UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES

DISSERTAÇÃO DE MESTRADO

Uma Proposta de Ferramenta de Apoio à Educação Musical via *Web* Usando Java e XML

Autor

Cláudio Roberto Araújo
Bacharel em Música – Fagote pela UNICAMP em 1997

Orientador

Prof. Dr. Leonardo de Souza Mendes
PhD em Engenharia Elétrica pela Syracuse University em 1991

Banca examinadora

Prof. Dr. Leonardo de Souza Mendes
UNICAMP/FEEC/DECOM

Prof. Dr. Ivan Luiz Marques Ricarte
UNICAMP/FEEC/DCA

Prof. Dr. Léo Pini Magalhães
UNICAMP/FEEC/DCA

Prof. Dr. Jonatas Manzoli
UNICAMP/IA

Prof. Dr. Ricardo Goldemberg
UNICAMP/IA

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Ar15p

Araújo, Cláudio Roberto
Uma proposta de ferramenta de apoio à educação musical via Web usando Java e XML / Cláudio Roberto Araújo.--Campinas, SP: [s.n.], 2002.

Orientador: Leonardo de Souza Mendes.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Java(Linguagem de programação de computador). 2. XML(Linguagem de marcação de documento) 3. Música – Estudo e ensino. 4. Ensino à distância. I. Mendes, Leonardo de Souza . II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Agradecimentos

Deixo aqui meu muito obrigado às pessoas que direta ou indiretamente contribuíram para a realização deste trabalho.

Muito obrigado a Deus, meu Pai e Criador que colocou em minha mente o sonho de vencer, em meu caminho os obstáculos necessários ao meu crescimento e a minha volta os anjos da guarda que me ajudaram e me ensinaram a transpor com coragem cada dificuldade. Obrigado pela inspiração, pela luz e pela força inquebrantável que me guiou em todos os momentos.

Meu agradecimento mais sincero aos meus amigos Ana Cristina, Mariana e Beto. Vocês me deram uma força e tanto. Amo vocês.

Agradeço de todo o coração à minha amiga, companheira e namorada Ana Rosa, uma flor que tem encantado minha vida com incentivo, colaboração e atenção.

Muito obrigado aos meus colegas de laboratório Luciana, Sílvia, Maurício, Antonio, Ernesto, Fábio, Meire e demais colegas da Feec dentre os quais digo um obrigado particular ao Paulo Fisch e ao Christian Medeiros por tudo que me ensinaram sobre Java.

Obrigado à Lígia, Glorinha, Solange, Stelinha, Carolina, dra. Maria Helena, dr. Luiz Sérgio, Nicodemos, Cação, Sr. Nelson, Garcia, Márcia e Carlos, pessoas que me incentivaram e que de alguma forma colaboraram com o meu trabalho.

Meu muito obrigado a todos os meus colegas músicos que criticaram, opinaram e contribuíram com idéias e sugestões para o trabalho.

Meu agradecimento mais profundo ao meu pai Osvaldo e à minha mãe Maria Rita que me ensinaram desde criança que a vida é dura para que faz corpo mole, mas é mais fácil para quem trabalha duro. Eles me mostraram que somente o trabalho árduo leva a realização de nossos sonhos.

Meu agradecimento aos professores Ivan, Léo Pini, Gilberto Prado e Ting.

Obrigado ao professor Jônatas, co-orientador deste trabalho e grande incentivador.

Obrigado ao professor Ricardo Goldemberg que me acompanha desde a minha graduação.

Agradeço em particular a minha companheira de recitais, professora, incentivadora e amiga Adriana Mendes. Obrigado, de coração.

Meu agradecimento mais especial ao mestre e amigo Leonardo Mendes. Obrigado por acreditar em mim e por me oferecer as condições necessárias para realizar este trabalho. Que o Criador abençoe, projeta e ilumine todos os teus dias.

RESUMO

Neste trabalho é apresentado o JavaMusic, um aplicativo desenvolvido em Java que consiste de um editor de partituras capaz de gerar um arquivo em XML. Com o uso de um *applet* Java, as partituras editadas através do JavaMusic podem ser visualizadas e tocadas numa página da Internet, possibilitando a criação de *sites* voltados ao ensino de música. O trabalho discute o uso de XML associado a *applets* Java para o desenvolvimento de aplicações educacionais para a área de música e propõem um cenário de trabalho no qual um professor de música, especialmente em cursos de nível superior, poderá usar a *web* como uma poderosa ferramenta de apoio à sala de aula de música.

ABSTRACT

This work presents JavaMusic, a Java application that consists of a score editor able to generate an XML file. With the use of a Java applet, a score edited through JavaMusic can be seen and played on an Internet page, allowing the creation of *sites* for music teaching.

It discusses the use of XML files associated to Java applets in the development of educational applications in music, and proposes a work scenario in which a music professor, mainly at undergraduate level, could use the *web* as a powerful tool supporting music education.

ÍNDICE

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

1 Introdução

O desenvolvimento de ferramentas computacionais de comunicação através de redes tem possibilitado a integração entre as mais remotas partes do mundo e conseqüente intercâmbio de culturas e conhecimentos. A Internet conseguiu agrupar num mesmo espaço conhecimentos, lazer e serviços, facilitando a vida das pessoas, propiciando uma comunicação fácil, barata e eficiente e possibilitando que um indivíduo tenha rapidamente acesso a informações de qualquer área do conhecimento.

Vislumbrando as inesgotáveis possibilidades oferecidas pela Internet em termos de difusão de conhecimento e serviços, tem sido despendido grande esforço no sentido de se melhorar suas condições de acessibilidade, interação com o usuário e oferta de aplicações.

Entre as aplicações que mais são pesquisadas, podemos destacar o comércio eletrônico, a transmissão de dados em múltiplas formas de mídia e a educação à distância.

No campo da educação, muitos avanços têm sido alcançados com o desenvolvimento de ferramentas educacionais tais como os ambientes AulaNet[1] e WebCT[2], além de inúmeras pesquisas na área de realidade virtual aplicada à educação, laboratórios e simuladores virtuais e *software* educacional. Além disso, cada área de conhecimento tem movido esforços no sentido de desenvolver ferramentas apropriadas para a transmissão de seus conteúdos.

Muitas propostas de desenvolvimento envolvem a utilização das linguagens de marcação de hipertexto como o HTML e das linguagens extensíveis como XML, e o desenvolvimento de novos

[1] http://anauel.cead.puc-rio.br/aulanet/index.html
[2] http://www.*web*ct.com

mecanismos de interação com o usuário, motivando educadores a repensar seus métodos pedagógicos de forma a aproveitar a contribuição que a Internet tem trazido em termos de acesso à informação e comunicação entre pessoas e grupos.

O desenvolvimento de tecnologias de processamento de dados à distância como os CGIs, *Servlets*, PHP, etc., expandiu as possibilidades de serviços educacionais na rede e as tecnologias de realidade virtual estão possibilitando simulações do mundo real que trazem ao estudante a oportunidade de experimentar situações antes pouco prováveis de se encontrar nas salas de aula.

Apesar das tecnologias de educação apoiada por computador e educação à distância estarem caminhando a passos largos, algumas áreas do conhecimento ainda necessitam de atenção especial para alcançarem um grau de desenvolvimento que possibilite uma real mudança de paradigma. Isso pode ser verificado, por exemplo, em relação ao ensino de música. Mesmo com a existência de muitos *software* para treinamento de percepção e solfejo, para escrita musical e para seqüênciamento de sons, os recursos hoje existentes voltados para ensino de música tanto dentro da sala de aula, quanto à distância, ainda são muito precários, sendo que até a poucas décadas, a transmissão do conhecimento musical estava limitada à direta transferência da informação do professor para o aluno.

O caso do ensino de música, comparado a outras áreas da educação, está inserido num contexto particularmente diferenciado, uma vez que trabalha com alguns tipos de mídias muito específicas dentre as quais podemos citar os sons, os instrumentos musicais e as partituras.

