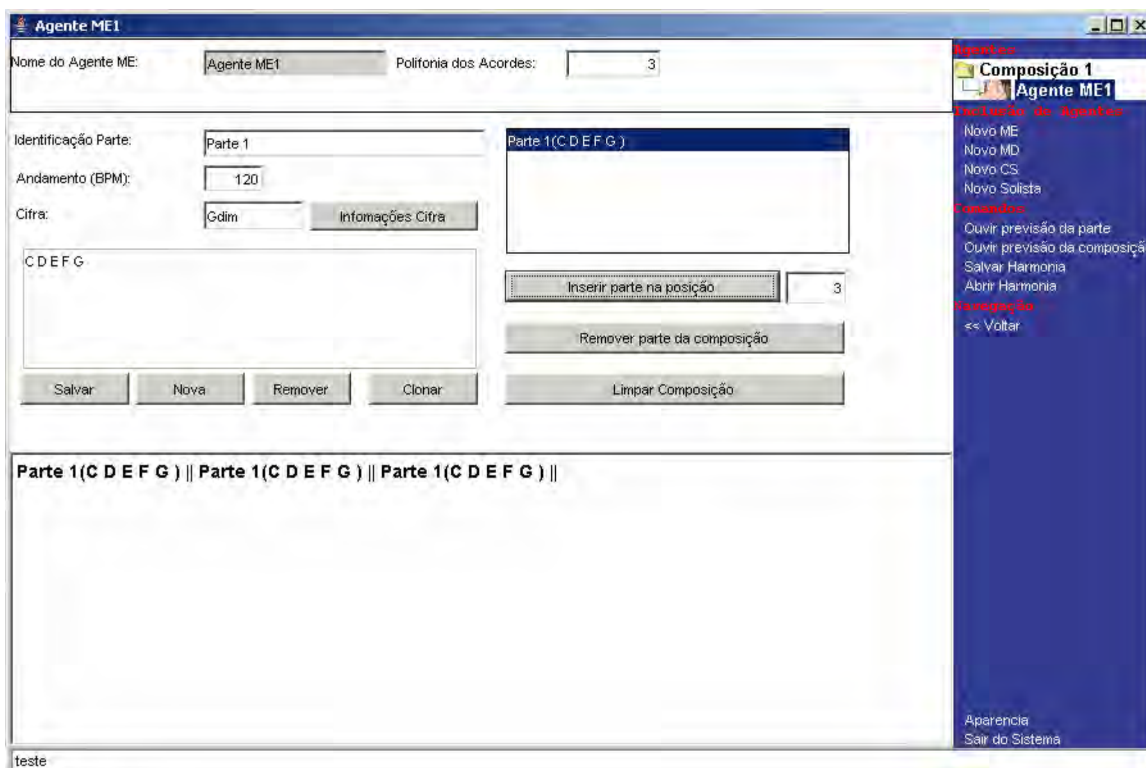



Figura 5.8: Tela do Agente ME.

Os comandos disponíveis para o usuário no Agente ME são:

**Ouvir previsão da parte:** Permite ouvir a seqüência harmônica da parte selecionada com a polifonia mínima em cada acorde, mas sem informações rítmicas.

**Ouvir previsão da composição:** Permite ouvir a seqüência harmônica da composição com a polifonia mínima em cada acorde, mas sem informações rítmicas.

**Salvar harmonia:** Grava em arquivo a harmonia para uso futuro em outra composição;

**Abrir Harmonia:** Carrega os dados do arquivo relativo a harmonia para o sistema.

O agente ME é o responsável pelo reconhecimento das cifras, cálculo do acorde e cálculo do desenho do acorde. A maior parte de como é feito isto já foi descrito no Capítulo destinado a explicação dos pacotes de reconhecimentoAcorde e representacaoAcorde, entretanto existe algumas especificidades em como o Agente ME faz isso e será tratada mais adiante neste Capítulo.

Pode-se ter quantos agentes ME quanto necessário em uma composição, sendo que as harmonias definidas serão tocas de maneira sobreposta. Para se tocar uma harmonia, deve-se ter vinculado no mínimo um agente MD a um agente ME.

## 5.6 Criando o Agente MD e os Padrões Rítmicos

Um agente MD deve estar vinculado a um Agente ME, logo não basta que se clique em "Novo MD". Antes, deve-se escolher a qual Agente ME o mesmo deve estar vinculado. Após a criação é possível visualizar o novo Agente MD abaixo do agente ME, no segundo nível da árvore, como mostra a Figura 5.9.

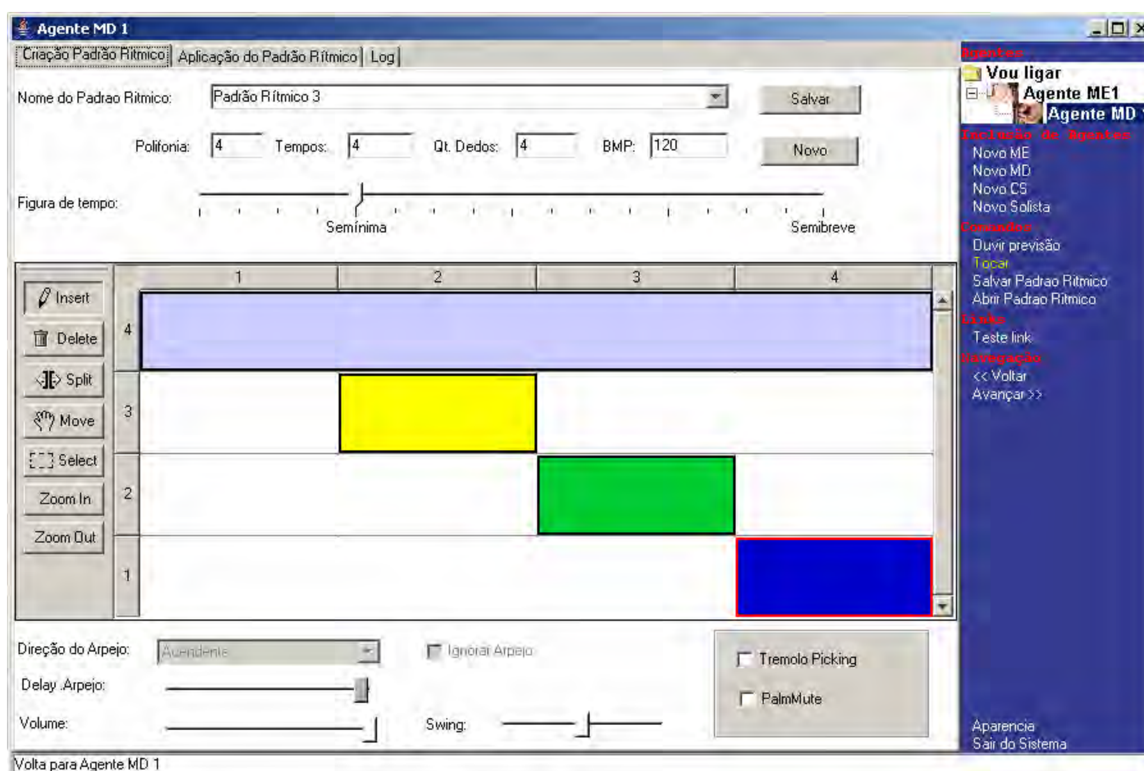



Figura 5.9: Tela do Agente MD

Para que possa visualizar a interface do agente MD, e criar os padrões rítmicos, é necessário clicar sobre o mesmo na árvore. A interface do agente MD está dividia em 3 partes:

**Criação Padrão Rítmico:** permite a criação do padrão rítmico e a configuração do mesmo.

**Aplicação do Padrão Rítmico:** Permite vincular os padrões criados as partes da harmonia, obtidas do agente ME.

**Log:** Mostra os desenhos de acordes escolhidos para cada acorde da harmonia.

### 5.6.1 Definindo Padrões Rítmicos

A Figura 5.9 mostra a tela onde um padrão rítmico foi criado. Um agente MD pode criar vários padrões rítmicos. São propriedades de um padrão rítmico:

**Nome:** Nome com que será identificado na hora da associação com a harmonia. Apesar do sistema aceitar nomes iguais, evite.

**Polifonia:** Número de notas simultâneas que o desenho de acorde deve ter. Isso reflete no número de linhas do componente gráfico de definição do padrão rítmico. Ao alterar esse parâmetro o padrão rítmico desenhado se apaga. Opcional quando o número da polifonia exceder o número de dedos disponíveis pra execução.

**Tempos:** O padrão rítmico se repete para cada acorde. A duração desse padrão é dada pelo número de tempos multiplicado pelo valor da Figura de Tempo. O número de tempos é o mesmo das colunas do componente gráfico de definição do padrão rítmico. Ao alterar esse parâmetro o padrão rítmico desenhado se apaga.

**Figura de tempo:** Define a duração de cada uma das colunas. O padrão é uma semínima que equivale a 1; Quanto mais alto o valor da figura de tempo, mas lenta é a execução da música.

**Quantidade de dedos:** Número de dedos da mão direita. Esse atributo auxilia no processo de decisão da escolha do desenho do acorde, que é feito pelo agente ME. A lógica é simples: se a quantidade de ataques simultâneos do padrão rítmico em um determinado tempo é maior que a quantidade de dedos disponíveis para "pinçar" as cordas, logo, trata-se de um arpejo rápido, ou seja, um único movimento unidirecional, provavelmente usando uma palheta. Nestes casos, as cordas do desenho do acorde precisam ser contíguas, ou seja, não pode haver cordas soltas entre duas posições, pois não seria possível saltá-las rapidamente.

**BPM:** Usado pra calcular a velocidade quando o usuário solicita a audição da previsão do padrão rítmico. Usado somente na previsão e não na composição.

As propriedades acima citadas são do padrão rítmico como um todo, porém para cada evento musical deste padrão rítmico existem atributos que podem ser trabalhados para se obter a sonoridade desejada pelo usuário. São eles:

**Volume:** Volume com que vai soar a nota. Usado para trabalhar a acentuação do padrão rítmico, visto que não há a idéia de compasso no protótipo.

**Swing:** Permite antecipar ou atrasar um evento musical em até 50% do valor da figura de tempo, com isso conseguimos uma precisão maior no tempo e inserir sincopes para dar uma sensação mais humana e menos mecânica ao ritmo.

**Direção Arpejo:** Um arpejo é quando dois ou mais eventos musicais começam no mesmo tempo. Tão logo o usuário selecione estes eventos, as opções "Direção Arpejo", "Velocidade Arpejo" e "Ignorar Arpejo" aparecem. A direção arpejo indica que as

cordas começarão a serem tocadas de cima pra baixo no violão (arpejo ascendente) ou de baixo pra cima (arpejo descendente).

**Delay do Arpejo:** Indica o tempo entre cada nota do arpejo. Quanto maior, mas lento será o arpejo. Esse valor é proporcional a figura de tempo vigente.

**Ignorar Arpejo:** Quando um arpejo é identificado pelo sistema, a propriedade de swing é ignorada em função do *delay* do arpejo que é calculada automaticamente. Se for desejo do usuário trabalhar o swing em um arpejo ele deve clicar esta opção.

**Tremolo Picking e Palm Mute:** Efeitos que serão implementados no futuro.

Para se obter uma idéia de como irá soar o padrão rítmico, pode-se solicitar a audição da previsão através do comando "Ouvir previsão". Será executado sobre a afinação básica (todas cordas soltas) do instrumento. Ainda são comandos:

**Salvar padrão rítmico:** Permite persistir o padrão em arquivo para uso futuro.

**Abrir padrão rítmico:** Carrega dados do arquivo para utilização na composição.

### 5.6.2 Associando os Padrões Rítmicos à Harmonia

A associação do padrão rítmico se dá em uma tela separada do contexto da criação, como mostra a Figura 5.10. Nela é possível visualizar os padrões rítmicos criados e as partes harmônicas recuperadas do Agente ME. Cada parte harmônica deve ter somente um padrão rítmico a ela associada. Entretanto, o mesmo padrão rítmico pode estar associado a várias partes harmônicas. Para se estabelecer uma associação entre o padrão rítmico e a uma parte harmônica, deve-se selecionar o padrão rítmico na lista da esquerda e o(s) padrão(ões) rítmico(s) da direita (use a tecla *CTRL* ou *Shift* para selecionar mais de um), em seguida clique no botão "Gravar".

Quando se faz a associação entre os padrões rítmicos e a harmonia, uma cor é sorteada para esta associação e representada na composição. Desta forma é possível visualizar se todas as partes harmônicas possuem um padrão rítmico. O padrão rítmico se repete para cada acorde da parte harmônica.

Depois de definida a associação é possível solicitar a execução da composição através do comando "Tocar". O comando "Tocar", dispara o processo de comunicação entre os agentes ME e MD visando chegar aos desenhos de acorde mais propícios aos padrões rítmicos. Conseguindo pelo menos uma representação de cada acorde no seu padrão rítmico, inicia a execução. A cada solicitação, todo o processo de escolha é refeito.

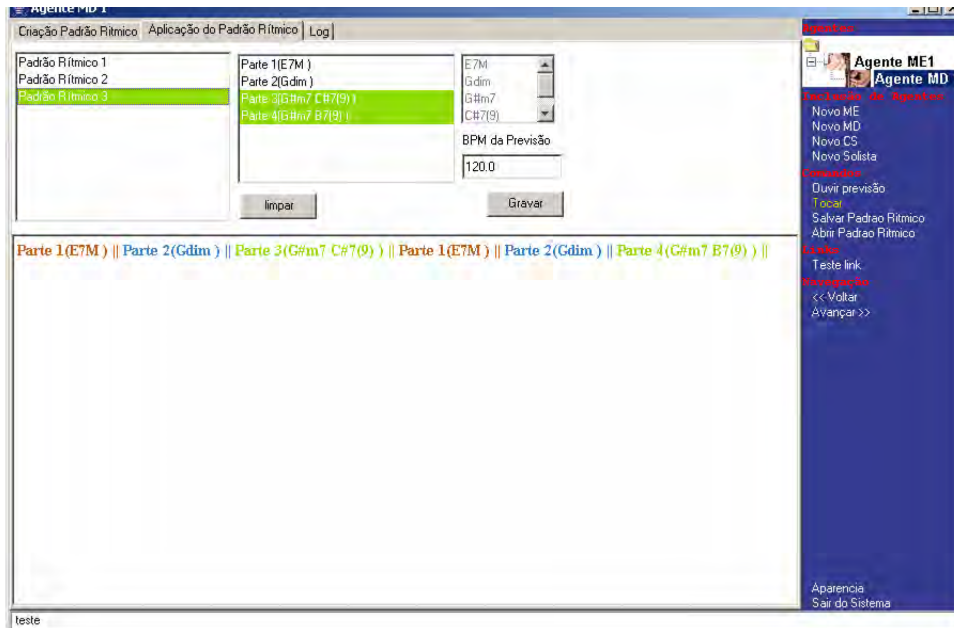



Figura 5.10: Tela da aplicação do padrão rítmico a harmonia.

A Figura 5.11 mostra a tela onde é feita a execução e a visualização das notas, em uma notação alternativa similar a notação tradicional porém, sem a figuras de tempo, sendo a duração representada pelo comprimento dos traços. Essa notação é proveniente do pacote jMusic, usado para auxiliar no tratamento de MIDI. Na tela de execução é possível: tocar, parar e exportar a música no formato MIDI.

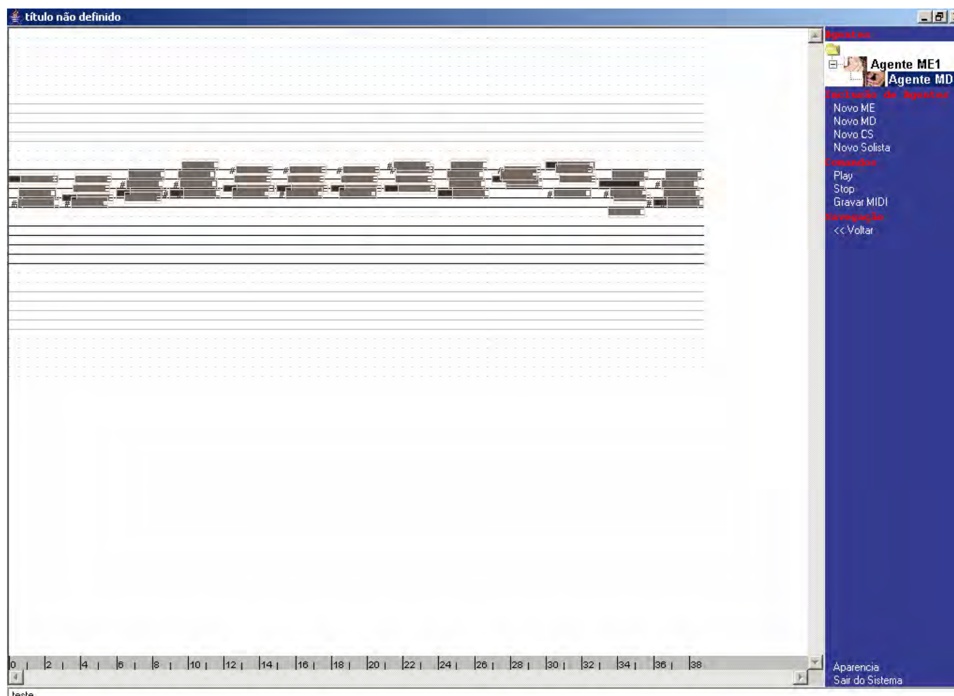



Figura 5.11: Tela de execução da composição.

### 5.6.3 Visualizando as decisões do sistema

Na tela de "*Log*" é possível visualizar quais desenhos de acordes foram escolhidos para um acorde, em função das restrições do usuário e características do padrão rítmico. Dois pontos valem ser ressaltados sobre esta decisão.

**Escolha por Similaridade:** A partir do segundo acorde, o sistema automaticamente procura por desenhos de acorde que mais se equivalem ao desenho anterior e que satisfaça aos requisitos do padrão rítmico. Neste caso os desenhos possíveis são ordenados pela facilidade de execução, ou seja, são paramentos respectivamente: menor abertura de dedos, menor quantidade de dedos, maior número de cordas soltas e proximidade da cabeça do violão. Não é calculada a condução de vozes (FILHO, 2005).

**Escolha do primeiro desenho de acorde:** A escolha do primeiro desenho de acorde pode ditar toda a região que os acordes que seguem vão tocar (altura), por isso, nem sempre o desenho de acorde mais fácil é o melhor a ser escolhido como primeiro, visto que o mesmo pode ser em uma região muito aguda. Portanto, para o primeiro desenho de acorde foi implementado um algoritmo que escolhe o desenho cuja média da soma dos trastes seja a menor, com isso escolhe-se os desenhos mais próximos da cabeça do violão.

A Figura 5.12 mostra quais os desenhos de acordes foram escolhidos para cada acorde. Muitas vezes, para o mesmo acorde podem existir dois desenhos diferentes em função do acorde que o precede.

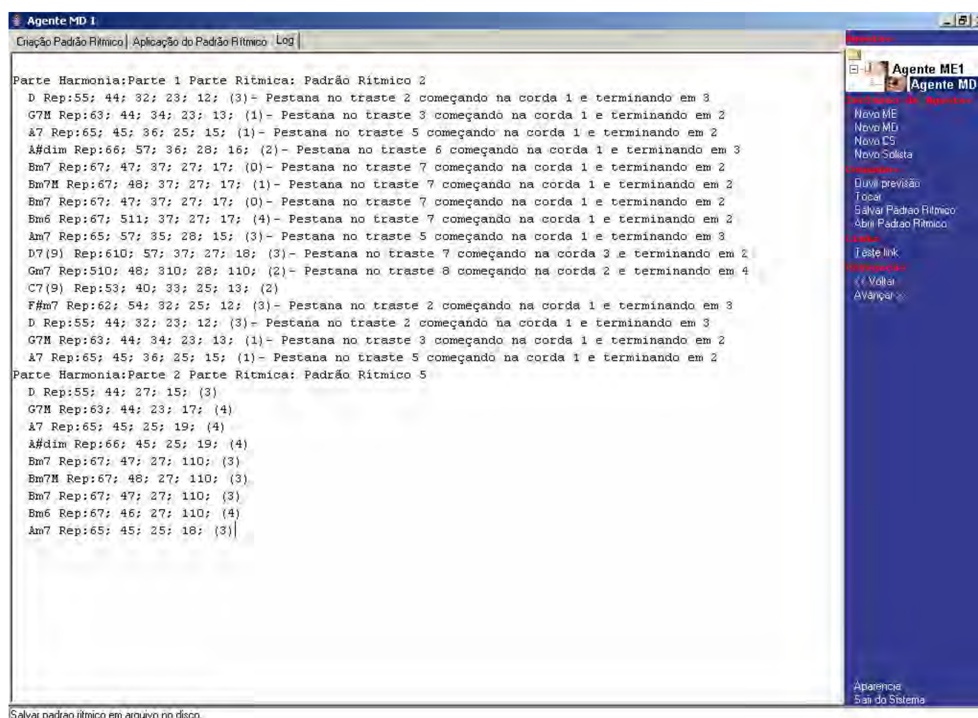



Figura 5.12: Tela de *Log*.

## 5.7 Ouvindo os agentes pela "Caixa de Som"

O sistema permite a criação de vários Agentes ME, cada um com vários Agentes MD associados. Para ouvir estes agentes tocando em conjunto é necessário que os eventos

sejam *mixados* pelo Agente CS. A Figura 5.13 mostra a interface simples do Agente CS, que contém somente uma tabela dos agentes no sistema.

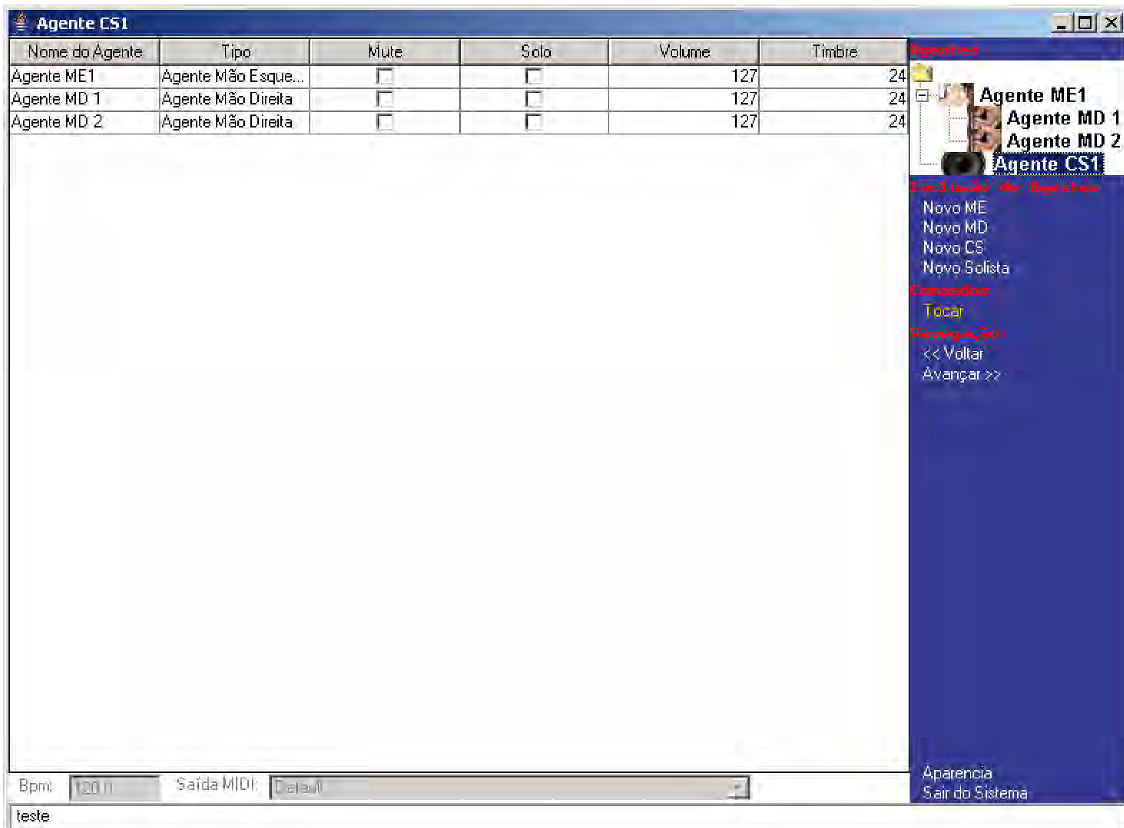



Figura 5.13: Tela do Agente Caixa de Som

Através da tabela é possível deixar algum agente mudo, tocando sozinho (solo), alterar seu volume e principalmente seu timbre (número MIDI). Toda alteração feita no agente ME se propaga para os agentes MD associados a ele. Pode-se ter quanto Agentes CS se desejar no sistema e com isso testar várias possibilidades de execução.

Na parte inferior da interface existem os campos de BPM (velocidade) global da composição e do dispositivo MIDI que irá sintetizar os eventos MIDI. Estas opções não estão disponíveis nessa versão.

Um único comando está disponível: Tocar. É essa sua função!

## 5.8 Resultados

Podemos dizer que, após os primeiros testes de utilização do sistema, o protótipo se mostrou eficaz e funcional, porém com muitas considerações.

Por ter sido feito em Java não existem restrições quanto a sua portabilidade e uso na web, entretanto apresentou algumas inconformidades em plataformas diferentes (Mac e Linux) da plataforma de desenvolvimento (Windows), principalmente no processamento de vídeo.

Muitas melhorias estão previstas e são relatadas nos trabalhos futuros, destacam-se:

- Mecanismo para melhor controle do tempo;

- Visualização gráfica das representações rítmicas;

A escolha dos acordes considera simplesmente a facilidade na transição entre os desenhos e não a condução de vozes, algo desejável para as próximas versões.

Questões de usabilidade de interface não foram consideradas nesta versão do protótipo, visto que não era o foco do trabalho.

Algumas amostras foram geradas por colaboradores do Centro de Música Eletrônica e violonistas e os arquivos estão no CD em anexo.

## 5.9 Resumo do Capítulo

Este Capítulo teve o objetivo de explicar o funcionamento e detalhes de implementação do protótipo.

O objetivo do protótipo foi validar a idéia de uma comunidade de agentes musicais interagindo e tomando decisões visando a execução de uma composição no violão. Este objetivo foi atingido apesar do protótipo precisar visivelmente de melhorias para se tornar um produto estável. Atualmente, o sistema possui em torno de 117 classes se somado todos os pacotes. Este elevado número de classes pode trazer dificuldade no gerenciamento e conseqüentes problemas, que serão identificados resolvidos futuramente

O protótipo já está sendo usado para geração de material sonoro visando composições por compositores do Centro de Música Eletrônica- CME. Também é possível usá-lo com fins didáticos no ensino do violão e conceitos rítmicos.

# 6 CONCLUSÃO

Foi apresentado neste trabalho um sistema multiagente capaz de simular performances musicais no violão considerando aspectos humanos, porém não se restringindo a eles. Foram descritas as etapas de desenvolvimento desde a idealização do sistema até sua implementação. Os resultados serão vistos ainda neste Capítulo.

Uma importante parte na construção do sistema foi a criação das bibliotecas para reconhecimento das cifras, cálculo do acorde e conseqüente cálculo do desenho do acorde. A abordagem multiagente foi especialmente útil no cálculo do desenho do acorde.

Quatro agentes foram modelados, entretanto somente três foram efetivamente implementados: Agentes Mão-Esquerda, Mão-Direita e Caixa-de-Som. O destaque nas interações fica entre os agentes ME e MD que, devido a interdependência nas suas atividades, precisam trabalhar juntos para que o desenho do acorde satisfaça as necessidades de ambos.

As contribuições, resultados obtidos, limitações e trabalhos futuros são descritos a seguir.

## 6.1 Contribuições e Resultados Obtidos

O sistema pode contribuir na execução das seguintes atividades:

Criação da parte de violão em uma composição por compositores que não tocam o instrumento.

- Execução de uma música com alto grau de dificuldade.
- Auxílio na educação musical, principalmente de conceitos rítmicos e de formação de acordes;.
- Experimentos musicais diversos, principalmente trabalhando a polirritmia, características do instrumento e extrapolação das limitações de execução humanas.
- Sugestão dos desenhos de acordes e suas transições, dada uma harmonia e o padrão rítmico. Uma espécie de dicionário de acordes dinâmico.

O sistema vem sendo testado e avaliado por alunos do curso de música da UFRGS e entusiastas da música em geral. A maior parte dos avaliadores conhece e toca violão com diferentes graus de habilidade, porém todos têm um bom conhecimento de informática. Os resultados são expressos em forma de material musical e pode ser ouvido no CD em anexo.

Algumas limitações relatadas sobre o sistema foram sendo implementadas nas novas versões, e portanto não serão mencionas. Somente as limitações mais conceituais são citadas e suas soluções estão previstas nos trabalhos futuros. São elas:

- Possibilidade de escolher o desenho do acorde em certas partes da composição;
- Visualização dos desenhos de acorde sobre a imagem do instrumento;
- Externalização das partes configuráveis do sistema para que o usuário possa fazê-lo a qualquer momento;
- Captura do padrão rítmico por instrumentos MIDI;
- Melhoria do mecanismo de associação dos padrões rítmicos à harmonia;
- Criação de bibliotecas de padrões rítmicos.

Quando os elementos musicais são tratados separadamente, torna-se difícil o posicionamento do "tempo", pois ele está presente tanto no ritmo (tamanho do padrão rítmico e figura de tempo) como na harmonia (repetição do acorde na seqüência). Isso dificulta a execução de uma música conhecida pelo sistema, tal qual um intérprete, pois exige do usuário a capacidade de trabalhar a criação das partes harmônicas e rítmicas em locais diferentes. Portanto, o sistema não é uma boa ferramenta para este fim. Buscando solucionar o problema, planeja-se adotar um mecanismo de sincronização entre os agentes que considere uma linha de tempo pré-definida pelo usuário.

O sistema não pode garantir que o desenho de acorde escolhido é o melhor para uma determinada situação, mas simplesmente que ele é adequado. Para isso, o sistema não poderia usar o processamento em fases, descrito no Capítulo 4. Nem sempre o desenho do acorde escolhido traz um bom resultado sonoro, talvez por não ter sido considerada a condução de vozes (FILHO, 2005). Nota-se também uma certa tendência do sistema a levar a música para regiões mais agudas do violão, devido aos pesos definidos no algoritmo que calcula a similaridade entre os desenhos de acorde. Soluções para contornar esse problema ainda estão em estudo.