O desenvolvimento de ferramentas para o ensino de música à distância, particularmente via Internet, se defronta com o problema da criação de meios de transferência de informação do professor para o aluno e vice-versa, sendo que tal informação ou conhecimento não se limita somente a textos explicativos sobre cada assunto relativo à música, mas também à transferência da informação gráfica das partituras e da informação auditiva dos sons. Este conjunto de informações, tão diferentes em sua natureza, faz com que o esforço de criação e desenvolvimento de tais aplicações ou ferramentas educacionais acabe por integrar diversas áreas do conhecimento, áreas estas que abrangem desde a pedagogia e didática musical até a computação gráfica, passando, obviamente, pelo domínio da escrita musical, da análise de sistemas e da engenharia de *software*, entre outras.

O presente trabalho vem apresentar uma proposta de utilização da Internet como uma ferramenta para o apoio à educação musical no âmbito de ensino superior. Propõe o uso da tecnologia XML associada ao Java para a criação de *applets* que, inseridos num *website*, possibilitam a visualização de partituras previamente preparadas pelo professor das disciplinas de música. Apresenta o

aplicativo JavaMusic, o *applet* JM e uma versão de codificação de partitura em XML, como protótipos desenvolvidos para esta pesquisa.

O objetivo deste trabalho é propor uma nova ferramenta de apoio ao ensino de música que permitam enriquecer os métodos tradicionais com recursos baseados nas novas tecnologias, buscando um aprimoramento e um crescimento da qualidade dos cursos de música hoje oferecidos pelas universidades, bem como a obtenção de mecanismos que futuramente podem servir ao desenvolvimento de cursos de música à distância que poderão atingir um público hoje excluído dos meios de ensino superior, seja devido ao custo do ensino, seja devido a uma indisponibilidade de tempo para freqüentar uma escola formal ou à impossibilidade de locomoção para um dos centros de ensino de música existentes.

O Capítulo 2 desta dissertação trata a questão do uso da informática na educação de um modo geral, comentando as experiências já existentes, as tecnologias que estão sendo desenvolvidas nesta área e os problemas mais comumente encontrados.

O Capítulo 3 traz a discussão para o âmbito do ensino de música tratando o problema da simulação da sala de aula de música num ambiente computacional, os *software* e *websites* de apoio à educação musical disponíveis no mercado, o desenvolvimento de *software* para educação musical e o uso de *applets* Java para a criação de *sites* de apoio à educação musical.

O Capítulo 4 propõe um modelo de documento XML para a codificação de partituras comparando-o com outros modelos já existentes.

O Capítulo 5 descreve o *software* JavaMusic, desenvolvido no decorrer deste trabalho, detalhando suas funcionalidades, todas as suas classes e seus métodos.

O Capítulo 6 propõe um cenário no qual o JavaMusic pode ser usado.

Segue, então, a conclusão onde são analisados resultados e problemas encontrados e são apresentados propostas para uma futura continuidade do trabalho.

Também são providos dois apêndices. O primeiro traz a DTD completa para os documentos XML gerados pelo JavaMusic e o segundo contém um exemplo de arquivo XML e a partitura a ele correspondente.

2 A informática na educação

O grande desenvolvimento ocorrido na informática e na ciência da computação verificado nas duas últimas décadas provocou uma verdadeira revolução na maneira como o homem se relaciona com todas as forma de conhecimento, no comércio, na economia e enfim em todos os campos da vida humana. O aperfeiçoamento dos computadores, a Internet, os grandes centros de processamento de dados, a informatização da indústria, dos serviços e da escola têm gerado novos paradigmas em todas as áreas de atuação do homem moderno.

Especificamente na área da educação novas portas têm se aberto graças ao advento da tecnologia. Não podemos deixar de refletir sobre o papel da informática na educação constatando dia a dia as inumeráveis vantagens de sua utilização. O uso de recursos tecnológicos, como o computador e a Internet, não só desperta nos alunos o interesse em estudar quanto os prepara para a integração com uma sociedade altamente tecnológica [FUE00]. A tecnologia cria novas relações culturais e desafia antigos e novos educadores, onde muitos apóiam as formas de educação indireta, ou seja, toda e qualquer atividade que desenvolva de forma implícita conceitos, princípios e teorias aprendidas nas aulas específicas das diversas disciplinas.

O uso do computador na educação começa concretizar um novo paradigma: estimular o aluno a construir o conhecimento. As relações entre professor e aluno são modificadas e as responsabilidades individuais aumentam. É preciso que tanto alunos como professores se envolvam tomando a iniciativa de participarem do processo de construção do saber.

Atividades grupais desenvolvidas através de ferramentas computacionais estimulam a participação de todos e geram mais autoconfiança em cada um [WOL98]. Ao introduzir a tecnologia em sala de aula, o professor perde o status de única fonte do conhecimento e passa a ser um colaborador no processo de descoberta e aquisição desse conhecimento, os alunos ganham maior atenção individual, colaboram mais uns com os outros e desenvolvem um espírito de equipe onde cada um tem algo a aprender ou a ensinar a seus colegas [FUE00].

Neste capítulo, serão abordadas as vantagens do uso do *software* educacional em sala de aula, o uso da Internet, o ensino à distância mediado por computador, o problema da interação do computador com o usuário e as soluções que têm sido encontradas na pesquisa de tecnologias para educação à distância.

2.1 Uso de *software* educacional

O uso de *software* educacional pode melhorar o aprendizado de cada aluno, pois através de simulações de situações reais, o aluno aprende a utilizar outras formas de inteligência que não simplesmente a inteligência lingüística e lógico-matemática [MAR99].

A rápida resposta dada pelas ferramentas computacionais encoraja os alunos a se autocorrigirem, os leva à experimentação e descoberta do conhecimento [FUE00] e dá a possibilidade de se promover uma imediata discussão dos erros e acertos de cada indivíduo e conduzi-los a uma melhor compreensão do assunto abordado.

A contribuição do *software* educacional no processo de ensino e aprendizado colabora no desenvolvimento de habilidades cognitivas de análise e compreensão dos problemas, além da habilidade de construção lógica de possíveis maneiras para a sua resolução [PIV99]. O computador pode ser usado no desenvolvimento de atividades complementares tanto para reforçar o conteúdo recebido em aula, quanto para eliminar dúvidas, e ainda para permitir atividades diferenciadas aos alunos mais adiantados ou para aqueles que apresentam mais dificuldades. Também pode ser usado antes da aula para introduzir os conceitos que serão estudados e propiciar uma experimentação mais efetiva desses conceitos antes de sua contextualização [FUE00].

De acordo com Martins et al [MAR99], pesquisas indicam que crianças gostam muito de estudar com computadores e que a auto-aprendizagem apresenta rápida evolução quando um *software* de

qualidade é introduzido em sala de aula. Alguns autores propõem o "*entertrainment*" - uma mistura de treinamento com entretenimento - para prender a atenção e estimular a imaginação dos estudantes.

Com o uso de *software* educacional, torna-se possível e repetição seguida de uma mesma tarefa e sua efetiva assimilação. O computador permite que as pessoas aprendam muito mais pelo fazer do que pelo ouvir. Schank defende que o processo de aprendizado pode ser baseado em objetivos a serem alcançados ao invés de lições consecutivas escravas do tempo [SCH98]. O autor preocupa-se com o desenvolvimento de métodos baseados no funcionamento da memória, afirmando que uma experiência será mais bem memorizada se houver uma conexão com experiências anteriores.

No entanto, o ingresso da informática nas salas de aula foi somente uma das grandes contribuições legadas pela tecnologia. Não podemos deixar de citar a grande contribuição que a Internet tem dado à área educacional, não somente por constituir-se fonte de conhecimento e pesquisa bibliográfica, como também por ter possibilitado um grande avanço no ensino à distância.

O desenvolvimento desses novos recursos tecnológicos para a educação atinge diretamente alunos e educadores sendo que, principalmente para os professores, o conhecimento destes recursos é cada dia mais imprescindível para uma boa atuação numa sociedade tão dominada pela tecnologia. Em relação ao uso da Internet, por exemplo, Piva Jr. afirma:

> "*... o professor que pretende se manter na profissão deve não só aprender como utilizar a Internet, mas, principalmente, saber como utilizá-la de forma pedagógica, envolvendo e guiando os alunos de forma efetiva e eficiente, priorizando a qualidade da informação*". [PIV99]

A velocidade com que se desenvolvem novas formas de interação com os usuários tem motivado educadores a repensar os métodos pedagógicos aproveitando a contribuição que a Internet tem trazido em termos de acesso à informação e comunicação entre pessoas e grupos.