No aspecto rítmico a interface gráfica implementada mostra-se eficiente para capturar arpejos simples (dedilhados), entretanto pode ser muito trabalhoso definir certos padrões rítmicos com figuras de tempo muito pequenas e muitos tempos (batidas).

Podemos dizer que são necessários, ainda, experimentos para determinar todas as potencialidades e restrições do sistema, entretanto pode-se afirmar que o sistema atinge seu propósito inicial.

## 6.2 Publicações

Nesta seção são apresentadas todas as publicações do autor durante o curso de mestrado. Observa-se que algumas não possuem uma relação direta como o foco deste trabalho, entretanto todas as publicações estão inseridas nos contextos abrangidos pelas pesquisas, ou seja, as áreas da Computação Musicais, IA e Informática na Educação.

### 6.2.1 Mini-Curso

MILETTO, Evandro, Manara, COSTALONGA, Leandro Lesqueves, FLORES, Luciano Vargas, FRITSCH, Eloi Fernando; PIMENTA, Marcelo Soares, VICARI, Rosa Maria; **Introdução a Computação Musical**. In IV Congresso Brasileiro de Ciência da Computação. UNIVALI, Itajaí-SC, 2004.

### 6.2.2 Trabalhos completos em anais de eventos

NAKAYAMA, Lauro; VICARI, Rosa Maria; WULFHORST, Rodolfo; COSTALONGA, Leandro Lesqueves; MILETTO, Evandro Manara. **The Musical Interactions Within Community Agents**. In: 5TH WORKSHOP ON AGENT-BASED SIMULATON, 2004, Lisboa. Proceedings 5th. Workshop on Agent-Based Simulation. Erlangen: SCS Publishing House, 2004. p. 189-194.

COSTALONGA, L. L. ; VICARI, Rosa Maria . **Multiagent System for Guitar Rhythm Simulation.** In: International Conference on Computing, Communications and Control Technologies, 2004, Austin - Texas - EUA. Orlando: IIIS, 2004. v. 1. p. 6-11.

MILETTO, Evandro Manara; FRITSCH, Eloi Fernando; FLORES, Luciano Vargas; LOPES, Natália; COSTALONGA, Leandro Lesqueves; PIMENTA, Marcelo S. **Rumo ao Portal da Música Computacional e Eletronica**. In: V FÓRUM DO CENTRO DE LINGUAGEM MUSICAL, 2002, São Paulo. V Fórum do Centro de Linguagem Musical. São Paulo: ECA-USP, 2002. v. 1, p. 104-107.

### 6.2.3 Resumos simples em anais de eventos

FRITSCH, Eloi Fernando; MILETTO, Evandro Manara; COSTALONGA, Leandro Lesqueves; **Uma Proposta de Método para o Ensino de Técnicas de Composição de Música Eletrônica por Computador**; In Anais do Simpósio Brasileiro de Informática na Educação – Simpósio Brasileiro de Informática na Educação, Manaus. 2004.

COSTALONGA, Leandro Lesqueves; NOBILE, Vinicius., MILETTO, Evandro Manara; VICARI, Rosa Maria **Simulação Rítmica de Violão Baseada em Perfil do Usuário**. In Anais do Ambientes de Aprendizagem Baseados em Agentes. Simpósio Brasileiro de Informática na Educação, Manaus. 2004

MILETTO, Evandro Manara; PIMENTA, Marcelo Soares; COSTALONGA, Leandro Lesqueves.**: Using the Web-Based Environment for Cooperative Music Prototyping CODES in Learning Situations.** In: 7TH INTERNATIONAL CONFERENCE ON INTELLIGENT TUTORING SYSTEMS, 2004, Maceió. Proceedings of the International Conference on Intelligent Tutoring Systems. Springer-Verlag, 2004. p. 835-837.

COSTALONGA, Leandro Lesqueves; MILETTO, Evandro Manara; VICARI, Rosa Maria. **Cálculo de Acordes com Cifras Personalizáveis**. In: IX SIMPÓSIO BRASILEIRO EM COMPUTAÇÃO MUSICAL (IX SBCM), 1993, Campinas. Anais do XXIII Congresso da Sociedade Brasileira de Computação. 2003.

COSTALONGA, Leandro Lesqueves; MILETTO, Evandro Manara; FLORES, Luciano Vargas; ALVARES, Luis Otávio C. **Um Sistema Multiagente para Simulação de Performance Rítmica no Violão**. In: IX SIMPÓSIO BRASILEIRO DE

COMPUTAÇÃO MUSICAL (IX SBCM), 1993, Campinas. Anais do XXIII Congresso da Sociedade Brasileira de Computação. 2003.

GRANDI, Roges ; COSTALONGA, L. L. ; MENEZES, Paulo et al. Utilização do Ambiente Astrha para Implementar um Dicionário de Acordes Baseado em Autômatos Finitos. In: SBCM, 2003, Campinas. Anais do XXIII Congresso da Sociedade Brasileira de Computação, 2003.

### 6.2.4 Artigos em revistas e periódicos

MILETTO, Evandro Manara; COSTALONGA, Leandro Lesqueves; FLORES, Luciano Vargas; FRITSCH, Eloi Fernando; PIMENTA, Marcelo Soares; VICARI, Rosa Maria. **Educação Musical Auxiliada por Computador: Algumas Considerações e Experiências**. RENOTE - Revista Novas Tecnologias na Educação, Porto Alegre, RS, v. 2, 08 mar. 2004.

### 6.2.5 Artigos submetidos e aguardando resposta

COSTALONGA, L.; VICARI, R. A Library for Recognize and calculate chords for cords instruments. ICMC 2005;

COSTALONGA, L., VICARI, R. **A Guitar Player Mutiagent System**; PROMAS – Workshop de Programação de Sistemas Multiagentes do AAMAS 2005.

COSTALONGA, L.; GOMES, E.; MILETTO, E.; VICARI, R. **Agent-Based Guitar Performance Simulation**. AAAI 2005

## 6.3 Trabalhos Futuros

Além de solucionar os problemas já relatados, o sistema tem como principal meta o aumento da cognição dos agentes, em especial do agente MD.

O trabalho pode ter continuidade no sentido de permitir que os agentes consigam não somente executar uma peça musical no violão, mas também que consigam demonstrar expressividade ao fazer. Para isso, pretende-se dotar o a Agente MD com a noção de emoção;

Pesquisas na área de neurociência demonstraram que as emoções reforçam e agilizam o mecanismo de tomada de decisão quando a situação demanda ações urgentes. Em situações onde há bastante tempo para decidir, geralmente, as decisões são baseadas em processos racionais envolvendo raciocínio e dedução(VENTURA et al. 1999). Segundo Damásio, quanto maior a urgência e seriedade da situação menos racional e mais emocional serão nossas decisões. Este é o jeito dos humanos lidarem com a complexidade(DAMÁSIO, 1994). A música é uma atividade dependente do tempo, logo susceptível as ações emocionais.

Emoções podem ser analisadas sobre dois diferentes pontos de vista: o externo, comportamental, onde a comunicação entre os indivíduos é considerada para ajudar nas atitudes baseadas na emoção, e a interna, funcional, na qual os mecanismos da emoção são cruciais no entendimento do processo de tomada de decisão(VENTURA et. al. 1999). O entendimento das emoções é difícil porque elas não são derivadas de pensamento verbal, logo, difíceis de descrever. Felizmente, humanos não se comunicam

somente de forma oral ou escrita, eles podem se comunicar não verbalmente usando gestos ou expressões faciais, por exemplo(KAISER, 1994).

Como uma forma de arte, a música é distinta pela presença de muitas relações que podem ser tratadas matematicamente, incluindo ritmo e harmonia. Existem também vários elementos não matemáticos tais como tensão, expectativa e emoção(DANNENBERG, 1993).Uma das principais características que difere os humanos de outros animais é o fato de que somos intrinsecamente musicais. Música é geralmente associada com a expressão das emoções, entretanto o intelecto desempenha importante papel nas atividades musicais(MIRANDA, 1999). A música também pode ser vista como uma forma de comunicação e em todas estas visões mencionadas pode servir como uma ponte mediadora na captura e expressão da emoção pelo computador(NEMIROVSKY et al. 1999).

A emoção é algo com que os músicos estão acostumados a lidar e isto fica claro nos conceitos harmônicos de consonância e dissonância que nos remetem a sensações agradáveis ou irritantes, respectivamente(ROEDERER, 2002). Desta forma, conclui-se que: se for possível extrair de uma música a emoção empregada pelo compositor também pode ser possível, conhecendo certas regras de composição, gerar uma música ou acompanhamento baseado em uma emoção. Esse é caminho evolutivo deste trabalho.

# REFERÊNCIAS

BATERA. Disponível em: <http://www.batera.com.br/>. Acesso em: mar. 2005.

BILOTTA, E.; PANTANO, P.; TALARICO, V. Synthetic Harmonies: An Approach to Musical Semiosis by Means of Cellular Automata. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL LIFE, 7., 2000, Portland. **Artificial Life VII**: proceedings. Cambridge: MIT Press, 2000.

CHEDIAK, A. **Dicionário de Acordes**. São Paulo: Irmãos Vitale Ed., 1984.

CONGER, J. **C Programming for MIDI**. Redwood City, California, USA: M&T Publishing, 1988.

CONTE, R.; CASTELFRANCHI, C. Minds is not Enough: Precognitive Based of Social Interaction, In: SYMPOSIUM ON SIMULATING SOCIETIES, 1992**. Proceedings…**[S.l.: s.n],1992. p. 93-110.

DAHIA, M. et al. Generating Rhythmic Accompaniment for Guitar: the Cyber-João Case Study. In: BRAZILIAN SYMPOSIUM ON COMPUTER MUSIC*,* SBCM, 9., 2003. **Proceedings…**[S.l.: s.n], 2003.

DAMÁSIO, A. R. **Descartes' Error, Emotion, Reason, and the Human Brain**. New York, NY: Goset/Putman ,1994 .

DANNENBERG R. Music Representation Issues, Techniques, and Systems. **Computer Music Journal**, Cambridge, v. 17, n.3, p. 20-30, 1993.

DANNENBERG, R. B.; THOM B.; WATSON D. A Machine Learning Approach to Musical Style Recognition. In: INTERNATIONAL COMPUTER MUSIC CONFERENCE, 1997. **Proceedings…**[S.l.: s.n], *1997.* p. 344-347.

DICIONÁRIO de Música. Rio de Janeiro: Zahar Ed., 1982.

DOEL, K. **Real-time audio synthesis using JASS** Disponível em: <http://www.cs.ubc.ca/spider/kvdoel/jass/> Acesso em: mar. 2005.

FERBER, J. **Multi-Agent Systems :** An Introduction to Distributed Artificial Intelligence. Harlow: Addison-Wesley, 1999.

FILHO, M. **Contraponto a duas vozes.** Disponível em: <http://www.guitarx.com.br/index.asp?url=library_html/destaque/1%C2%AAsemana_nov/mauricio_contraponto.htm>. Acesso em: mar. 2005.

FOX M, et al. An Organizational Ontology for Enterprise Modeling. Simulation Organizations Computational – Models of Institutions and Groups. *In:* PRIETULA, M.; CARLEY K.; GRASSER, L. (Ed.)**. Simulating Organizations:** Computational models of institutions and groups. Cambridge: AAAI Press/ MIT Press, 1998. p. 131-152.

GABRIELSON, A. Rhythm in Music**.** In: EVANS, J. R; CLYNESm M. (Ed.). **Rhythm in Psychological, Linguistic and Musical Processes.** Springfield: Charles C. Thomas, 1986. p.131-167.

GERRIT, G. **Java and Midi Programs:** Wire Provider. Disponível em: <http://www.geocities.com/ggehnen/>. Acesso em: set. 2003

HUHNS M.; SINGH P. **Reading in Agents.** San Francisco, California: [s.n], 1998. p. 91-97

JAVASOUND. Disponível em: <http://java.sun.com/products/java-media/sound/>. Acesso em: mar. 2004.

JFUGUE. Disponível em: <http://innix.com/jfugue/>. Acesso em: ago. 2003

JMSL - Java Music Specification Language. Disponível em: <http://www.algomusic.com/jmsl/index.html>. Acesso em: set. 2003.

JSCORE. Disponível em: <http://homepages.nyu.edu/~ray208/Jscoring/html/JScore.html>. Acesso em: set. 2003 .

JSYNTHLIB - Universal Synthesizer Patch Editor. Disponível em: <http://www.overwhelmed.org/jsynthlib/>. Acesso em: set. 2003

KAISER, S.; WEHRLE, T. Emotion research and AI: some theoretical and technical issues. **Revue d'Intelligence Artificielle**, Genève, 1994.

MARSANYI, R. **JavaMIDI**. Disponível em: <http://www.softsynth.com/javamidi/>. Acesso em: set. 2003

MCNABB, M. **Java MIDI Kit:** Fantasia. Disponível em: <http://www.mcnabb.com/software/fantasia/index.html>. Acesso em: set. 2003.

MIDI SHARE. **Java Programming:** MidiShare and Java. Disponível em: <http://www.grame.fr/MidiShare/Develop/Java.html> . Acesso em: set. 2003

MIRANDA, E. Music, Machines, Intelligence and the Brain. **San Journal of Electroacoustic Music,** [S.l.], n.12, 1999.

MIRANDA, E. R. Emergent Sound Repertories in Virtual Societies. **Computer Music Journal**, Cambridge, v. 26, n.2, p. 77-90, Summer 2002.

NEMIROVSKY, P.; DAVENPORT, G. GuideShoes: Navigation based on musical patterns. In: CHI,1999. **Extended Abstracts** . New York, NY: ACM, 1999. p. 266-267.

NOSUCH MIDI. Disponível em: < http://www.nosuch.com/nosuchmidi/>. Acesso em: set. 2003.

PG Music Inc. **Band-in-a-box Pro 8.0**. Canadá, 1998.

PROJECT XEMO. Disponível em: <www.xemo.org>. Acesso em: set. 2003.

RAMALHO, G.; ROLLAND, P.-Y.; GANASCIA, J.-G. An Artificially Intelligent Jazz Performer. **Journal of New Music Research,** Amsterdam , v.28, n.2, p. 105-129, 1999.

REZENDE, S.O. **Sistemas Inteligentes:** Fundamentos e Aplicações. Barueri, SP, Manole Ed.,2003.

ROADS, C. **The Computer Music Tutorial**. Cambridge : Mit Press, c1996.

ROEDERER, J.**Introdução à Física e Psicofísica da Música**. São Paulo: Ed. EDUSP, 2002.

RUSSELL, S. J. **Inteligência Artificial.** Rio de Janeiro: Campus, 2004.

RUSSELL, S.; NORVIG, P. **Artifical Intelligence:** A Modern Approach. New Jersey, Prentice Hall, 1995.

SANTANA, H. et al. VexPat - An Analysis Tool for the Discovery of Musical Patterns. In: BRAZILIAN SYMPOSIUM ON COMPUTER MUSIC, SBCM, 9., 2003; **Proceedings…**[S.l.: s.n], 2003.

SHER, C. **The New Real .** Berkeley: Sher Music, 1991.

SIMON, H. **Model of Bounded Rationality:** BehavioralEconomics and Business Organization. Cambridge: MIT Press,1982.

SORENSEN, A.; BROWN,A. **jMusic:** Music Composition in Java. Disponível em: <http://jmusic.ci.qut.edu.au/>. Acesso em: set. 2003.