2.2 Educação à distância

O desenvolvimento de ferramentas computacionais de comunicação através de redes e o advento da Internet, possibilitou a integração entre as mais remotas partes do mundo e conseqüente intercâmbio de culturas e conhecimentos. A Internet conseguiu agrupar num mesmo espaço informação, lazer e

serviços, facilitando a vida das pessoas, propiciando uma comunicação fácil, barata e eficiente e possibilitando que um indivíduo tenha acesso rapidamente a informações de qualquer área do conhecimento. Vislumbrando as inesgotáveis possibilidades oferecidas hoje pela Internet em termos de difusão de conhecimento e serviços, tem sido despendido grande esforço no sentido de se melhorar suas condições de acessibilidade, interação com o usuário e oferta de aplicações. Entre as aplicações que mais são pesquisadas, podemos destacar o comércio eletrônico, transmissão de dados em múltiplas formas de mídia, comunicação escrita, falada e visual e educação à distância.

Com o advento da Internet surgiram muitos cursos à distância baseados nessa tecnologia, sendo que muitos deles, promovidos por universidades, abrangem desde cursos de extensão e graduação até cursos de pós-graduação. Muitas empresas, também, têm usado a Internet como importante ferramenta de treinamento dos seus funcionários.

A pesquisa em educação à distância mediada por computador tem caminhado a passos largos. Os avanços propostos para a Internet sobre a utilização das linguagens de marcação de hipertexto como o HTML (*HiperText Markup Language*)[3] e das linguagens extensíveis como o SGML (*Standand Generalised Markup Language*)[4] e o XML (*eXtensible Markup Language*)[5] tem contribuído para torná-la cada vez mais eficiente na comunicação e na apresentação da informação. A possibilidade de distribuir numa mesma ferramenta (a Internet) mídias diversas como texto, imagem, som, animação, vídeo, bem como a possibilidade de comunicação entre pontos remotos através de correio eletrônico, comunicação escrita instantânea via bate-papo e a teleconferência, têm possibilitado a criação de ferramentas para educação à distância chamadas de Ambientes de Ensino a Distância, onde o professor pode desenvolver e disponibilizar todo o material didático para a Internet além de gerenciar os cursos e o desempenho dos seus alunos, mesmo tendo muito pouco conhecimento de informática.

O surgimento de cursos à distância via Internet permite que pessoas impossibilitadas de se locomover até uma determinada escola ou com limitações de tempo possam estudar nos horários e lugares que lhes convêm. Graças à grande eficiência na comunicação via Internet é possível a realização de trabalhos colaborativos entre alunos separados pelo tempo e pelo espaço sem prejuízo da qualidade educacional.

---

[3] http://www.w3.org/MarkUp
[4] http://www.w3.org/MarkUp/SGML
[5] http://www.w3.org/XML

2.2.1 Interação

Apesar do grande desenvolvimento dos recursos direcionados a aplicações na Internet, ainda há muito que se pesquisar para se melhorar a qualidade das interações para oferecimento de recursos de comunicação e colaboração realmente de qualidade [RAP99].

Mesmo com os recursos já existentes, ainda encontramos algumas limitações no campo da interação. Em [RAP99] são citados:

- A falta de controle sobre a aparência das interfaces de acesso à Internet, uma vez que esse controle cabe aos *browsers* e estes por sua vez dependem de sua implementação interna;
- dificuldade de interfaces independentes de plataforma, o que faz com que o mesmo documento da *web* tenha uma aparência diferenciada para cada sistema operacional;
- As diferenças no tipo de conexão para a Internet e desempenho dos computadores, o que pode tornar a execução de determinadas interfaces impraticável em conexões muito lentas ou em computadores de baixo desempenho; o fato do protocolo HTTP não garantir uma taxa de transmissão mínima para aplicações contínuas como áudio e vídeo provocando interrupções na transmissão.

Abaixo, segue uma breve descrição de alguns dos recursos atualmente usados na interação com o usuário na *web*, suas vantagens e desvantagens.

HTML

O HTML (*HiperText Markup Language*) é o padrão para a apresentação de hipertexto na *web*. Formado por um conjunto de marcas (*tags*), possui recursos para estruturação e formatação do texto, inclusão de imagens e multimídia, criação de tabelas, divisão em quadros e definição para cores ou imagem de fundo sendo que todos esses elementos citados são totalmente estáticos.

Os elementos de interação com o usuário, definidos no padrão HTML, são as âncoras para a interligação de documentos, o mapeamento de imagens e um conjunto de *widgets* (elementos da interface) que abrange botões, *checkboxes*, listas de seleção, caixas de entrada de texto. Esses *widgets* são usados na criação de formulários. Porém, apesar de todos esses elementos, ainda não se consegue um grau de interação muito grande com o usuário, pois uma vez carregada uma página HTML, o usuário não pode interferir no visual da página ou fazer alguma personalização, pois todos os elementos são dispostos automaticamente pelo *browser*.

O usuário também não pode carregar uma parte da página: atualizações envolvem o carregamento completo da página e o servidor não tem acesso às ações intermediárias dos usuários como, por exemplo, o preenchimento de um campo de texto, pois todos os dados que são enviados ao servidor são remetidos de uma só vez ao final da operação.

Outra questão importante refere-se à memória de *cache* do *browser*: todas as páginas solicitadas pelo usuário ficam armazenadas num *cache* para acelerar o carregamento de páginas que já tenham sido visitadas. Isso significa que uma possível alteração na página original pode não ser vista pelo usuário.

As soluções encontradas para resolver essas questões passam por dois caminhos: o primeiro, a geração de páginas HTML de forma dinâmica pelo servidor, o que ainda exige a realimentação das páginas na máquina do usuário (cliente), o segundo, é legar ao cliente o processamento de parte dos dados em sua própria máquina, através de programas que possam interagir com o usuário e ou com o servidor.

Processamento no servidor

Atualmente existem muitas soluções que estão sendo usadas para gerar páginas HTML de forma dinâmica através de programas que rodam no servidor. Abaixo segue uma breve descrição de algumas das soluções mais usadas:

CGI

Os CGIs (*Common Gateway Interface*) propiciam a criação dinâmica de páginas HTML, permitindo sua adequação ao ambiente receptor (cliente) e um certo nível de personalização de acordo com a necessidade do cliente. Os programas CGI podem ser construídos em qualquer linguagem que tenha acesso às variáveis de ambiente e assim podem ler bancos de dados, processar informação e gerar as respostas às solicitações dos usuários.

Com o uso de CGI, um *site* educacional pode, por exemplo, disponibilizar um exercício a ser realizado pelo aluno e, depois de concluído, as respostas do aluno podem ser enviadas ao servidor onde o programa CGI faz a correção e envia, ao aluno, uma nova página com a correção do exercício. No entanto, cada vez que o usuário solicita uma informação do servidor ou um exercício, no caso de um *site* educacional, é aberta uma nova conexão na rede para solicitar ao servidor e outra para receber a resposta já processada. Considerando o tempo de processamento do CGI, se o usuário estiver conectado à Internet através de uma rede muito lenta, como as redes *Dial-Up*, o processo todo poder se tornar muito demorado.

Servlet

Seguindo o mesmo princípio dos CGI, encontramos os *servlets* que consistem de aplicações do servidor escritas em Java e que carregam certas vantagens dessa linguagem como a independência de plataforma e a segurança.

Os *servlets* têm basicamente a mesma função dos CGIs, mas encontraram um destaque em aplicações combinadas entre *servlets* no servidor e *applets* no cliente. Trataremos os *applets* logo a seguir.

PHP

O PHP (*Personal Home Page*) é um recurso de processamento no servidor baseado em *scripts* que rodam sobre um interpretador PHP.

A implementação e a manutenção de *sites* que utilizam PHP é bastante simples, tornando esta tecnologia cada vez mais utilizada [CHO01]. Ela permite a criação dinâmica de páginas HTML, processamento de dados enviados pelo cliente e acesso a bancos de dados.

Processamento no cliente

Para minimizar as transferências de dados pela rede quando da solicitação de informações pelo usuário, uma das alternativas encontradas foi usar a máquina cliente em parte do processamento. Com isso, tornou-se possível, por exemplo, validar o preenchimento de um formulário antes do seu envio para o servidor, evitando que um formulário preenchido erroneamente tenha que ser retornado ao usuário para efetuar as correções. As técnicas mais usualmente encontradas na *web* são descritas abaixo:

Applet

Os *applets* são programas construídos também em linguagem Java que rodam dentro do *browser* do usuário. Esses programas usam uma área da janela do *browser* cujo tamanho é definido pelo código HTML e podem conter figuras, imagens em movimento, formulários, sons, enfim, uma infinidade de informações multimídia, que dão à página HTML uma grande possibilidade de recursos.

Por terem todo o processamento realizado na máquina cliente, os *applets* minimizam o uso da conexão e, como a linguagem Java é uma linguagem de programação muito completa e orientada a objetos, sua capacidade de processar dados é muito grande e sua capacidade de interagir com o usuário tem poucas limitações.

Os *applets* têm algumas restrições de segurança que não os permite acessar diretamente a máquina cliente, não podendo, por exemplo, gravar ou ler arquivos locais, ler as variáveis locais ou se

conectar a outra máquina que não seja o próprio servidor. No entanto, seu uso associado com *servlets* permite a comunicação síncrona entre dois usuários.

A maior limitação apresentada pelos *applets* é que estes podem ser programas muito extensos e levar um tempo razoável para serem transmitidos do servidor para o cliente. Além disso, podem funcionar de forma muito lenta se a máquina cliente não for um computador muito eficiente.

Apesar disso, os *applets* têm sido usados na criação de laboratórios virtuais, permitindo a realização de experimentos didáticos via Internet[6], entre outras aplicações.

JavaScript

Mais simples de ser implementado que os *applets*, o JavaScript é uma linguagem de *script* que permite gerar algum processamento de dados na máquina do cliente podendo ser usado para validar informações de preenchimento de formulários, corrigir exercícios, gerar páginas com textos em movimento, pequenas animações e até alguns jogos.

O uso do JavaScript melhorou a interação com o usuário e reduziu a necessidade de comunicação com o servidor para processar dados enviados pelo usuário. O código ou *script* é escrito dentro do próprio código HTML e também pode ser usado em combinação com CGIs ou *Servlets*.

VRML

Outra linguagem que merece ser comentada é o VRML (*Virtual Reality Modeling Language*)[7]. Ela permite a criação de ambientes de Realidade Virtual na máquina cliente. Estes ambientes, que podem ser ricos em interação e em recursos multimídia, têm todo o processamento feito na máquina cliente e por isso necessitam de pouco recurso em termos de conexão com a Internet, mas têm um alto custo computacional despendendo grandes recursos de processamento, o que torna a execução muito lenta em computadores de baixo desempenho. Além disso, não possui uma interface amigável para a criação dos ambientes de Realidade virtual obrigando sua implementação diretamente sobre o código da linguagem.

Flash

O uso de *Flash*[8] nas páginas da Internet também merece grande destaque, uma vez que hoje podemos encontrar muitas páginas totalmente baseadas nesta tecnologia.

---

[6] http://physics*web*.org/vlab/
[7] http://www.vrml.org
[8] http://www.macromedia.com

O *Flash* permite grande interação com o usuário, processamento de informações, portabilidade, segurança, largo uso em multimídia, mas apresenta como grande desvantagem o fato de ser uma tecnologia proprietária e relativamente cara.

É ideal para a criação de animações para demonstração de temas em física, química e mecânica, por exemplo. Ou seja, apresenta uma larga gama de possibilidades de emprego didático, com fácil implementação, mesmo por pessoas que tenham pouco conhecimento na área de informática e computação.

2.3 Considerações finais

Não podemos negar o fato de que a tecnologia aplicada à educação tem gerado uma verdadeira revolução nos conceitos de ensino, de educação e de treinamento. O desenvolvimento de ferramentas de educação à distância baseadas na *web* abre um grande leque de possibilidades e democratiza o ensino, permitindo que pessoas separadas fisicamente da escola possam ter acesso a um ensino barato e de qualidade.

A possibilidade de comunicação escrita, falada e visual através da Internet permite que pessoas com pouca flexibilidade de possam realizar cursos à distância, permite que empresas possam treinar seus funcionários com um custo muito baixo e sem a necessidade de deslocamento.

As novas ferramentas que estão sendo desenvolvidas facilitam não somente a comunicação e a troca de conhecimentos, mas também, a colaboração entre os participantes, a realização de trabalhos em grupo, mesmo que os personagens estejam distantes uns dos outros e sem a necessidade de sincronismo de horários, mas também permitem a sincronização se isso for realmente necessário ao processo de aprendizado e à colaboração entre participantes.

No entanto, o uso de ferramentas computacionais também tem algumas desvantagens, ou, pelo menos, riscos: é preciso cuidar para que o computador não perca seu papel de ferramenta de trabalho, tornando-se o objeto primário das discussões. É importante salientar que a máquina deve permanecer com o status de meio e nunca se tornar um fim. Não basta a simples adequação das tecnologias. É preciso reprojetá-las frente aos novos paradigmas da educação [SAN99], é preciso aperfeiçoar os *software* educacionais e os ambientes de ensino a distância, desenvolver novas formas de interação homem-computador, melhorar as interfaces tornando-as cada vez mais intuitivas e sempre ter em vista que, para o desenvolvimento de um bom curso, seja ele presencial ou à distância, é esperado que o

professor tenha domínio de sua área de conhecimento e suposto que, não necessariamente, tenha um conhecimento profundo sobre Internet ou outras ferramentas computacionais [LUC99].

3 Educação musical e a informática

Observando a história da humanidade, podemos perceber que a música manteve-se fortemente presente em todas as civilizações.

Apesar de cada povo ter desenvolvido uma cultura musical própria, a transferência desse conhecimento através das gerações seguiu por um mesmo caminho: a transferência oral.

Dando um salto na história e chegando aos dias de hoje, continuamos a encontrar um cenário bastante diversificado no que se refere à cultura musical dos povos atuais. Porém, a transmissão dessa cultura já não está mais limitada à tradição oral podendo ser encontrada uma grande diversidade de formas de transferência do conhecimento musical.

Falando-se especificamente da música ocidental, o aparecimento da escrita musical possibilitou a conservação desse conhecimento além dos limites da tradição oral. Hoje, é possível estudar a música que era tocada desde a Renascença graças aos registros escritos e, com o advento da gravação sonora no final do século XIX, um passo ainda maior foi dado: a possibilidade da preservação e do estudo da interpretação de uma música congelada pelo gramofone, pelo vinil e nos dias atuais pelo *Compact Disc* e pelo DVD.

Outro recurso que, pouco a pouco, começa a ser usado no estudo e no ensino da música é o computador. Porém, antes que o homem chegasse à música computacional e a recursos como o MIDI [MID][9] e os sintetizadores, um longo caminho foi percorrido.

As primeiras pesquisas com música eletrônica começaram a partir de Leon Theremin que, em 1924, desenvolveu um equipamento que gerava sons a partir da modulação de ondas de rádio [MOO96]. Esse equipamento foi usado por compositores como Oliver Messien (Ondes Martenot) e Paul Hindemith (Trautonium).

Anos mais tarde, na década de trinta daquele século, Leopold Stokowski[10] disse:

> *"Vejo um tempo em que os compositores poderão criar diretamente com os sons e não com papel".*

Naquele momento, já era vislumbrado que as fronteiras dos instrumentos musicais seriam expandidas pela existência de tecnologias capazes de modelar os sons, armazená-los e manipulá-los como hoje é feito através do uso de computadores.

A música computacional surgiu na década de 1950 com os trabalhos do químico e compositor Lejaren A. Hiller que usava modelagem estocástica no processo de composição. Sua ferramenta de trabalho foi o computador ILLIAC da Universidade de Illinois.

Em 1957, Max Matheus, pela primeira vez usou o computador como uma espécie de instrumento musical desenvolvendo o primeiro programa para sintetizar música.

Nos anos seguintes, muitos avanços foram obtidos com o desenvolvimento dos circuitos integrados, do microprocessador e dos sintetizadores e mais tarde na década de 1980, o aparecimento do MIDI (*Musical Instrument Digital Interface*), um padrão de interligação entre instrumentos musicais eletrônicos, possibilitou a completa integração entre a música e o computador.