TOM DA MATA. Disponível em: <http://www.tomdamata.org.br/cancioneiro/ritmo.asp>. Acesso em: mar. 2005.

TRITONUS: Open Source Java Sound. Disponível em: < http://tritonus.org/>. Acesso em: set. 2003.

VENTURA, R.; CUSTÓDIO, L.; PINTO C. **Artificial Emotions** - Good Bye Mr. Spock! In: **Cognitive Science**, Tokyo, Japan, p. 938-941, 1999.

WESSEL, D.; WRIGHT, M. Problems and Prospects for Intimate Musical Control of Computers. **Computer Music Journal**, Cambridge, v.26, n.3, p. 11-22, Fall 2002.

WEST, R.; HOWELL, P.; CROSS, I. Musical Structure and Knowledge Representation. In: HOWELL, P.; WEST, R.; CROSS, I.(Ed.). **Representing Musical Structure.** London: Academic Press, 1991. p.1 -30.

WULFHORST, R. et al. An Open Architecture for a Musical Multi-Agent System. In: BRAZILIAN SYMPOSIUM ON COMPUTER MUSIC, SBCM, 8., 2001. **Proceedings…**[S.l.: s.n], 2001.

WULFHORST, R. **Uma Abordagem Multiagente para Sistemas Musicais Interativos**.2002. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

# ANEXO A ANÁLISE DE SOFTWARES MUSICAIS FOCADOS EM VIOLÃO/GUITARRA

## Introdução

O objetivo desta análise foi embasar as funcionalidades inovadoras da proposta e compará-la com trabalho similares. A seleção dos software analisados baseou-se na data de distribuição dos mesmos, sendo a maior parte deles do ano de 2002 em diante. Alguns casos específicos não obedecem a este critério. Isto ocorreu quando à funcionalidade desses software são particularmente similares ao software desenvolvido na dissertação.

Devido ao grande número de software encontrados, os mesmos foram classificados em: Dicionários de Acorde, Editores de Tablatura, Sistemas de Treinamento, Software para Composição. Não se pretende com essas classificações cobrir todos os tipos de software que tem o foco em guitarra, simplesmente organizar os software que possuem funcionalidades afins com o software fruto da dissertação.

## Dicionário de Acordes

Nesta categoria são analisados software que tem como principal característica mostrar como um acorde é executado no violão/guitarra, seja através de uma representação gráfica, sonora ou textual. É comum encontrar Dicionários de Acordes como parte de software classificados em outras categorias que possuem objetivos mais abrangentes que os Dicionários de Acordes.

**ChordSMART 2.0:** Permite a visualização do acorde sobre a imagem do violão, com fácil navegação entre os vários desenhos dos acordes. Trabalha com os principais intervalos, porém estes são fixos, não permitindo a alteração dos símbolos e a composição de intervalos. Um ponto positivo é a possibilidade da alteração da afinação do violão, entretanto esta alteração é limitada a uma lista de opções.

**D´Accord Guitar Chord Dictionary 2.0:** Um bom software brasileiro desenvolvido pela UFPE. O D´Accord demonstra visualmente e sonoramente o acorde em seus diversos desenhos. Os acorde podem ser escolhidos de uma base de dados ou desenhados diretamente no braço do violão, que pode ser personalizado (visão de frente, canhota, destra etc.) Integrado ao software existe um transpositor de tons que auxilia nos cálculos da cifras.

**FretboardDots 2.2:** Um dicionário de acordes com uma abordagem diferente das demais analisadas. Os acordes não são identificados pelas cifras e sim pelos intervalos. Apesar de confuso para usuário menos experientes em música, o sistema conseguiu resolver com simplicidade a questões das divergências de símbolos de cifragem. A afinação do instrumento pode ser alterada. Um ponto negativo é que trata somente os intervalos de uma oitava.

**WinChord 4.2:** Um dicionário de acordes bem completo. Como quase todo dicionário que calcula os acordes ao invés de uma simples indexação dos desenhos, ele permite alterar os parâmetros de calculo do algoritmo (afinação do instrumento, quantidade de dedos, abertura dos dedos etc) para melhor ajustar-se as necessidades e restrições do usuário. O acorde é mostrado no braço da guitarra, em partitura e em um teclado musical. A lista de intervalos é bem ampla porém os símbolos são estáticos e estão na notação de cifragem (EUA). O caminho inverso, ou seja, a identificação do acorde através do desenho do acorde também é possível. Como ponto negativo temos a interface muito confusa.

**ChordWizard Gold 2.0:** A completude e generalidade com que os temas musicais foram abordados por este software tornam-o de difícil classificação. Não é o objetivo deste trabalho descrever com detalhes o software analisado, portanto apenas lista-se apenas os pontos mais significativos deste software: personalização da interface, suporte a vários instrumentos de corda, representação gráfica e sonora dos acordes como descrição das notas e intervalos (inclusiva com partitura), dedução explicativa do acorde a partir do desenho do acorde, indicação de acordes e escalas equivalentes, metrônomo, afinador, completa biblioteca de escalas, tutorial de teoria musical e bom suporte a impressão.

**AxMaster 1.1:** Um dicionário de acordes com glossário e teoria musical. Apesar de não ter suporte a som, possui uma interface simples e clara.

## Editores de Tablatura

Esta categoria é provavelmente a mais abrangente dentre todas mencionadas. O principal objetivo dos software desta categoria é documentação da música em tablaturas/partituras musicais e a manipulação dos dados armazenados nas suas mais diversas variações, como por exemplo, a execução sonora da música (funcionalidade comumente encontrada). Em editores ainda é comum encontrar dicionário de acordes, afinadores, modos das escalas, etc. Estas ferramentas agregadas as funcionalidades básicas do editores de tablaturas, transforma o simples editor em verdadeiros ambientes de auxílio a composição.

**Chord Writer 2.0.2:** É um editor de tablatura muito simples, pouco mais que um dicionário de acordes. Simplesmente vincula informações harmônicas (desenho do acorde) a letra da música para impressão ou troca de dados. Não faz nenhum processamento musical com esses dados.

**TEFview 2.60:** Apesar de estar classificado com um editor de tablatura, esta aplicação é simplesmente um bom visualizador. Dentre suas principais funcionalidades: visualização em partitura e tablatura simultaneamente. As diagramações dos acordes também podem ser inseridas na própria partitura ou visualizadas em uma janela flutuante enquanto a música é reproduzida. Tem como ponto negativo o formato proprietário do arquivo.

**TablEdit 2.62:** Engloba todas as funcionalidades do TEFview 2.60 além das funcionalidades de edição. É possível entrar as notas diretamente na partitura/tablatura usando teclado virtual, guitarra virtual ou as figuras de notas disponíveis em um menu. O que diferencia este editor dos demais são os recursos extras para inserção de símbolos específicos para tablaturas de guitarra como efeitos (*bend, hammer-on, pull-off* etc) e informações de mão direta (dedos e direção do arpeggio). Infelizmente as informações de mão direita não influenciam na sonoridade no ato da reprodução da música e também não há suporte para entradas de dados via instrumento MIDI. Gravam no formato proprietário TEF.

**Tab Player 4.053:** Permite a reprodução e visualização dos acordes em tablaturas, ou seja, seis linhas horizontais com números indicando os as casas que devem ser pressionadas. Como tablaturas não são precisas no tempo, pode ocorrer imprecisão na reprodução da mesma. Um ponto positivo é um mecanismo de busca de tablaturas na web integrado na ferramenta.

**Power Tab 1.7:** Um excelente representante da categoria dos editores de tablatura com algo mais: controle dos padrões rítmicos. Além de todas as funcionalidades de um bom editor de tablaturas como dicionário de acordes, suporte a exportação e impressão, afinador etc esta ferramenta traz símbolos específicos de tablaturas para violão que afetam a sonoridade produzida, são efeitos como: *bend, pull-off, slides, vibrato* etc. O grande diferencial deste editor esta no controle dos padrões rítmicos, apesar de um pouco difícil de entender, possui paremetrizações como a direção do arpeggio, o acento, duração, forma de abafamento etc.

**Guitar Pro 4:** Um dos primeiros software musicais focado nas necessidades dos violonistas/guitarristas. Por sua maturidade, sem dúvida, um dos melhores e mais usados. Possui uma interface bem organizada que fornece ao usuário várias possibilidades de personalização. Dicionário de acordes, afinador, escalas e outras ferramentas compõem este software. A simbologia da tablatura é bem completa e isto permite uma boa precisão musical e fácil acompanhamento na reprodução da música. Apesar de possibilitar na notação da tablatura informações de mão direita, tais como o direção do arpejo e o dedo que irá tocar cada corda, fica evidente a omissão de como os padrões rítmicos devem ser executados na guitarra, isto porque os acordes (apesar de notados em diagramas sob a tablatura) são considerados notas individuais espalhas no tempo ou sobrepostas.

**Desktop Guitarist 2:** Para a categoria que está inserido, este software é demasiadamente simples. Na verdade, este software foi desenvolvido por um guitarrista que não teve preocupações técnicas e de engenharia de software na construção da aplicação. Possui afinador, escalas e dicionário de acordes, mas tudo muito simples. Não possui um acompanhamento da tablatura com o som gerado, deixando simplesmente que o sistema indique como os acordes são feitos em uma ilustração do braço da guitarra.

**Decifra 2:** Na versão de demonstração analisada o software não passa de um visualizador de cifras (ou diagramas) associadas a letra da música. Também possui um fraco dicionário de acordes e lições de guitarra.

**WebLyrics 1.0.3.1:** É um ambiente para controle de letras cifradas. O grande diferencial deste software brasileiro é a forma como a tablatura é reproduzida, isto é, o usuário determina o arpejo dos acordes usando o teclado numérico, possibilitando a impressão de uma certa expressividade e controle do andamento.

## Sistemas de Treinamento

São sistemas que tentam transmitir conhecimento aos usuários e validar o aprendizado de alguma forma. Em geral trazem exercícios e grande repositório de informações para consultas e referencia.

**Fretboard Warrior 1.0:** Aplicação simples em formato de jogo que mostra a nota em uma representação gráfica do braço da guitarra e simultaneamente toca o som correspondente da nota. O usuário deve indicar em um menu qual é a nota que a aplicação lhe mostrou.

**Ultimate Guitar Chord Trainer:** Aplicação que demonstra a formação de acordes através da inclusão das notas (seqüencialmente) na figura de um braço de guitarra. O nome das notas, o som, os intervalos que compõe o acorde e o nome do acorde são mostrados. O usuário pode parametrizar como os acordes são formados. A interface é um pouco confusa e não ha como navegar entre os desenhos do acorde, isto porque os acordes são mostrados em seqüência com tempo de troca entre eles pré-determinado. Isto seria um erro grave se considerássemos esta aplicação como um dicionário de acordes.

**NSA Song Player 1.22:** Apesar de o próprio fabricante o considerar um sistema de treinamento, isto é duvidoso. A aplicação possui um dicionário de acordes que permite alteração da afinação e um guia de escalas. Escolhendo os acordes pode-se criar uma música e reproduzi-la ou ainda buscar os dados musicais de uma tablatura. A interface é extravagante. Desenvolvido em Flash e, talvez por isso, deixa a desejar em um projeto de interface mais adequado.

**aGuitar Pro 1.0:** Uma aplicação que envolve teoria musical e exercícios para treinar o ouvido do guitarrista. Possui testes para reconhecimento de acordes, notas, notas no braço do instrumento além de um executor de tablaturas (simplesmente toca as tablaturas).

**Guitar Trainer 2.4:** Relaciona as notas na partitura com sua posição na guitarra. Possui uma série de testes associados a um aprendiz que pode ser criado.

**Guitar Power 1.1.6:** O fabricante o considera um sistema de aprendizado, mas não o vejo desta forma. É um dicionário de acordes que mostra arpeggios (bem simples) e também trabalha com escalas. Tem uma boa interface e classifica bem os acordes, alias, este é seu diferencial.

**EarMaster Pro 4.0:** Um excelente sistema de treinamento para qualquer instrumento musical. Possui testes para treinamento rítmicos, melódico, harmônico de intervalos e de reconhecimento de acordes em vários níveis onde a dificuldade vai sendo aumentada gradualmente a pedido do usuário. Interface bem projeta.

## Software para Composição

Geram material sonoro que podem ser usados em composições. Podem aparecer em forma de *plug-ins* de software de gravação ou como aplicações independentes que sintetizam os sons e descarregam em um arquivo de áudio.

**Voicings 4.06:** Um software com uma idéia simples porém inovadora. Neste software é possível trabalhar separadamente as vozes dos acordes e seqüenciá-las definindo efeitos de volume, *pull off, hammer* e outros. É como se estivéssemos

trabalhando o modo de arpejar, ou seja, informações de mão direita. Tem ainda o recurso de dicionário de acordes e uma boa estratégia de *loopings*. Isto é muito parecido com o objetivo desta dissertação, porém com um enfoque diferente.

**Rhythm n Chords Lite 2.04 (Plug-in para o Cakewalk):** O mais profissional dos software analisados. É distribuído em forma de *plug-in* para os software de gravação multitrilha para os produtos da Cakewalk. Objetiva criar a parte de guitarra da composição através da criação de uma progressão de acordes associados a padrões rítmicos armazenados em bibliotecas de ritmos. Permite parametrizações do controle dos acordes e dos arpeggios.

**Melody Assistant 6.2.0:** O Melody Assistant compõe um ambiente integrado de 3 ferramentas: Harmony Assistent, Virtual Singer e o próprio Assistent. Eles podem funcionar separadamente, mas possuem a mesma interface. Este ambiente é semi-profissional, possui uma boa qualidade de som e suporte a MIDI. Sua interface é bonita, personalizável, mas confusa. O Melody Assintent é a parte do ambiente que trabalha com a melodia e genérico a qualquer instrumento. Só se tem indícios que é um software com foco (ou um dos focos) na guitarra quando, ao explorarmos, encontramos um dicionário de acordes e a opção de visualizar a melodias em tabalturas.

**Harmony Assistant 8.2.0:** Estende o Melody Assistente com funcionalidade de acompanhamento e ritmo. Tem muitas opções de configuração que sugere um poder maior do que realmente o software tem. Falha na documentação que o acompanha e isso o torna ainda mais difícil de ser operado e compreendido. Sugeri em sua documentação que podem ser criados padrões rítmicos para qualquer instrumento, inclusive guitarra/violão, mas não foi possível testar devido falta de documentação apropriada. Entretanto é claro de ver que existe um bom controle de padrões rítmicos voltados a percussão. É uma ferramenta com funcionalidades bem poderosas.

# ANEXO B TECNOLOGIAS JAVA APLICADAS A COMPUTAÇÃO MUSICAL

## Introdução

São várias as linguagens de programação que vêm sendo usadas para escrever software musicais, entretanto uma vem se destacando como a preferida pelos principais grupos de pesquisa: Java. É fato que Java é uma linguagem que ganhou grande notoriedade e vem sendo usada em diferentes áreas de conhecimentos tanto na indústria como na academia, porém quais aspectos qualificam Java como uma boa linguagem para aplicações musicais?

Este trabalho apresenta as principais tecnologias Java aplicadas à computação musical através de uma análise funcional e exemplos práticos. A seguir serão descritos os pacotes analisados.