---

[9] MIDI *Musical Instruments Digital Interface*. Os arquivos MIDI, armazenam seqüências de eventos MIDI que podem ser enviados a uma placa de som ou a um instrumento musical eletrônico conectado ao computador. Vide [MID].

[10] Leopold Stokowski (1882-1977), nascido em Londres, é considerado um dos maiores regentes que o mundo já conheceu. Esteve à frente da Orquestra Filarmônica de Chicago por 24 anos consecutivos, responsável pela orquestração de diversas obras incluindo compositores como Bach e Mussorgsky, entre outros. Participou de diversos filmes, entre eles Fantasia de W. Disney.

Neste capítulo será abordado como a informática pode contribuir com o processo de educação musical, principalmente na formação superior em música, voltada à preparação de profissionais de música. Será discutido como as ferramentas usadas na sala de aula de música podem ser complementadas por recursos computacionais num cenário onde o professor de música usará a Internet como extensão da sala de aula. Veremos ainda alguns exemplos de *software* desenvolvidos especificamente para a educação musical e adaptações que podem ser feitas em *software* desenvolvido inicialmente para outras finalidades. Será tratada ainda a questão da representação computacional da música e as dificuldades encontradas no desenvolvimento de *software* para educação musical.

3.1 A aula de música

O ensino tradicional de música[11] ocorre, em geral, em dois cenários distintos: o primeiro, é a aula teórica, na qual um grupo de alunos, confinados em sala de aula, recebe conteúdo expositivo de assuntos teóricos ligados à música, tais como História da Música, Harmonia, Análise Musical, Contraponto, entre outras; o segundo é a aula prática, sendo que esta pode ser individual ou em grupo. As aulas individuais são, geralmente, as aulas de instrumento ou de técnica vocal, enquanto as aulas em grupo são Coral, Arranjo, Composição, Regência, Música de Câmara, entre outras.

Tanto as matérias teóricas como as práticas utilizam-se de um conjunto de ferramentas através das quais as informações sobre a ciência da música são transferidas e/ou manipuladas. Entender-se-á por ferramenta todo e qualquer meio ou material usado em sala de aula, por professores e alunos, seja para o registro do conteúdo teórico, seja para a prática da música ou para seu registro sonoro. Neste sentido, também será usado o termo **mídia**, ou seja, meio ou veículo usado em sala de aula como ferramenta de ensino e produção musical.

Assim, as ferramentas ou mídias usadas no ensino destas matérias podem ser: quadro-negro, caderno pautado, partitura, livros, métodos[12], piano e instrumentos musicais variados, gravações, vídeos, etc.

---

[11] Ao usar a expressão "ensino tradicional de música", estou me referindo ao ensino de música aplicado pelas escolas de música, tais como, conservatórios e escolas de música, faculdades de música e as disciplinas de artes dos cursos de pré-escola e básico nos quais algumas escolas ainda mantêm o ensino de música.

[12] Em educação musical, o termo **método** não se refere exclusivamente a um tipo específico de pedagogia de ensino, mas costuma ser empregado para designar um livro ou conjunto de livros contendo exercícios voltados ao desenvolvimento de uma determinada habilidade musical segundo determinada pedagogia. Assim podemos encontrar, por exemplo, métodos de piano, para o estudo de piano, métodos de solfejo, para estudo de leitura e solfejo, métodos de canto, com exercícios de canto etc.

A utilização destas ferramentas em sala de aula pode variar de acordo com o conteúdo de cada matéria, podendo uma determinada matéria usar diversas ferramentas enquanto outra se apega ao uso de uma única mídia.

É importante relacionar as ferramentas usadas pelas matérias de um curso de música, pois discutiremos como essas ferramentas poderiam ser complementadas ou até mesmo substituídas pelo uso do computador em sala de aula ou no ensino à distância.

Hoje em dia, podemos encontrar três situações em sala de aula:

Adoção de métodos:

O professor escolhe e adota um método entre os muitos encontrados no mercado editorial. Todos os alunos são obrigados a comprar o livro adotado (embora seja uma prática muito comum a cópia pirata dos livros, tendo em vista a dificuldade de encontrá-los nas livrarias ou mesmo o alto custo de sua aquisição). Apesar de muitos destes métodos terem seu uso consagrado já há décadas, é comum que apresentem revisões e alterações, forçando tanto alunos quanto professores a adquirirem novas edições se realmente desejarem manter-se atualizados quantos as modificações.

Uso de apostilas:

Nesta situação, o professor desenvolve um método próprio, escreve uma apostila e distribui cópias xerocadas da mesma. Trata-se de uma prática bastante comum, pois facilita a atualização e revisão do material. Muitas vezes, as apostilas são distribuídas página a página de acordo com a necessidade das aulas, porém essa prática trás o risco de extravio de parte do material. A revisão do material, por sua vez, implica em novas cópias bem como na redistribuição das mesmas.

Uso do quadro negro:

Alguns professores preferem transcrever o material didático para o quadro negro, obrigando seus alunos a efetuarem a cópia de próprio punho. Apesar disto resolver o problema da revisão e distribuição do material, implica num grande desperdício de tempo de aula e um constante re-trabalho tendo em vista que, a repetição de uma mesma aula, obriga o professor a reescrever no quadro-negro todo o seu conteúdo.

Estas três situações podem ser encontradas em praticamente qualquer disciplina de um curso de música. No entanto, poderiam ser amenizadas com a introdução de um computador na sala de aula. O

uso de um projetor (*DataShow*) viria a substituir o quadro-negro, possibilitando melhorar a dinâmica das aulas. O material didático poderia ser alocado num *website* criado para cada matéria, fazendo da atualização e distribuição do material um processo dinâmico e contínuo.

Outro problema comum numa sala de aula de música, principalmente nas disciplinas de percepção, análise, harmonia e contraponto, é a necessidade do uso de exemplos sonoros (musicais). Neste caso, se o professor for um instrumentista (especialmente um pianista), o problema é resolvido com um piano ou teclado em sala de aula (ou outro instrumento de domínio do professor). Porém, há situações em que ou o professor não é instrumentista ou a sala não dispõe de um instrumento musical. Aqui, novamente, o computador pode tornar-se um grande aliado, permitindo que o professor prepare os exemplos musicais em arquivos .mid que, além de serem utilizados durante a aula, também podem ser disponibilizados aos alunos através da Internet.

O uso de ferramentas computacionais no apoio à educação é cada vez mais discutido e incentivado por diversos pesquisadores e educadores. Em [UNE95] encontramos uma defesa da inclusão da tecnologia MIDI entre as ferramentas usadas nos cursos de música, propondo seu uso para a gravação e edição digital de acompanhamento musical para as aulas de instrumento e música de câmara, o uso em aulas de composição para permitir ao aluno a audição de seus trabalhos sem a necessidade de qualquer executante e a editoração de partituras.

3.2 Simulando as mídias usadas no ensino de música no computador

O ensino de música requer o uso de mídias muito específicas as quais nem sempre podem ser satisfatoriamente substituídas ou simuladas por meio de computadores. Essas mídias se prestam à comunicação visual (livros, partituras, quadros-negros) e gestual (regência), à comunicação sonora (instrumentos musicais, canto, gravações) e ao registro gráfico, textual e sonoro (caderno pautado e gravador de cassete).

O maior problema na simulação das mídias usadas na sala de aula de música ocorre quando estudamos a adequação dessas mídias para um formato que possa ser distribuído via *web*.

Mesmo considerando a existência de ferramentas que possibilitam a comunicação sonora e a teleconferência via Internet, esses mecanismos ainda sofrem com os atrasos provocados pela falta de banda nas redes IP, o que os torna pouco eficientes com relação à interação síncrona entre seus usuários. No entanto, em relações não síncronas, elementos como os livros podem perfeitamente ser reproduzidos

em páginas HTML, uma comunicação escrita entre alunos e professores pode ser feita através de bate-papo eletrônico num modo síncrono e de correio eletrônico e listas de discussão em modo assíncrono.

Gravações de música e arquivos MIDI podem ser disponibilizados para *download*, e mesmo gravações dos alunos cantando e/ou tocando podem ser trocadas via Internet entre alunos e professores.

A linguagem HTML permite a adequação de uma grande variedade de recursos multimídia como Flash, *applets* Java, GIFs animados, imagens diversas, vídeos MPEG, AVI, jogos eletrônicos, sons e *hiperlinks*, trazendo uma grande gama de possibilidades para a transferência do conteúdo teórico das salas de aula para a *web*.

A maior perda que se tem na utilização da *web* está na questão da interação e do contato visual entre aluno e professor: na sala de aula, o professor pode observar e interagir diretamente com o aluno, analisando sua postura diante de um instrumento musical, orientando sua execução, tirando dúvidas nas aulas teóricas e fazendo-se de modelo para seus alunos.

Tampouco os instrumentos musicais podem ser perfeitamente simulados, pois a audição da gravação de um instrumento não tem o mesmo efeito didático que tem em sua audição ao vivo e, tratando-se de uma gravação, perde-se qualquer possibilidade de interação entre os personagens atuantes.

No caso específico das partituras, estas sim podem ser perfeitamente simuladas no computador, inclusive em ambientes como a Internet. Por se tratar de simples representação gráfica da música, podem ser digitalizadas e inseridas em páginas HTML em formato de imagens GIF ou JPEG, possibilitando a criação de métodos de estudo para instrumentistas e aulas de teoria musical, harmonia, contraponto e enfim matérias teóricas em geral.

Uma possível utilização destes recursos está em otimizar o estudo individual, acrescentando recursos motivadores como interação com o usuário, resposta imediata a exercícios e, enfim, criar uma nova roupagem para os métodos tradicionais de ensino de teoria musical, fazendo da Internet, uma extensão da sala de aula e do computador um aliado ao ensino.

Apesar de tudo citado anteriormente, a preparação de material para *web* não é uma tarefa trivial, exigindo do professor alguns conhecimentos de HTML, de conversão de formatos de arquivos de imagem, de recursos multimídia e a posse de equipamentos de digitalização, câmeras de vídeo, equipamentos de gravação, etc.

Se considerarmos que a partitura é a mídia mais usada na sala de aula de música, todo o problema será amenizado, quando tivermos uma ferramenta computacional que permita a geração de partituras e sua direta adequação à *web*, sem a necessidade de geração de arquivos de imagens. Com

esse recurso, o professor poderá usar a Internet para distribuir conhecimentos teóricos de música, mesmo tendo poucos recursos em termos de *hardware* e de conhecimento de informática.

No entanto, para que tal ferramenta computacional seja viável, seu projeto deve levar em conta que as interfaces de usuários precisam ser intuitivas no que refere à sua utilização e leves no que se refere à execução, partindo-se do princípio que o computador nem sempre é uma ferramenta amigável do ponto de vista do aluno ou do professor, que eles não necessariamente dominam a ferramenta computacional o bastante para resolver problemas relativos à eficiência da máquina, configuração de parâmetros, instalação de *software*, etc. Mais adiante será demonstrado um *software* desenvolvido como uma proposta de solução a essa questão.

Por hora, podemos afirmar que o conteúdo potencialmente aplicável via *web* pode ser definido entre as matérias de ordem teórica, uma vez que, neste caso elas minimizam a necessidade da presença do professor durante o estudo, bem como matérias ligadas ao estudo prático, como percepção e prosódia musical, e que requerem dos alunos grande dedicação em termos de estudo individual.

3.3 *Software* de apoio à educação musical

A maioria dos programas para educação musical encontrados na Internet consiste de *software* com exercícios de *ear training* (treinamento auditivo), leitura e ritmo e não necessariamente aulas de música.

Na realidade, existem páginas na Internet com aulas de música sobre temas referentes ao programa básico (escalas, acordes, arpejos, harmonia, notação musical e história), porém todo esse conteúdo é apresentado num formato de livro eletrônico em HTML e, assim, assume um caráter estático e não extensível.

Não foi encontrada nenhuma página que tivesse um caráter dinâmico no que se refere à preparação das aulas, salientando que o termo "página estática ou fixa" aqui se refere àquela página ou *site* construído em forma de um livro em HTML e que não pretenda, ou ao menos não pareça, pertencer a uma estrutura maleável e o termo "página dinâmica" refere-se àquela preparada ou construída de acordo com a demanda de um curso formal pré-existente à página HTML e teria por função principal atuar como um estudo complementar ao curso.

3.3.1 *Software* para *Ear training* e leitura musical

Os aplicativos para *ear training* são programas destinados a desenvolver as habilidades auditivas do aluno de música. Em geral, apresentam exercícios divididos em modalidades tais como:

- intervalos melódicos: o computador toca duas notas em seqüência e o aluno deve indicar a distância intervalar entre as notas;
- intervalos harmônicos: semelhantes aos intervalos melódicos, porém as notas são tocadas simultaneamente;
- acordes: o computador toca um acorde e o aluno deve dizer que notas foram tocadas e/ou classificar o acorde e/ou indicar o nome do acorde. Pode, ainda, ser apresentado um acorde arpejado (uma nota de cada vez) ou harmônico (todas as notas de uma só vez);
- ritmos: o computador toca um ritmo e o aluno deve reproduzi-lo ou escrevê-lo;
- melódicos: é tocada uma linha melódica e o aluno deve escrevê-la. Esse tipo de exercício também é conhecido por "ditado" e pode combinar notas e ritmos diferentes.

Os exercícios de *ear training* são programas em sua maioria proprietários com um custo em torno de 30 a 80 dólares e podem ser comprados via *web*. Apresentam diversos tipos de exercícios de treinamento auditivo capazes de desenvolver a percepção de intervalos harmônicos e melódicos, acordes diversos, escalas e ritmos.

As interfaces do usuário geralmente apresentam um teclado como o de um piano ou um braço de violão, onde o aluno indica quais as notas que foi capaz de ouvir e identificar dentro da estrutura musical proposta. No entanto, alguns *software*, como por exemplo o *Ear Power* da *Good Ear*[13], oferecem em sua interface a opção de usar o microfone como entrada de dados, onde o aluno pode cantar as notas ao invés de clicar com o *mouse* num piano virtual, sendo que este recurso de voz pode tanto ser usado nos exercícios melódicos e harmônicos quanto nos exercícios rítmicos.

---

[13] Good Ear - On-line Ear Training Site, disponível em http://www.good-ear.com, oferece recursos para o treinamento rítmico através da repetição de padrões gerados aleatoriamente pelo programa. O aluno deve repetir o padrão escutado, cantando ou batendo no teclado e o programa verifica a precisão rítmica. Exercícios de intervalos propõem notas de partida para intervalos propostos pelo programa e que devem ser reproduzidas num teclado virtual ou cantados pelo aluno. Para exercício de acordes, o programa toca um acorde e o aluno deve identificar cada nota do acorde, cantando ou indicando no teclado virtual.

Os exercícios de leitura, em geral, também utilizam um piano virtual em sua interface, onde o aluno indica as notas que lê numa partitura gerada aleatoriamente pelo programa, mas também existem *software* capazes de reconhecer a entonação do aluno captando sua voz através do microfone, como por exemplo, o *Sight Singer Trainer*[14].

Também podem ser encontrados na Internet *sites* com exercícios *on-line* de treinamento de leitura, acordes, escalas e intervalos, entre outros. Estes *sites* usam *applets* Java, ASP, CGI e PHP em sua estrutura. A grande vantagem em sua utilização é o fato de fornecerem exercícios gratuitos, com os quais o aluno pode treinar suas habilidades auditivas, sem a necessidade de compra e instalação de qualquer *software*. Sua principal desvantagem está no tempo de espera para o carregamento de cada página. Alguns exemplos podem ser vistos nos *sites*: *Ear Training WebSite*[15] e Java *Music Theory*[16].

3.3.2 *Sites* de teoria musical

São páginas da Internet contendo informações sobre teoria musical como: notação musical, harmonia, escalas, história da música, etc.

Alguns *sites* de conteúdo teórico (aulas) visitados durante esta pesquisa apresentam uma estrutura muito parecida com a de um livro e, como mencionado anteriormente, têm um caráter bastante estático. Em geral, não são associados a um curso específico de música, sendo mais voltados para estudantes autodidatas. Sua interface utiliza imagens de partituras digitalizadas (.gif ou .jpg) e a audição dos exemplos musicais é possível via arquivos .mid e .wav.