## Java Sound (JSDK 1.4.2)

Java Sound é uma API que provê suporte para operações de áudio e MIDI com alta qualidade, tais como: captura, mixagem, gravação, sequenciamento e síntese MIDI. O Java Sound vem junto com o J2SE desde a versão 1.3 e por isso é considerada por alguns como a fonte do sucesso do Java para a programação musical

Em constante evolução e melhoria, são características nativas do Java Sound:

- Suporte aos formatos de arquivos de áudio: AIFF, AU e WAV;
- Suporte aos formatos de arquivos de música: MIDI Type 0, MIDI Type 1, e Rich Music Format (RMF);
- Formatos de som em 8/16 bits, mono e stereo, com sample rate de 8 a 48 Hz;
- Dados de áudio codificados em linear, a-law, and mu-law para qualquer um dos formatos de áudio suportados;
- Síntese e sequenciamento MIDI por software, bem como acesso a qualquer dispositivo MIDI em hardware;
- Mixer com capacidade para mixar e renderizar mais de 64 canais de áudio digital e sons MIDI sintetizados.

Como exemplos de aplicações potenciais para o Java Sound, cita-se:

- Frameworks para comunicação, como conferencia e telefonia;
- Sistemas de entrega de conteúdo como media players;
- Programas interativos como jogos e websites dinâmicos;
- Criação e edição de conteúdo sonoro e musical;
- Componentes e ferramentas para manipulação Sonora;

A API Java Sound é o suporte a áudio de mais baixo nível na plataforma Java, permitindo aos programas um bom controle nas operações de som. A API não inclui editores de som sofisticados ou ferramentas gráficas, mas é extensível o suficiente para que isso seja construído a parir dele, como uma API de baixo nível deve ser.

Existem API´s de mais alto nível para desenvolvimento mais rápido e fácil de aplicações multimídia. A própria Sun, fabricante do Java Sound, distribui o Java Media Framework (JMF). O JMF especifica uma arquitetura unificada, protocolo para trocas de mensagem e interface de programação para captura, execução e sincronização de mídias baseadas no tempo, como som e vídeo.

O Java Sound suporta tanto áudio digital como MIDI, através de seus dois pacotes distintos:

- javax.sound.sampled: Captura, mixa e toca audio digital (samples);
- javax.sound.midi:Síntese, sequenciamento e tratamento de eventos MIDI;

Dois outros pacotes permitem que terceiros criem e distribuam componentes de software customizados para estender as capacidades do Java Sound.

- javax.sound.sampled.spi
- javax.sound.midi.spi

**Configurações de áudio**

O Java Sound API utiliza os recursos de hardware disponíveis na máquina para desempenhar sua principal função: transportar áudio (mover dados formatados para dentro fora do sistema). Para executar a contento esta função, o Java Sound implementa métodos de gravação e conversão entre alguns formatos de áudio deixando a cargo de terceiros a implementação de *plug-ins* que venham suportar novos formatos de áudio.

Muitas API´s para tratamento de som trabalham com o conceito de dispositivo de áudio (*devices*). Normalmente, um *device* é uma interface para um dispositivo físico de entrada/saída, No contexto do Java Sound, *devices* são representados por objetos do tipo Mixer, que tem a função de juntar e direcionar diversas entradas de áudio a uma ou mais saídas.

O objeto responsável por mover o áudio para dentro e para fora do sistema (normalmente para dentro e fora do *Mixer*) é a *Line*. Outros objetos também podem ser vistos como uma *Line* tais como portas de entrada e saída (microfone, CD-ROM, *headfone* etc.) e até mesmo o *Mixer*, pois eles também podem ser caminhos para o áudio. Pode-se fazer uma analogia da *Line* com uma trilha (*track)* em um gravador *multitrack* ligado a um *mixer* externo, com a diferença que, no Java Sound, uma *Line*

pode ser multi-canal de acordo com o formato de áudio em transporte. A Figura 1 mostra a relação de herança entre  algumas classes do Java Sound.

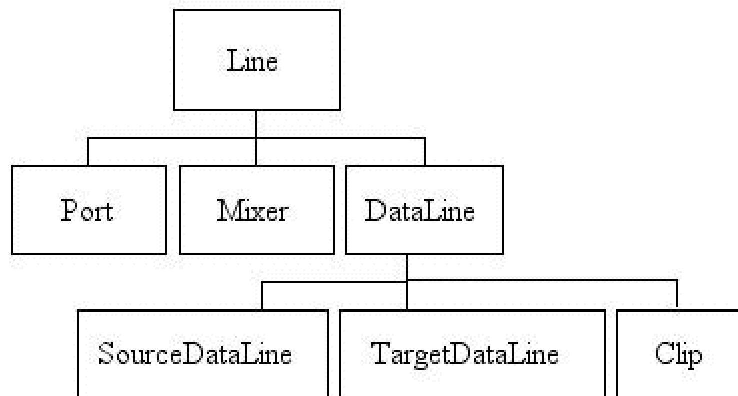



Fig. 1 Modelo hierárquico do objetos do Java Sound.

A interface Mixer define métodos para se obter as *Lines* que ele gerencia, que podem ser do tipo SourceDateLine (input) ou TargetDataLine(output). A *inteface Line,* em si, não define métodos para controle de gravação e execução, sendo estes só definidos nas *DataLine´*s.

*Clip* é um *DataLine* que armazena todo o som na memória antes de sua execução. Graças a essa característica, é possível navegar pela mídia e programar repetições (*loop*).

Todo recurso de áudio é obtido através da classe que gerencia os recursos de hardware da máquina: *AudioSystem*.

**Uso do Java Sound na Web**

*Applets* podem usar o Java Sound, entretanto deve-se ficar atento as restrições de segurança definidas na classe *AudioPermission*.O padrão é que uma *Applet* rodando sobre restrições de segurança pode executar, mas não gravar sons. Isso pode ser modificado caso o usuário permita.

**MIDI**

O pacote javax.sound.midi tem classes responsáveis pelo transporte, sequenciamento e síntese de eventos MIDI .

O diagrama mostrado na Figura 2 ilustra as relações funcionais entre os componentes principais possíveis em uma configuração MIDI baseada no Java Sound.

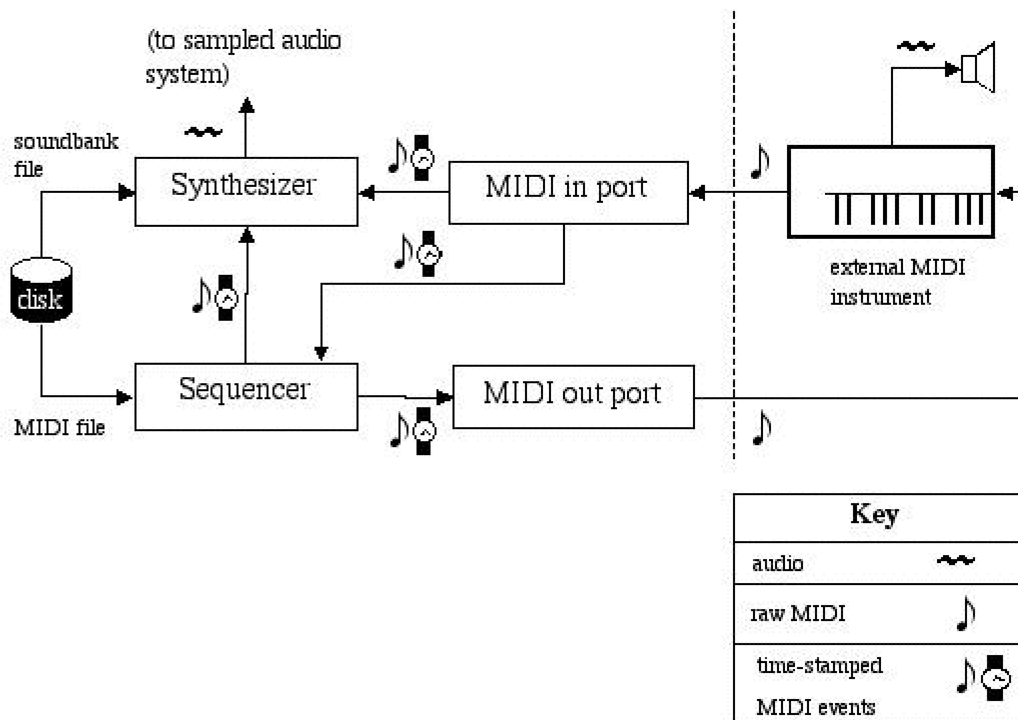



Fig. 2: Configuração de componentes no Java Sound.

No exemplo acima, uma performance musical é preparada a partir de uma partitura armazenada em MIDI e persistida em disco.Este arquivo MIDI é lido e executado pelo seqüenciador (software), que envia as mensagens MIDI para outros dispositivos, com o sintetizador (interno e/ou externo). O sintetizador pode usar *soundbanks* para uma melhor qualidade nos timbres dos instrumentos sintetizados.

**Interfaces para Expansão de Funcionalidades (SPI - Service Provider Interfaces)**

Os pacotes `javax.sound.sampled.spi` and `javax.sound.midi.spi` contém interfaces(*mixer* de áudio, sintetizador, conversor etc.) que permitem a desenvolvedores independentes criarem novos recursos de MIDI e áudio que podem ser distribuídos separadamente e conectados (*plug-ins*) as implementações feitas com Java Sound sem que estas necessitem sofrer modificações de código. Estas implementações podem ser puramente software ou interfaces para melhor uso do hardware.

**Considerações sobre o Java Sound**

O Java Sound não foi a primeira biblioteca a manipular sons e dados musicais (MIDI) na plataforma Java, mas sem dúvida é a mais importante delas. Apesar das restrições, o Java Sound permite que as aplicações escritas com suas classes funcionem da mesma forma nas diversas plataformas suportadas pelo Java.

Nota-se também que a Sun não tem interesse em desenvolver muito mais o Java Sound, que teve sua missão cumprida, ou seja, ele é e vai continuar sendo uma biblioteca de baixo nível para manipulação de som e dados musicais com uma importante alternativa de expansão, o SPI.

## JMSL – Java Music Specification Language

JMSL é um *framework* desenvolvido em Java que auxilia no desenvolvimento de software musicais, em especial para composição de música computacional, performances interativas e construção de instrumentos virtuais.

Entre as principais vantagens do JMSL, pode-se citar:

Total integração com a linguagem (Java) e recursos da mesma, incluindo conectividade com banco de dados, ferramentas de rede, gráficos 2D e 3D, Servlet etc.;

Comunicação direta com outras bibliotecas e dispositivos implementados em Java, como o Jsyn, JavaMIDI, MidiShare e Java Sound;

Incorporação do pacote JScore, uma biblioteca que permite edição da notação musical;

Permite ao compositor distribuir as aplicações localmente ou através de *Applets*;

Gratuito em uma versão mais restrita (Lite);

Baseada no HMSL – Hierarquical Music Specification Language, o JMSL mantém a idéia original de seu predecessor ao basear-se no conceito de hierarquias ao desenvolver aplicações musicais. Hierarquias nada mais são do que relacionamentos pai-filho, com representado na Figura 3.

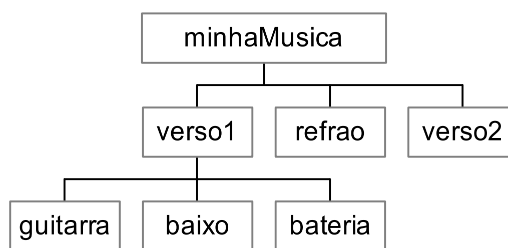



Fig. 3: Exemplo de organização hierárquica em uma música. .

As hierarquias do JMSL não necessariamente precisam ser fieis a realidade, podendo-se criar estruturas hierárquicas altamente complexas sem nenhuma equivalência aparente no mundo real. Hierarquias podem ser previamente definidas ou criadas durante a execução de uma peça musical, entretanto todos os objetos hierarquizáveis devem ser do tipo *Composable* (interface que define tal característica). Dois tipos de hierarquias podem ser criados no JMSL: seqüencial e paralela; Estas são, respectivamente, representadas pelas classes S*equencialCollection* e *ParallelCollection*.

Hierarquizar, no contexto do JMSL, não significa apenas posicionar os objetos em uma visão *top-down*, definindo assim a importância dos “objetos” (entidades) no contexto musical. No JMSL, posicionar um objeto na hierarquia significa definir o momento em que o mesmo será executado. Desta forma, o objeto mais alto na hierarquia solicita aos seus “filhos” (objetos ligados diretamente a ele) que sejam executados todos juntos (hierarquia paralela) ou seqüencialmente (hierarquia seqüencial).

O trecho de código abaixo (Exemplo 1), mostra como é possível montar uma hierarquia paralela. Não vamos entrar em detalhes dos objetos neste momento, entretanto, para uma boa compreensão do código, é importante citar que tanto o *ParallelCollection* como a *MusicShape* são especializações do *MusicJob*, que por sua vez, implementa a interface *Compasable*.

```
ParallelCollection col = new ParallelCollection();
col.setRepeats(3);

MusicJob mj = new MusicJob();
mj.setRepeatPause(2.0);
mj.setRepeats(4);

MusicShape sh = new MusicShape(3);
sh.add(1.0, 66, 120);
sh.add(1.0, 67, 120);
sh.add(1.0, 68, 120);
sh.add(1.0, 69, 120);
sh.setRepeats(2);

col.add(mj);
col.add(sh);
col.launch(JMSL.now());
```

Exemplo 1: Construção de uma hierarquia paralela.

No Exemplo 1, 3(três) objetos são criados: "col" (ParallelCollection), "mj" (MusicJob) e "sh" (MusicShape) e estão hierarquizados como mostra a Figura 4. Por estarem ligados a um objeto do tipo *ParallelCollection*, os objetos "mj" e "sh" são executados simultaneamente a partir do momento que o "col" é executado (linha 14). Cada um dos objetos terá sua execução repetida por diferentes vezes. A cada execução do "col" (são 3), o mj é executado 4 vezes e o "sh" duas vezes, sendo que o "mj" tem um intervalo entre as repetições de 2 segundos.

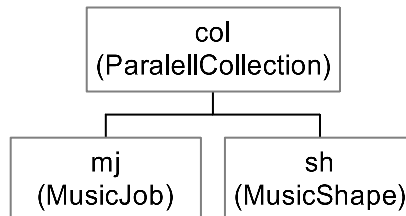



Fig. 4: Representação visual da hierarquia do Exemplo 1.

**Principais Classes e Intefaces**

As classes e interfaces do JMSL e API´s correlatas (JScore, Jsyn, JavaMIDI etc.) estão organizadas em pacotes, como é de costume. Os principais pacotes e suas respectivas características são citados na Tabela 1.

| Pacote | Principais Características |
|---|---|
| com.softsynth.jmsl | Pacote que contém as principais classe e interfaces do JMSL. Imprescindível para qualquer aplicação que venha usar o JMSL. |

| | |
|---|---|
| com.softsynth.jmsl.jsyn | Engloba as classes e interfaces necessárias para a integração com o Jsyn. |
| com.softsynth.jmsl.midi | Classes necessárias para manipulação de eventos MIDI, incluindo comunicação com dispositivos externos. |
| com.softsynth.jmsl.net | Permite manipulação(serialização) de objetos JSML na Web. |
| com.softsynth.jmsl.score | O JScore agregado ao JSML. Classes e interfaces que permitem edição de notação musical. |
| com.softsynth.jmsl.util | Conjunto de classes utilitárias de especial uso para composição algorítmica. |
| com.softsynth.jmsl.view | Conjunto de classes gráficas para exibição de informações musicais tratadas com o JSML. |

Tab. 1 Principais pacotes do JSML

Descreve-se a seguir as classes e interfaces mais relevantes do pacote com.softsynth.jmsl, que por sua vez é o principal pacote do JSML.

**Composable**

Qualquer objeto Java que implemente a interface *Compasable* pode ser posicionado em uma hierarquia e programado para ser executado em determinado tempo e/ou ordem; *Compasable* é a principal interface do JMSL. Ela define métodos para o escalonamento dos objetos. Na hierarquia, o objeto pai envia mensagens aos filhos indicando o momento dele ser iniciado e este, por sua vez, retorna ao pai o momento em que encerrou sua execução.

Por ser uma sub-interface da interface base *Playable,* os objetos que implementam a *Composable* são obrigados a sobrescrever o método *play*() retornando o momento do final da execução do mesmo.

Além dos métodos de controle de execução musical como: *play(), stop(), halt(), finish(), repeat()* e outros, a interface *Composable* ainda define métodos para transpor tonalidades, controlar andamento e visualizar a hierarquia onde está inserida.

A maior parte das classes que veremos a seguir implementam a interface *Composable*, portanto, deixaremos os exemplos para mais adiante.

**MusicJob**

É a classe base do JMSL, uma implementação direta da interface *Composable* que controla as ações repetidamente no tempo. Generaliza importantes classes como a *ParallelCollection*, *SequentialCollection* e *MusicShape*;

Além dos métodos definidos na interface *Composable*, o *MusicJob* tem eventos que permitem inclusão de atrasos e pausas entre os eventos e também a capacidade de ser serializado e, conseqüentemente, persistido, propiciando seu uso na Web. Qualquer

código Java que precise ser gerenciado no tempo pode fazer uso do *MusicJob*, desde do envio de notas musicais até mudança da cor do fundo ou exibição de uma mensagem.

Dois tipos de abordagens OOP podem ser usadas em relação ao *MusicJob*: a sobre-escrita dos métodos em sub-classes ou a inclusão de funcionalidade através do método *setPlayable()* na própria *MusicJob* instanciada. O Exemplo 2 mostra como é possível estender a classe *MusicJob* com uma de nossa própria implementação.

```
package JMSLTutorial;
import com.softsynth.jmsl.*;
public class MinhaMusicJob extends MusicJob {

/** Sobre-escreve repeat() para prover nossa funcionalidade*/
public double repeat(double playTime) throws InterruptedException {
System.out.println("Eu sou uma MusicJob");
return playTime; // retorna o tempo da conclusão
}

/** Teste */
public static void main(String args[]) {
MinhaMusicJob minhaMusicJob = new MinhaMusicJob ();
minhaMusicJob.setRepeats(10);     // repete 10 vezes
minhaMusicJob.setRepeatPause(1.5);// espera 1.5s entre as       //repetições

minhaMusicJob.launch(JMSL.now());   // inicia
```

Exemplo 2: Sobre-escrevendo os métodos da MusicJob

**MusicShape**

A *MusicShape* é um lista multidimensional, onde as linhas são chamadas de elementos e as colunas de dimensões. Os dados contidos em uma *MusicShape* não tem forma definida, ficando a cargo de outra classe a interpretação(*Interpreter*) dos mesmos. Esta característica garante flexibilidade a classe, podendo a mesma representar melodias, harmonias, ritmos ou qualquer outro tipo de dado, inclusive não musical, que precise ser escalonado no tempo.

Os dados podem ser tocados pela própria *MusicShape* (implementa a *Composable*) ou por qualquer outra *Composable*;

Os dados abaixo representam uma melodia, onde a dimensão 0 (primeira coluna) guarda a duração, dimensão 1 (segunda coluna) guarda a nota MIDI e a dimensão 2(terceira coluna) guarda a velocidade(*velocity*) MIDI. É importante salientar que os dados não precisam ter uma interpretação obvia, como no MIDI.

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1.0 | 65.0 | 120.0 |
| 1 | 1.5 | 68.0 | 110.0 |
| 2 | 0.5 | 72.0 | 100.0 |

| 3 | 1.0 | 60.0 | 120.0 |
|---|---|---|---|
| 4 | 1.0 | 63.0 | 80.0 |

Tab. 2: Dados em uma MusicShape.

Os dados contidos em uma *MusicShape* podem ser modificados em tempo real pelo usuário usando o *MusicShapeEditor*, contido no JMSL; Por exemplo, os dados de uma *MusicShape* podem ser definidos e tocados algoritmicamente, como mostra o trecho de código do Exemplo 3.

```
.
.
for (int i=0; i < numberOfElements; i++){
 double value = centralValue + JMSLRandom.choosePlusMinus(i);
 myShape.add(value);
}
```

Exemplo 3: Geração randômica dos dados de uma MusicShape

## Instrument

O instrumento JMSL toca os dados que são interpretados quando o método *play*() da interface *Instrument* é disparado. Para uma interpretação customizada dos dados, sobrescreve-se o método *play();* Seja qual for o tipo de dado (nota MIDI, parâmetro do JSyn, parâmetros de cor etc.) a interpretação é de total responsabilidade do *Instrument,* ficando a cargo dele escolher enviar para um interpretador customizado se este for o caso.

Qualquer classe que implemente a interface *Instrument* “toca” os dados de uma *MusicShape* (um vetor de ponto flutuante). Quando a *MusicShape* é carragada, ela pega um elemento por vez e entrega ao seu *Instrument*, através do método play(), assim o *Instrument* pode interpretar diretamente o dado ou passar ao seu *Interpreter* (interpretador) customizado.

A interface *Instrument* tem uma classe adaptadora que implementa os seus métodos para facilitar a programação: a *InstrumentAdapter*. O Exemplo 4 mostra como simplesmente imprimir os dados declarando nossa própria subclasse da *InstrumentAdapter* e sobrescrevendo o método o método *play().*

Note que no código do Exemplo 4, o momento da execução é passado como argumento (*playTime*) do método *play(),* que por sua vez retorna o tempo atualizado.

```
class PrintingInstrument extends InstrumentAdapter {

public double play(double playTime, double timeStretch,              double dar[]){
  JMSL.out.println("Dados do vetor DAR");
  JMSL.printDoubleArray(dar);
  JMSL.out.println();
  return playTime + (dar[0]*timeStretch); //dar[0]=duração
}
```

```
}
```

Exemplo 4: Como usar o IntrumentAdapter.

### Interpreter

Como já mencionado, o *Instrument* usa um *Interpreter* para interpretar os dados. As classes que implementam a *Instrument* já tem um *Interpreter default*, entretanto o mesmo pode ser substituído para uma interpretação customizada. Este procedimento pode ser visto no Exemplo 5, onde criaremos nosso próprio *Interpreter* e o "*plugaremos*" ao instrumento *default* de um *MusicShape*.

Para definir nosso *interpretador* para o instrumento da *MusicShape* utilize o seguinte *sintaxe*: "myMusicShape.getInstrument().setInterpreter(new PrintingInterpreter());

```
class PrintingInterpreter extends Interpreter {

  /** Sobre-escrito para uma interpretação customizada. */
  public double interpret(double playTime, double timeStretch, double dar[], Instrument
ins) {
     JMSL.out.print(getName() + " chamado por " + ins + " com ");
     JMSL.printDoubleArray(dar);
     return playTime + (dar[0]*timeStretch);
}
}
```

Exemplo 5: Criando um interpretação própria

### Player

Às vezes é razoável ou pelo menos sensato executar várias *MusicShape*´s com o mesmo Instrumento. Por exemplo, uma peça musical pode ter 100 compassos, e será executada em uma *DrumMachine*. Depois de montar um *Instrument* para executar o trabalho (um *MidiInstrument*, por exemplo, que envia no canal 10), seria conveniente ter uma única *MusicShape* para representar cada compasso. Estes compassos deveriam ser executados em seqüência, do primeiro ao último, usando o mesmo Instrumento. Em outro caso, algumas poucas *MusicShapes* podem compor uma mesma peça e devem ser tocadas simultaneamente, mas com instrumentos e interpretadores diferentes. É para executar este tipo de ação que o *Player* existe.

Como o *Player* é uma subclasse da *SequentialCollection*, o padrão é que as *MusicShapes* sejam executadas seqüencialmente, mas isto pode ser modificado utilizando a interface *Behavior*.

## JMusic

jMusic é uma biblioteca de programação musical de alto nível escrita em Java, desenvolvida na Universidade de Tecnologia de Queensland (Brisbane – Austrália). Assim como algumas outras linguagens e ambientes de programação, o jMusic foi projetado para ser usado por músicos e não programadores, com a função de auxiliá-los no processo composicional. Entretanto, muitos programadores fazem uso do jMusic

como uma poderosa API para desenvolver aplicações musicais, em especial instrumentos digitais, ambientes de educação e análise musical.

Pode-se citar como vantagens do jMusic:

Por ter sido escrito em Java, o jMusic consegue manter as principais virtudes da linguagem como sua flexibilidade e portabilidade, além de usar conhecimento prévio de Java no aprendizado do jMusic;

JMusic é gratuito e aberto distribuído sobre a GNU General Public Licence;

Fácil aprendizado e uso, uma vez que foi construído de acordo com as convenções musicais tradicionais;

O material musical construído em outros programas e interfaces musicais pode ser importado ou exportado com facilidade;

Grande variedade de ferramentas utilitárias para visualização e audição da composição em construção;

**JMC (J-Musical-Constants): A interface raiz do jMusic**

Parâmetros dos protocolos de comunicação, síntese e manipulação de áudio podem não ser muito intuitivos e de fácil leitura para os músicos, público alvo do jMusic. Para tornar a codifição mais legível e musical, o jMusic fornece uma interface que define constantes associando nomes conhecidos a seus valores computacionais. Exemplo, o valor MIDI para um Dó central (4 oitava) é 60. Em classes que implementam a JMC é possível substituir o valor 60 por um intuitivo C4.

Atualmente a JMC definie constantes para altura da nota, ritmo, dinâmica, e MIDI. A Tabela 3 lista um resumo dos principais valores.

| **Descrição da Constante** | Valores |
|---|---|
| Pitch (Altura da Nota) | Combinação das Notas (A...G) + Acidente (S = sustenido ou F =bemol) + oitava (1..9)<br>Na oitava ainda é possível usar o N para valores negativos. Ex. CN1.<br>Exemplos: C4, FS3, GF9, D1 |
| Cálculo de Freqüência | Existe ainda uma função que calcula a freqüência de uma nota. A FRQ[n], onde n é a nota. N pode ser o valor MIDI da nota ou uma das constantes acima mencionadas.<br>Exemplo FRQ[36] ou FRQ [C4] |
| Pausa | A pausa é considerada um nota especial no jMusic. A constante REST substitui a nota.<br>Ex; |

| Valores Rítmicos | Semibreve: WHOLE_NOTE,WN;<br>Mínima: HALF_NOTE, HN, MINIM;<br>Semínima = CROTCHET, C, QUARTER_NOTE, e QN. (valor de referencia = 1.0);<br>Colcheia = QUAVER, Q, EIGHTH_NOTE, ou EN;<br>Semicolcheia= SEMI_QUAVER e SQ;<br>Fusa = THIRTYSECOND_NOTE, TN, TSN;<br>Os sufixos DOTTED,DOUBLE_DOTTED podem ser usados para aumentar metade do valor e dobrar o valor da nota, respectivamente.<br>O prefixo TRIPLET diminui em 1/3 o valor da nota e o DEMI, a metade.<br>* olhar documentação para mais valores |
|---|---|
| Dinâmica | SILENT = 0, PPP = 10, PP = 25, P = 50, MP = 60, MF = 70, F = 85, FF = 100,<br>FFF = 120; |
| Panning(Stereo) | PAN_CENTRE = 0.5, PAN_LEFT = 0.0, PAN_RIGHT = 1.0; |
| Duration Articulation | STACCATO = 0.2, LEGATO = 0.95, SOSTENUTO = 1.2, TENUTO = 1.0; |
| Timbre (MIDI Program Change) | ABASS, AC_GUITAR, ACCORDION, ACOUSTIC_GRAND, GUITAR, AGOGO, AHHS, ALTO, ALTO_SAX, APPLAUSE, PIANO . |
| Escalas | MAJOR_SCALE , MINOR_SCALE , HARMONIC_MINOR_SCALE , MELODIC_MINOR_SCALE , NATURAL_MINOR_SCALE , DIATONIC_MINOR_SCALE , AOLEAN_SCALE , DORIAN_SCALE , LYDIAN_SCALE , MIXOLYDIAN_SCALE , PENTATONIC_SCALE , BLUES_SCALE<br>TURKISH_SCALE ,INDIAN_SCALE |

Tab. 3: Constantes da JMC

**Representações dos dados no jMusic**

As informações musicais são organizadas e armazenadas tal qual a pauta musical, ou seja, a partitura(*score*) é formada de partes (*part*), que por sua vez, é constituída de frases musicais(*phrases*) onde estão contidas as notas.

A nota é a estrutura básica usada no jMusic e traz consigo uma série de atributos, como: altura(*pitch*), volume (*dynamic*), figura de tempo (*RhythmValue*), controle de *stereo* (*pan*), duração (*duration*) e os acidentes (sustenido e bemol). Frase pode ser vista como vozes de uma parte; por exemplo, no piano cada mão tocaria uma voz. A frase só tem um atributo realmente importante, a lista de notas. Veremos adiante que existem classes distintas para cada estrutura musical que se deseja construir.

Uma parte contém um vetor de frases (sons familiares) e também um título ("Violino 1" por exemplo), um canal e um instrumento (em MIDI, um número de mudança de programa - em Áudio, um índice no vetor de instrumentos). O Exemplo 6 mostra como estas estruturas musicais são ligadas para produzir um som.

```
import jm.JMC;
import jm.music.data.*;
import jm.util.*;
import jm.audio.*;

public final class SonOfBing implements JMC{
public static void main(String[] args){
Score score = new Score(new Part(new Phrase(new Note(C4,
MINIM))));
Write.midi(score);
Instrument inst = new SawtoothInst(44100);
Write.au(score, inst);
}
}
```

Exemplo 6: Geração simples de um som usando jMusic

Quando cria-se uma frase, passa-se como parâmetro para o construtor um *double*. Este paramento é o tempo de início medido em *beats*. A tempo de referencia de um *beat* é um QUARTER_NOTE / CROTCHET (1.0), todos os outros valores de ritmos são relativos a estes.

Pausas também são consideradas notas e como tal, tem uma duração definida. A constante REST representa a pausa, ou seja, o intervalo de silêncio.

**Estrutura para Geração de Áudio em Tempo-real**

O jMusic também possui classes que permitem a composição em tempo real: RTLine e RTMixer. A Figura 5 mostra as classes necessárias para os dois tipos de composição (off-line e tempo real).

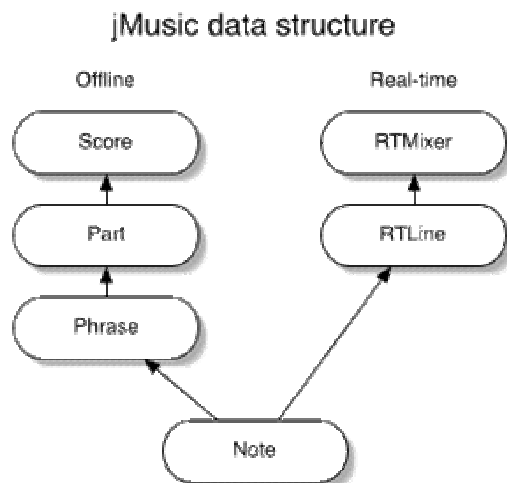



Fig. 5: Classes necessárias nas composições Offline e Real-time

A RTLine é uma classe abstrata que estendemos para prover processamento de áudio em tempo real. Esta pode ser entendida como uma mistura da *Part* com a *Phrase*, pois a mesma envia notas diretamente para peça, como a *Part*, apesar de ser monofônica como a *Phrase*.