Além dos *sites* de teoria musical, também foram localizados dezenas de *sites* sobre história da música. Dois exemplos de *sites* de teoria musical são: *Teoría*[17], *MusicArrangers.com*[18] e exemplos de *sites* sobre história da música: *Carolina Classical Connection*[19] e *Historical Women Composers*[20].

---

[14] O *software* Sight-Singing Trainer é uma ferramenta de estudo de leitura de intervalos melódicos onde é apresentada uma partitura que o aluno deve entoar corretamente. O programa indica a nota que o aluno está cantando e marca as notas corretas. Pode ser encontrado para compra em http://www.good-ear.com.

[15] Ear Training *Web*Site - disponível em http://www.earpower.com, contém um acervo de exercícios teóricos de diversos gêneros.

[16] Java Music Theory - disponível em http://*web*1.hamilton.edu/javamusic, contém exercícios construídos em Java podendo ser acessados diretamente na página que utiliza *applet* Java ou também na versão para *download*.

[17] Teoría - Music Theory - Teoría de la música, disponível em http://www.teoria.com, trata-se de um *website* sobre teoria musical com livros HTML, artigos e exercícios e apresenta uma versão em inglês e outra em espanhol.

[18] MusicArrangers.com Lessons from beginners to the very advanced - disponível em http://music-arrangers.com/star-theory consiste de um curso de teoria musical *on-line*.

3.3.3 *Software* não-educacionais

Além dos *software* desenvolvidos especificamente para o ensino de música, muitos *software* de apoio ao trabalho do músico podem ser encontrados no mercado. São programas para a edição de partituras, seqüenciadores[21] de eventos MIDI, editores e processadores de sons, *mixers*, conversores de áudio para .wav e .mp3, enfim uma imensa gama de produtos que têm auxiliado muito as tarefas do músico ligadas à preparação das partituras, gravação e mixagem, programação de teclados MIDI, etc.

Apesar destes *software* comerciais não terem inicialmente uma função didática, segundo Fritsch et al [FRI99]:

> "*Todo software pode ser considerado educacional se seu uso estiver inserido num contexto pedagógico onde exista uma metodologia direcionando todo o processo.*"

Isto significa dizer que um professor pode se utilizar destes programas de forma didática para criar, por exemplo, exercícios a serem aplicados aos alunos em sala de aula, além dos exemplos de programas educacionais já citados e dos exercícios tradicionais como a prova escrita ou a chamada oral.

3.3.4 Desenvolvimento de *software* educacional

O desenvolvimento de *software* para educação musical é considerado por Fritsch et al. uma atividade de caráter interdisciplinar, uma vez que envolve diversas áreas do conhecimento como educação musical, psicologia, música computacional, interface homem-computador, análise de sistemas, projeto de *software* e programação. A autora ressalta sua preocupação com os programas educacionais desenvolvidos no Brasil que, em geral, não são baseados em estudos recentes de cognição, sendo, ao contrário, baseados em métodos tradicionais de apresentação, aplicação de conceitos e avaliação. Afirma

---

[19] Carolina Classical Connection - em http://classicalmus.hispeed.com/links.html, trata a história da música dividida entre os grandes períodos históricos.

[20] Historical Women Composers - em http://music.acu.edu/www/iawm/historical/historical.html, aborda a história de mulheres compositoras desde a renascença até os dias atuais.

[21] Seqüenciadores são programas capazes de gravar uma seqüência de eventos MIDI que poderão dar origem a um arquivo MIDI ou ser enviada a um instrumento musical ou à placa de som conectada ao computador.

também que um grande número de *software* educacionais não promovem o direto envolvimento do aluno com a música, apegando-se a tópicos isolados [FRI99].

No entanto, há um exemplo de *software* desenvolvido inicialmente para composição e que está sendo adaptado para ser aplicado em educação musical e consciência sonora para crianças. Trata-se do Rabisco [MANZ], um *software* de composição baseado em linhas traçadas na tela do computador e que são associadas às notas musicais. Inicialmente escrito em *Visual Basic*, foi transformado em *applet* Java e está sendo adaptado pelo projeto PGL[22] para ser aplicado no ensino de música para crianças de 5ª a 8ª séries do ensino fundamental (10 a 14 anos). A idéia é usar a capacidade de composição através de uma interface visual para estimular a criança a reconhecer os sons e suas características de uma forma lúdica e criativa, rompendo com as formas mais tradicionais de musicalização infantil.

Uma das maiores dificuldades no projeto de programas relacionados com música e, especialmente, com escrita musical está na separação que existe entre a escrita e o som.

O programador deve ter consciência de que a escrita em si não define o resultado sonoro, sendo apenas sua representação gráfica e para a qual é preciso estabelecer uma relação de valores entre cada símbolo escrito e sua possível conversão em som, conversão esta, que no cerne do computador, não se processa diretamente, pois é preciso primeiramente gerar um evento MIDI para que este, então, gere o som através de uma placa seqüenciadora.

Figura 1: Notação musical convertida em eventos MIDI[23]

Outra questão que dificulta o desenvolvimento de *software* musical está no tratamento dos eventos gerados pelo usuário. Quando o usuário vê a partitura desenhada na tela do computador e deseja inserir, por exemplo, uma nova nota musical, ao clicar com o *mouse* na posição desejada, o programa deve estabelecer a relação entre a posição do ponteiro do *mouse* sobre a imagem e seu significado

[22] O PGL, The Partnership in Global Learning, é um projeto desenvolvido em parceria entre Universidade da Flórida (EUA), Fundação Getúlio Vargas, Universidade Estadual de Campinas, Pontifícia Universidade Católica do Rio de Janeiro e Instituto Tecnológico de Estudos Superiores de Monterrey (México) para a criação e compartilhamento de material didático para a *web*, destinados a apoiar o Ensino Fundamental. Vide: http://grove.ufl.edu/~pgl.

[23] É importante esclarecer que a partir de um recurso MIDI (como um teclado, por exemplo) pode-se gerar eventos que são interpretados pelo programa e convertidos para notação musical.

musical, o que significa a geração de duas informações: primeiramente a informação gráfica que deverá ser desenhada tela e, a seguir, sua associação com o signo musical representado.

Para o computador, tanto a imagem quanto seu significado não representam um som em si, mas a partir do significado da imagem, contido numa variável ou num objeto, é que será gerado o evento MIDI que por sua vez gerará o som.

3.4 A utilização de *applet* Java em *sites* de educação musical na Internet

Um dos objetivos deste trabalho é propor a Internet como uma aliada à educação musical e buscar soluções computacionais simples para a inserção rápida de partituras em páginas HTML, permitindo a criação de páginas de conteúdo de teoria musical que possam complementar o ensino formal de música.

Como já vimos anteriormente, existem dois caminhos através dos quais o autor do conteúdo didático pode disponibilizar partituras na Internet em *sites* de ensino de música: gerar seus exemplos musicais através de editores de partitura existentes no mercado e disponibilizá-los para *download* ou digitalizar as partituras e inseri-las nas páginas HTML em formato GIF ou JPEG.

Essas duas opções apresentam algumas desvantagens que serão analisadas a seguir.

No primeiro caso, apesar do fato de que muitos editores de partitura já são de uso comum entre a maioria dos músicos e estudantes de música, por serem proprietários, têm um custo de aquisição elevado e um professor que utilize um desses formatos estará obrigando seus alunos a adquirir o *software* ou pela compra ou pela pirataria. Além disso, as partituras postas para *download* não poderiam ser diretamente visualizadas no *browser* do usuário (aluno) e assim limitaria as possibilidades do uso pedagógico do *site*, uma vez que o aluno se veria obrigado a sair do navegador e abrir seu editor de partituras para visualizar o exemplo baixado da *web*.

No caso da digitalização das partituras, além de tornar necessário o uso de um *scanner* para digitalizar as partituras, quando da inserção das imagens na página HTML se perderia a parte mais importante da informação que é o som em si, o que obrigaria o autor do material disponibilizar também arquivos .wav ou .mid para *download* e assim novamente o aluno teria que deixar de lado o navegador, abrir uma ferramenta de áudio como *Real Player* ou o *Media Player*, por exemplo, para escutar os arquivos de som.

Como já foi introduzido no início deste capítulo, falta-nos uma ferramenta com a qual onde o professor autor do material didático possa escrever seus exemplos musicais para a aula e gerar um

arquivo que seja visível diretamente na página HTML e permita também a audição da partitura sem a necessidade da utilização ou instalação de qualquer outro programa no computador do aluno.