O RTMixer recolhe todos os fluxos de áudio gerados pelos RTLines, junta-os e envia para o dispositivo de som (usando JavaSound.) Normalmente, vai existir somente um RTMixer por aplicação mas serão vários RTLines, um para cada parte da música.

Note que a classe *Note* é central para ambas as estratégias de composição.

**Gerando Música Polifonia**

Existem duas maneiras de se escrever músicas polifônicas no jMusic, equivalentes ao mundo real. A primeira é gravar várias vozes em uma única parte (*Part*), como ema fuga no órgão, onde o compositor escreve para duas ou mais partes simultâneas para uma mão tocar. A segunda é escrever para duas ou mais partes, como em um quinteto de madeiras e sopros. A partitura de uma orquestra é uma mistura dos dois tipos, contendo instrumento monofônico, como a flauta, e polifônicos, como o piano.

Os trechos de código abaixo fazem parte de uma aplicação que grava uma peça polifônica formada por 3 instrumentos monofônicos. Os trechos foram divididos para uma melhor narrativa e compreensão dos mesmos.

```
Score score = new Score("Row Your Boat");
Part flute = new Part("Flute", FLUTE, 0);
Part trumpet = new Part("Trumpet", TRUMPET, 1);
Part clarinet = new Part("Clarinet", CLARINET, 2);
```

Exemplo 7: Estruturação da peça polifonica

O trecho do Exemplo 7, simplesmente declara as estruturas de maior hierarquia que irão compor a peça polifônica em questão. Como é possível observar, a partitura (*score*) será formada de 3 partes (*Part*) correspondendo a flauta, trompete e clarinete, cada instrumento no seu próprio canal MIDI.

```
int[] pitchArray ={C4,C4,C4,D4,E4,E4,D4,E4,F4,G4,C5,C5,C5,G4,G4,G4,E4,
E4,E4,C4,C4,C4,G4,F4,E4,D4,C4};
```

```
double[] rhythmArray = {C,C,CT,QT,C,CT,QT,CT, QT,M, QT, QT, QT, QT, QT,
QT, QT, QT, QT, QT, QT, QT, CT, QT, CT, QT,M};

Phrase phrase1 = new Phrase(0.0);
phrase1.addNoteList(pitchArray, rhythmArray);
```

Exemplo 8: Montagem da melodia em peça polifônica

O Exemplo 8 mostra como as frases musicais são formadas. Diferentemente do que o músico está acostumado na notação clássica, as notas e suas durações estão divididas em dois vetores: pitchArray e rhythmArray.O primeiro carrega informações da altura da nota e o segundo sua duração.Sendo assim, a primeira nota a ser tocada seria um Dó Central com duração de uma Semínima (*crotchet/quarter note*).

A primeira frase (*phrase1*) foi criada com os vetores acima e tempo de início igual a 0, ou seja, é tocada imediatamente após a solicitação.

```
Phrase phrase2 = phrase1.copy();
phrase2.setStartTime(4.0);
Phrase phrase3 = phrase1.copy();
phrase3.setStartTime(8.0);

Mod.transpose(phrase1, 12);
Mod.transpose(phrase3, -12);

Mod.repeat(phrase1, 1);
Mod.repeat(phrase2, 1);
Mod.repeat(phrase3, 1);
```

Exemplo 9: Definição dos parâmetros das frases

No exemplo 9, trabalha-se a tonalidade e atrasos de duas novas frases que são cópias da frase 1. Esta operação e irá fazer soar polifonicamente as notas através da sobreposição das frases.

```
flute.addPhrase(phrase1);
trumpet.addPhrase(phrase2);
clarinet.addPhrase(phrase3);
score.addPart(flute);
score.addPart(trumpet);
score.addPart(clarinet);

Write.midi(score, "rowboat.mid");
```

Exemplo 10: Vinculando os objetos composicionais

Por fim, o Exemplo 10 mostra como vincular os objetos e gravar o resultado para um arquivo MIDI;

**Exibindo material musical**

jMusic pode ser visto como um código para descrever música. A música pode ser representada em diversas notações diferentes levando-se em consideração aspectos como: conhecimento musical do usuário, quantidade de frases e partes musicais, objetivo do software em desenvolvimento etc. O jMusic possui uma classe utilitária usada para visualizar de diversas formas a música.

A forma mais comum de representar a música é através na da notação clássica, denominada no jMusic como *Common Practice Notati*on, ou mesmo CPN. Atualmente, a implementação da CPN permite que somente frases musicais sejam exibidas e outras pequenas operações como: salvar em MIDI, modificar notas e formulas de compasso, tocar etc. O Exemplo 11 demonstra como é possível representar uma frase musical na forma de CPN usando a classe *View*. O resultado pode ser visto na Figura 6.

```
import jm.music.data.*;
import jm.JMC;
import jm.util.*;

public class Dot01 implements JMC {
    public static void main(String[] args) {
        Note n;
        n = new Note(C4, QUARTER_NOTE);
        Phrase phr = new Phrase();
        phr.addNote(n);
        Mod.repeat(phr, 15);
        View.notate(phrase);
    }
}
```

Exemplo 11: Representando uma frase musical na notação clássica

Fig. 6: Representação na forma de pentagrama

Apesar de bastante útil, a CPN possui restrições de implementação e possui difícil leitura para leigos em música. Uma das maiores desvantagens é fato de não consegui exibir duas frases ao mesmo tempo em uma única janela. Uma notação que vem ajudar a transpor estas restrições é a *ShowScore*; um ponto intermediário entre a CPN e a *Piano Roll* (*Screcth*) que veremos a seguir.

Na ShowScrore, frases ( e até mesmo partes) podem ser vistas simultaneamente, como a Figura 7 mostra. Dentre as funções mais significativas da ShowScore, cita-se:

Cada parte tem uma cor definida randomicamente, ou seja, notas da mesma parte possuem a mesma cor, facilitando a visualização;

A intensidade das cores representam a dinâmica da nota;

Notas cromáticas são indicadas com o símbolo do sustenido '#';

Cada frase tem um retangulo em volta facilitando o reconhecimento;

Barras horizontais e verticais ajudam o posicionamento no tempo;

Zoom;

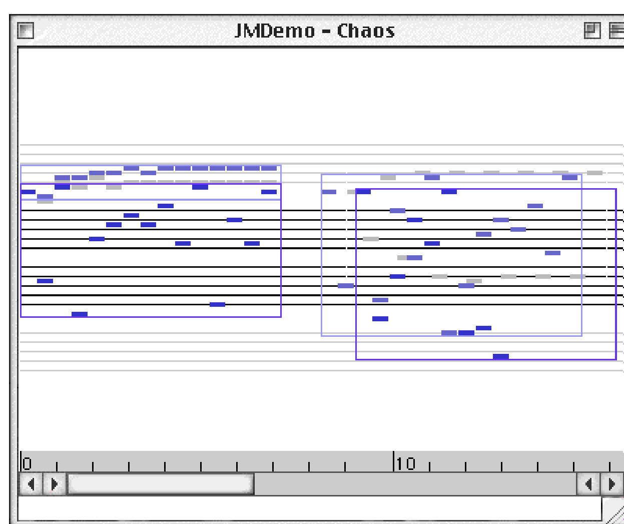



Fig. 7: Notação ShowScore.

*Screcth* é um visualizador compacto, que organiza as notas cromaticamente ao invés de pautas. Este tipo de interface também é conhecido como *piano-roll* e permite inclusão de notas em tempo de execução. A Figura 8 mostra um exemplo.

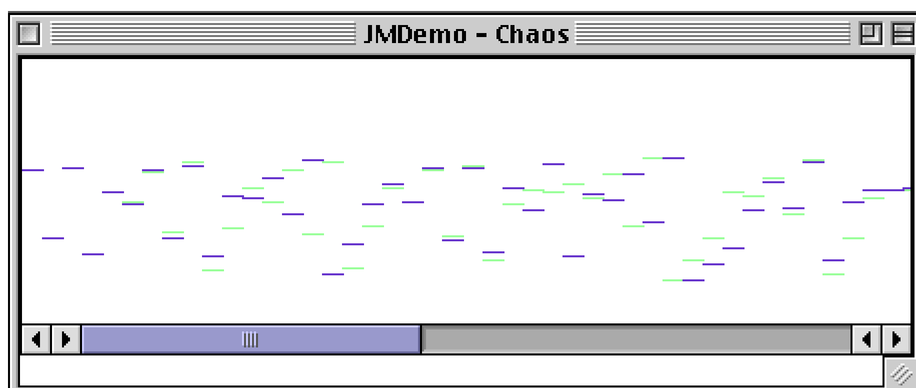



Fig. 8: Notação Screcth (Piano Roll).

Tanto o *ShowScore* como o *Screcth* possuem métodos sobrecarregados que aceitam desde frases(*phrases*) até partituras(*score*).

**Arpegios**

Uma importante classe para manipulação de melodias é a *Mod*. Através dela é possível repetir, inverter e até mesmo arpejar em forma de palíndromos.

**Sintetizando notas através de instrumentos virtuais (digitais)**

O jMusic possui um completo pacote de áudio que envolve síntese sonora, construção de instrumentos virtuais e uso de *samples* em composições.

Instrumentos virtuais são classes usadas para renderizar partituras do jMusic em arquivos de áudio (ou saída de áudio em tempo real) e são feitos a partir de objetos de áudio organizados em uma estrutura hierárquica denominada "corrente" (*chain*). Estas correntes de objetos de áudio definem todas as propriedade do som que se ouve e o tipo de síntese a ser utilizada para gerá-lo.

Todos os instrumentos virtuais devem estender a classe *Instrument*. Isto permite que o novo instrumento criado se preocupe apenas com os aspectos de timbre do mesmo, deixando os detalhes de comunicação entre os objetos de áudio e as notas para a superclasse.

A qualidade do som produzida pelo instrumento (*sample rate*) pode ser modificada de acordo com a aplicação, entretanto deve-se ficar atento para o *sample* rate não ser menor que 8000, levando –se em consideração a fórmula de Nyquist.

O arranjo hierárquico dos objetos é necessário porque a *Instrument* é essencialmente uma classe para gerar / processar *samples* (amostras de som). Os *samples* são passados de uma objeto para outro através da corrente, formando um fluxo de samples; como mostra a Figura 9.

Fig. 9: Fluxo de dados de áudio.

Os principais objetos de audio são:Wavetable, Envelope, Volume, Panner, SampleIn, SampleOut, Filter, Add, Splitter, e WhiteNoise.

A organização hierárquica de objetos de áudio (e música) não é algo recente. Uma das primeiras linguagens de programação para música, Music V escrita por Max Marthews nos Laboratórios da Bell –EUA, estabelecia a convenção de usar digramas de fluxos de sinais, como a Figura 10 mostra. Nela, o objeto de cima é um envelope que pega dois atributos para gerar uma amplitude de saída, que por sua vez, é um paramento de entrada junto com a freqüência em um oscilador que gera ondas senoidais. O mesmo diagrama transposto para o jMusic ficaria como na Figura 11.

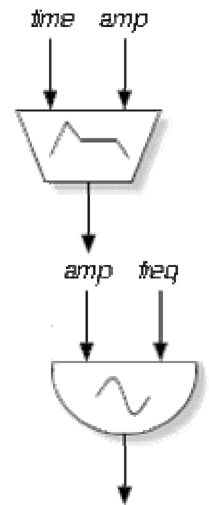



Fig. 10: Diagrama Music V

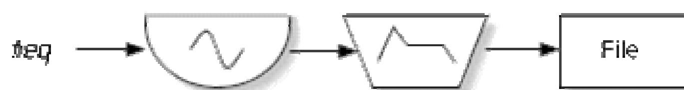



Fig. 11: Diagrama jMusic

O trecho que código do Exemplo 12 codifica o diagrama da Figura 12. Nele é possível observar como ocorre a síntese de áudio no JMusic.

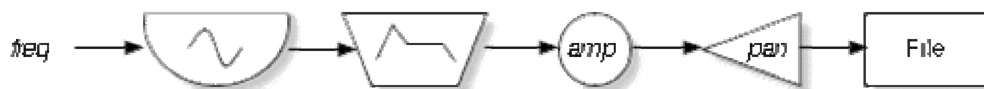



Fig. 12:Exemplo de diagrama de audio.

```
public void createChain(){
    WaveTable wt = new WaveTable(this, this.sampleRate,
    Oscillator.getSineWave(this.sampleRate), channels);
    Envelope env = new Envelope(wt, pointArray);
    Volume vol = new Volume(env,(float)1.0);
    StereoPan span = new StereoPan(vol);
    SampleOut sout = new SampleOut( span, "jmusic.tmp");
}
```

Exemplo 12: Codificação de um diagrama de áudio

Como visto no Exemplo 12 os instrumentos virtuais podem sintetizar as notas através dos objetos de áudio como o *Wavetable*.

Em Java, as *WaveTables* (tabelas de ondas) são vetores preenchidos com valores calculados ou extraídos de gravações digitais de sons acústicos. Com esses atributos do som devidamente processados consegue-se obter o som desejado do instrumento. Outros tipos de síntese como a granular, aditiva e subtrativa também estão disponíveis no jMusic.

Outra opção dos instrumentos virtuais é usar amostras de som (*samples*). Nesta técnica, pequenos arquivos de som são gravados e definidos como notas que podem ter sua

freqüência (*pitch*) e duração modificados e seqüenciadas para gerar uma composição. O Exemplo 13 mostra uma aplicação utilizando *samples*.

```
public class WaveformExample implements JMC {

  public static void main(String[] args) {
        new WaveformExample();
  }

  public WaveformExample() {
        Note n = new Note(C4, MINIM);
        Score score = new Score(new Part(new Phrase(n)));
        Instrument sineWave = new SimpleSineInst(44100);
        Write.au(score, "WaveformExample.au", sineWave);
  }
}
```

Exemplo 13: Uso de samples.

Uma série de instrumentos virtuais prontos para usar estão disponibilizados no site da jMusic.

**Principais Classes e Intefaces**

As classes e interfaces do jMusic estão organizadas em pacotes por funcionalidade, sendo assim, a Tabela 4 lista os principais pacotes.

| Pacote | Principais Características |
|---|---|
| jm | Simplesmente possui a interface JMC de alta importância para o desenvolvimento com o jMusic. Todos os outros pacotes estão contidos neste pacote. |
| jm.audio | Contém as classes necessárias para qualquer manipulação do áudio, incluindo: criação de intrumentos virtuais, síntese e samples. |
| jm.constants | Contem as super interfaces para a interface JCM; |
| Jm.gui.* | Formado de 7 “sub-pacotes” (jm.gui.cpn ,.jm.gui.graph jm.gui.helper ,jm.gui.histogram ,jm.gui.show ,jm.gui.sketch ,jm.gui.wave), este conjunto de pacotes contém úteis classes para ajudar na interface gráfica da aplicação. |
| jm.midi | Manipulação do MIDI de uma forma geral, dos eventos aos dados. |
| Jm.music.* | Classes e ferramentas que ajudam na composição de músicas computacionais. |

| | |
|---|---|
| jm.util | Classe utilitárias. |
| jmms | Integração com o pacote MIDI Share |
| jmqt | Pacote d eintegração com o QuickTime |

Tab. 4: Principais pacotes do JSML.

Devido à generalidade deste pacote, seria uma tarefa deveras difícil eleger as principais classes, visto que isto dependeria dos objetivos a serem alcançados. Entretanto, acredita-se que nos exemplos postados ao longo da apresentação do jMusic, as principais classes tenham sido devidamente citadas e referenciadas para um estudo mais profundo, se necessário, na documentação do jMusic.

**Considerações sobre o jMusic**

É um pacote bem completo e com interessantes recurso tanto para áudio como para tratamento MIDI.

O tutorial é claro e feito para quem não tem experiência em programação, como os músicos.Entretanto é pouco atrativo para quem já possui experiência em programação e Java, pois o mesmo mistura os conceitos e definições, obrigando quem quer aprender simplesmente jMusic a ler todo o tutorial.

## Wire /Wire Provider

Nas primeiras versões do Java Sound existia um problema de implementação que só foi resolvido, ainda que parcialmente, na versão JDK1.4.1 (2003). O problema era mais evidente e grave quando a aplicação rodava sobre o MS Windows. Basicamente, o JavaSound da versão 1.3 não conseguia trocar mensagens MIDI com dispositivos externos.

Para suprir a falta de comunicação MIDI com dispositivos externos, alguns fabricantes e programadores disponibilizaram bibliotecas que permitem tal comunicação. O problema é que estas bibliotecas não são escritas em Java, e isto pode afetar a portabilidade da aplicação que a usa. O melhor mesmo é verificar se as limitações atuais do Java Sound são mesmo um problema para sua aplicação e avaliar o uso destas bibliotecas em detrimento da possível perda da portabilidade.

Duas versões do Wire estão disponíveis. A primeira chama-se simplesmente Wire e foi escrita por uma pequena empresa alemã chamada Bonneville. É um exemplo típico de JNI (*Java Native Interface*), ou seja, classes escritas em C++ e aproveitadas pelo Java. Só funciona no Windows.

A outra versão foi escrita por Dr.-Ing. Gerrit Gehnen e foi baseada na versão de Niel Gorisse (Bonneville) e no pacote para Linux da Tritonus. Por ser um service provider (ver Java Sound – SPI) basta copiar dois arquivos para que a JRE reconheça e disponibilize os dispositivos MIDI.

Nestas duas versões o uso do JavaSound é obrigatório, entretanto se o usuário quiser usar outra biblioteca musical ou rodar em plataforma diferente do Windows, terá que usar uma outra biblioteca de comunicação MIDI, tais como: Java MIDI, NoSuch ou MidiShare.

## JavaMIDI - 2001

JavaMIDI é um conjunto de classes, escritas por Robert Marsanyi, que permite uso do MIDI sobre a platforma Java assim como o Wire e o Wire provider com a diferença de rodar sobre Windows e Macintosh.

## NoSuch MIDI

Uma boa biblioteca para trabalhar com MIDI na plataforma Windows. É capaz de lidar com dispositivos MIDI externos, mensagens exclusivas, escalonamento em tempo real da saída, escrita e leitura de MIDI. Não precisa do Java Sound por ser uma implementação independente, o que a faz não rodar em browsers. Gratuita para uso não comercial.

## MIDI Share

Feita em um tempo onde não existia suporte a MIDI no Java sobreviveu ao Java Sound por não apresentar problemas com dispositivos MIDI externos e sobressai-se das demais bibliotecas similares por rodar em *Applets*, mas para tanto, é necessário instalar dois arquivos(bibliotecas nativas) no cliente JMidi e JPlayer;

Dois pacotes:

JMidi: Contem as a maioria das funções do MidiShare ;

JPlayer: Biblioteca que implementa sequenciadores multi-track e sincronizáveis;.

Principais classes:

Midi: Dá acesso as funções do MidiSahre através da biblioteca nativa JMidi;

MidiPlayer: Controla execução do MIDI;

MidiAppl : Esqueleto de aplicações MIDI;

MidiTask: escalonador de tarefas MIDI;

MidiFileStream: Permite leitura e escrita de arquivos MIDI;

## MIDI Kit

Esta biblioteca é uma boa alternativa para quem precisa de portabilidade nas aplicações MIDI. Com ela é possível desenvolver com facilidade aplicações MIDI simples e locais, bem como servidores de processamento MIDI distribuído em rede que se configura dinamicamente para atender as necessidade de cada cliente.

Da mesma forma que outras bibliotecas que usam rotinas não escritas em Java, o MIDI Kit necessita que arquivos sejam postados no cliente para funcionar em Applets.

Para usar os recursos MIDI da biblioteca basta estender as MIDIProcessor ou Performer.

Apesar do nome, o MIDI Kit também consegue processar áudio através da classe *AudioProcessor* e suas sub-classes. Até o momento, somente o processamento de arquivos de som e execução dos mesmos estão disponíveis.
O modo de funcionamento do MIDI Kit é baseado na arquitetura do NeXT Music Kit (Stanford – CCRMA, 1998). É construída sobre o preceito de centralizador de eventos que ficam em *loop* (*EventLoop*). Eventos de vários tipos, incluindo eventos MIDI, interface gráfica, controle de sincronismo e tempo, são criados por *threads* e entrada/saída e colocadas em um fila (FIFO) central. Na *thread* principal, chamada de *Main EventLoop*, os eventos são removidos, um de cada vez , e processados pelos seus respectivos *Processors*. Como todos os eventos são processados por uma única *thread*, não é necessário o esforço extra em código para manter a sincronização.

**Principais Classes**

MIDIProcessor: É onde a ação está! Várias subclasses da MIDIProcessor desempenham operações como o mapeamento da notas de entrada, sequenciamento, execução de som, envio de mensagens MIDI e até mesmo encaminhar eventos MIDI (MIDIEvents) para outros processadores (MIDIProcessor) na rede.

**AudioProcessor**: Controla todo o processamento de áudio digital do MIDI Kit que, até o momento, não é muito.

**Performers**: É uma subclasse da MIDIProcessor que interage com a Conductor para perfazer ações em um determinado momento.

**Conductor**: Outra subclasse da MIDIProcessor que também controla uma fila ordenada por tempo de eventos musicais, mas em sua própria *thread*. Quando chega o momento do evento ser executado, o evento é encaminhado a fila principal (EventLoop). Possui duas noções de tempo: beat e cronometro(hh:mm:ss:ms).

**Time**: É um objeto estático que tem a função de ser o relógio de todo o sistema.

## JFugue

JFugue é um conjunto de classes Java para programação musical. Esta biblioteca usa simples *strings* (cadeia de caracteres) para representar dados musicais, incluindo notas, acordes e mudanças de instrumentos. JFugue também permite a definição de música através de padrões que podem ser transformados para criar novos segmentos musicais derivados de peças musicais já existentes.

JFugue pode escrever arquivos MIDI e simplifica a programação musical.

## Tritonus

Tritonus é uma implementação com o código aberto mais limpa e eficiente do Java Sound 1.0 para Linux. É distribuída de acordo com os termos da GNU Library General Public License.

Os coordenadores do projeto são importantes pesquisadores e desenvolvedores da área, como Matthias Pfisterer e Florian Bomers, entretanto a maior parte do código foi desenvolvido por voluntários. Isso permite a distribuição gratuita da API, mas deixa a desejar em suporte e freqüência de lançamento de novas versões, sendo a última atualização datada ao ano anterior a escrita deste trabalho.

Tritonus é mais estável que o Java Sound, mas ainda não está completo e perfeito(muito longe disso). Apesar da precariedade de funcionalidade e de sua limitação de plataforma (somente Linux), o Tritonus é quase uma unanimidade entre aqueles que programam em Linux.

**Funcionalidades de Áudio**

Existe uma relação unilateral do Java Sound em relação ao Tritonus, ou seja, o que for feito para o Java Sound, através dos pacotes de SPI, deve funcionar no Tritonus. Isto é muito cômodo para o pessoal do Tritonus, mas também auxilia o desenvolvimento do Java Sound, pois é mais fácil estender um serviço do que criá-lo inteiramente. Isso se aplica no suporte a compressão e descompressão de dados de áudio.

Suporta leitura e gravação de arquivos no formato .au, .aiff and .wav, assim como Java sound,mas diferentemente deste não possui pacote s para Services Providers estenderem sua capacidades, tendo os mesmos que trabalharem direto no código fonte.

Uma outra diferença do Java Sound é o fato do Tritonus não possuir um Mixer, tendo que buscar algum disponível no sistema. Isto facilita a integração do código com infra-estrutura de hardware da máquina mas também dificulta a criação de novos dispositivos de áudio,que tem que prover implementações para as interfaces Mixer, SourceDataLine, TargetDataLine e Clip;

Standard linear : Mono e Stereo, Big e little endian, 8, 16, 24 e 32 bits, A-law and u-law codec;

Mp3 decoder (Javalayer 0.0.7 incorporado ao Tritonus, Java MP3 Player Project criação de MP3 usando a biblioteca LAME). O Java Sound não prove mais suporte a MP3 por problemas autorais.

**Funcionalidades do MIDI**

As partes principais do suporte ao MIDI estão implementadas baseada no seqüenciador ALSA. Uma implementação baseada no Midi Share está em desenvolvimento.

- Escrita e leitura de arquivos MIDI;.
- Um sistema de síntese de software (TiMidity) não muito estável.
- Interface para sintetizadores em hardware (com restrições)
- MIDI IN/OUT (acesso a dispositivo MIDI externos)

## JASS (Real-time audio synthesis)

JASS (Java Audio Synthesis System) é uma unidade geradora baseada em ambientes para programação de síntese de áudio. Escrita em Java puro, o ambiente baseia-se em um pequeno número de interfaces e classes abstratas que implementam a funcionalidade necessária pra criação de *patches. Patches* são criados juntando unidade geradoras em complexas estruturas e podem renderizar sons em tempo real. A comunicação com o hardware de áudio foi escrita com o Java Sound e, em algumas plataformas, JNI (*Java Native Interface*).

A biblioteca se propõe a ser uma alternativa ao Java Sound com baixa latência devido a métodos nativos. Atualmente possui implementação para Linux (ALSA e OSS),

Macintosh (OS/X) e Windows (DirectX, ASIO *blocking* API, ASIO *callback* API). Todas as implementações, exceto ASIO *callback*, utilizam uma classe de entrada/saída de audio em tempo real escrita em C++ por Gary P. Scavone;

JASS tem o código fonte aberto e está disponível para uso não comercial;

## Jsyn

Jsyn permite o desenvolvimento de programas Java para computação musical. Pode-se rodar como aplicações stand-alone ou como applets em webpages (usando o plug-in). Escrito em C para prover síntese em tempo real, o Jsyn pode ser usado para gerar efeitos de som, ambientes de áudio ou música. Jsyn se baseia no tradicional modelo de unidades geradoras que juntas podem gerar sons complexos.

- Síntese em tempo real e de alta fidelidade usando a CPU;
- Biblioteca de unidades geradores incluindo osciladoras, filtros, envelopes, geradores de ruídos e efeitos.
- Todas a operações usam precisão de 32 bits.
- Pode-se combinar samples o sons sintetizados em tempo-real;
- Fácil uso das classes Java para criar, conectar e controlar as unidades geradoras;
- Uso de time-stamping para programação de eventos no tempo;
- Uso de fila de samples e dados do envelope para programar repetição e colagem
- Suporte a entrada de áudio para gravação e processamento de voz;
- Suporte para dispositivos multi-canal;
- Suporte para plataformas Windows, Macintosh e Linux;
- SDK gratuito;

Editor gráfico(Wire) que permite gerar sons conectando unidades geradoras inteiramente,podendo exportar o código Java resultante.

### 6.3.1 Custo

- O Plug-in é grauito;
- Para uso não commercial, o Jsyn SDk também é gratuito.
- Para uso comercial e acesso a ferramentas mais elaboradas o produto deve ser registrado, ou seja, pago.

O principal pacote do Jsyn é o com.softsynth.jsyn.*, entretanto possui diversos sub-pacotes:

- com.softsynth.jsyn.util : classes utilitárias;
- com.softsynth.jsyn.view102: utiliades gráfica do AWT1.0.2;
- com.softsynth.jsyn.view11x: utiliades gráfica do AWT.1.x;
- com.softsynth.jsyn.circuits.*: Exemplo usando circuitos;

## JSCORE

JScore é um analisador sintático em Java para dados de música que gera as partituras no formato XML e o áudio (MIDI). O primeiro objetivo do JScore era demostrar potencialidade do JLex (gerador de analisadores sintáticos) integrado ao JMusic que descreve a música no modo que o JLex deve entender.

## Xemo

Xemo é um projeto que visa desenvolver um robusto e extensível *framework* para composição e notação musical. A principio o projeto não tem relação com qualquer tecnologia, ou seja, apenas define o que tem que ser feito. Passado um ano da criação do projeto, a primeira API de música foi lançada com a intenção de motivar voluntários a se engajarem no projeto.

A API define interfaces para layout e renderização de símbolos musicais em alta resolução em dispositivos 2D, incluindo monitores e impressoras. Com isto, pode-se desenvolver programas de composição interativa, editores de notação musical, jogos e software educacionais.
Atualmente, a API é simplesmente um conjunto de classes gráficas vazias, ou seja, não possuem nenhuma funcionalidade para armazenar ou ler arquivos de áudio ou MIDI ou mesmo executar o que foi escrito.

A API contém funcionalidades especificas para representação musical, execução e composição interativa. Seus elementos base são pacotes para notação musical, representação de estruturas musicais e execução e performance MIDI.

| XEMO™ API | Description |
|---|---|
| Representation | Representação musical em diferentes formatos, incluindo XML (MusicXML e MIDIXML). Pssui mecanismos para definir, construir e manter referencia cruzadas entre diferentes representações. |
| Notation | Define mecanismo para renderização de objetos da notação musical clássica; |
| Playback | Implementa interfaces de alto nível para geração de arquivos MIDI e streams de áudio co suporte da dinâmica, tempo e seleção do patch. |

Tab. 5: Pacotes do XEMO.

O projeto XEMO tem seus objetivos bem estruturados, possui uma boa organização e ótimas idéias, entretanto ainda é muito imaturo para uso no desenvolvimento de

aplicações musicais. Está em uma fase que precisa muito mais de ajuda do que é capaz de ajudar.

## Conclusão

Este trabalho apresentou e analisou 14 ferramentas que agregam ao Java a capacidade de trabalhar com áudio e dados musicais. Foram vistas características, arquiteturas e exemplos destas bibliotecas que foram classificadas em 8 categorias distintas.

Chegou-se a conclusão que as melhores API ´s para trabalho com som (de maneira geral) são o Jsyn, jMusic e Java Sound. Para trabalhar com MIDI, as melhores API´s são jMusic, JMSL e Java Sound. O destaque deste trabalho foi a jMusic que se mostrou completa, estável e de fácil uso em todas as categorias.

# ANEXO C DOCUMENTAÇÃO DOS PACOTES DE FORMAÇÃO E REPRESENTAÇÃO DE ACORDES

Devido ao volume de páginas da documentação, a mesma se encontra no CD em anexo.