Diversos autores propõem o uso de *applets* Java em situações como a aqui apresentada, em que se faz necessário o uso de uma mídia não reconhecida pelos programas de navegação, como é a partitura.

Em [FRI99], encontramos a seguinte afirmação:

> "*A programação em Java torna possível o acesso a programas através da Internet, onde essa tecnologia começa a ser empregada no ensino à distância, sendo que programas para educação musical podem fazer uso desta técnica para incrementar sua difusão e uso remoto em laboratórios virtuais.*"

Também temos [SOA99] afirmando:

> "*Utilizar como base de desenvolvimento uma tecnologia multiplataforma, flexível, rica em recursos de programação, bastante difundida e que vem sendo empregada internacionalmente por diversas instituições comerciais e acadêmicas pode ser uma importante medida para a implementação, manutenção e, sobretudo, para o crescimento de um sistema que prevê a agregação de novos recursos*"

E ainda:

> "*A motivação de usar um software multiplataforma apresenta o Java como uma poderosa solução para a programação de aplicações para a Internet.*"

No entanto, o simples desenvolvimento de uma aplicação em Java para a Internet não resolve de toda a questão, pois tem que se considerar que a aplicação Java será apenas a interface que apresentará a partitura na página HTML. É preciso ainda obter-se um modo de encapsular as informações da partitura e transferi-las para o *applet* que se encarregará de processá-las. Uma solução que se faz pertinente a este problema é o uso do padrão XML para transportar as informações musicais do servidor para o cliente (*browser* do aluno).

A escolha do XML como solução ao problema aqui apresentado será abordada em detalhes no próximo capítulo.

4 Usando XML na codificação de partituras

A XML (*eXtensible Markup Language*), uma simplificação do SGML (*Standard Generalized Markup Language*), é uma linguagem de marcação de texto que se assemelha ao HTML (*HiperText Markup Language*), tendo como principal diferença a capacidade da criação de novas *tags* (marcas) capazes de reconhecer o significado das informações, podendo ser usado para transmissão de qualquer tipo de dado.

As *tags* são marcas delimitadas pelos sinais "<" e ">". Têm a função de demarcar uma região do documento definindo o modo de apresentação (exibição) ou o significado.

A XML apresenta muitas vantagens sobre o HTML, apesar de sua aparente semelhança, destacando-se sua flexibilidade e adaptabilidade, sua capacidade de especialização, sua interoperabilidade [SAN99]. Seu uso se destaca em sistemas onde é exigida uma rápida troca de informações como no comércio eletrônico, no acesso a banco de dados remotos, em educação à distância, entre outros.

O HTML trabalha com *tags* de marcação que são usadas para classificar o texto segundo sua apresentação. A XML também pode usar *tags* para definir parâmetros de apresentação, mas seu grande potencial está no fato de suas *tags* serem usadas para classificar o texto segundo seu significado.

Numa estrutura HTML, podemos identificar dois grandes blocos: o cabeçalho, onde encontramos marcas que definem informações sobre o conteúdo da página (meta-dados) e o corpo da página, onde o conteúdo é apresentado (dados).

Para esclarecer, vejamos uma estrutura HTML típica:

```
<html>
    <head>
        <title>Página da Web</title>
        <meta name="GENERATOR" content="bloco de notas">
    </head>
    <body>
        <p>Este texto aparece com caracteres normais
        <p><b> Este aparece em negrito </b>
        <p><i> Este está em itálico</i>
        <p><u> Este é sublinhado</u>
        <p><u><b><i>e este esta sublinhado, negrito e itálico</i><b></u>
        <p><font color=#ff0000> Este texto é vermelho</font>
    </body>
</html>
```

A marca <html> indica que o documento está no formato HTML, as marcas <head> e <body> dividem o conteúdo em cabeçalho e corpo respectivamente. Dentro da marca <head> encontramos <title> para designar o título do documento e <meta> onde é informado o nome do editor usado para a construção do documento. Na marca <body> aparece <p>, indicação de parágrafo, <b> para negrito (*bold*), <i> para itálico e <u> para sublinhado (*underline*). Também encontramos <font color=#ff0000> indicando que a cor da fonte é vermelho, segundo o padrão RGB (*red, green, blue*). Abaixo podemos ver o resultado obtido com o código exemplificado:

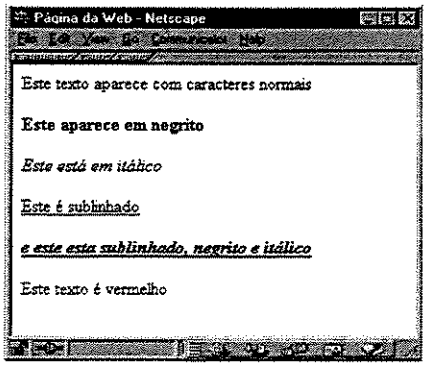


Figura 2 arquivo HTML visto no *browser*

Como podemos perceber, a marca <meta> apenas presta informação sobre o código escrito - neste caso, qual editor foi utilizado. Porém, não interfere nem no conteúdo nem na apresentação. As demais *tags* estão diretamente ligadas à apresentação do conteúdo.

Também podemos perceber que as *tags* aparecem aos pares, isto é, uma *tag* de abertura (ex.:) e outra de fechamento (ex.:) – apesar disso não ser uma regra já que existem *tags* que aparecem solitárias (ex.: <p>,
).

A aparência de um código em XML, é muito semelhante ao HTML aqui apresentado, no entanto, o significado de cada marca é bastante diferente, uma vez que não existe uma preocupação direta com a apresentação, mas com a significação do conteúdo.

Imaginemos que tenhamos que promover a circulação de dados de uma agenda de endereços de alunos de uma escola, onde deverão constar o nome, sobrenome, filiação com nome e sobrenome, endereço com rua, número, complemento, bairro, cidade, estado, país, cep, telefone. Assim, poderíamos encontrar um código semelhante a:

```
<?xml version="1.0"?>
<agenda nome="NOMEDAAGENDA" valor="ALUNOS">
    <aluno id="035">
        <sobrenome>Silva</sobrenome>
        <nome>Pedro</nome>
        <filiacao>
            <mae>
                <sobrenome>
                    Santos
                </sobrenome>
                <nome>
                    Maria Aparecida
                </nome>
            </mae>
            <pai>
                <sobrenome>
                    Silva
                </sobrenome>
```

```
            <nome>
                Jose Alberto
            </nome>
        </pai>
    </filiacao>
    <endereco>
        <logradouro tipo="RUA">
            15 de novembro
        </logradouro>
        <numero>
            33
        </numero>
        <complemento tipo="APARTAMENTO">
            102
        </complemento>
        <complemento tipo="BLOCO">
            B
        </complemento>
        <bairro tipo="VILA">
            São Joaquim
        </bairro>
        <cep>
            01033-020
        </cep>
        <municipio>
            São Paulo
        </municipio>
        <estado>
            SP
        </estado>
        <pais>
            Brasil
```

```
            </pais>
        </endereco>
        <telefone tipo="FIXO">
            <area>
                11
            </area>
            <numero>
                30334567
            </numero>
        </telefone>
        <telefone tipo="CELULAR">
            <area>
                11
            </area>
            <numero>
                91337654
            </numero>
        </telefone>
    </aluno>
</agenda>
```

Neste exemplo, vemos como a XML pode ser usada para estabelecer um significado para a informação por ela contida. Cada *tag* representa um tipo de conteúdo com um significado muito bem definido. Esse significado pode abranger desde um grande conjunto de informações - uma agenda inteira (<agenda>), onde sabemos que seu conteúdo define uma agenda, ou um endereço <endereco>, onde se subentende conter rua, número, bairro, cidade, etc., ou filiação onde se espera encontrar nome do pai e da mãe da pessoa registrada - até informações muito mais específicas como nome e sobrenome, número de telefone, nome da rua.

Podemos observar que o código XML permite não somente a sua designação de significado como nas marcas <nome> e <sobrenome>, mas também a qualificação de seu conteúdo como em <logradouro tipo= "AVENIDA" > onde sabemos que seu conteúdo representa o nome de um logradouro e que este, por sua vez, trata-se de uma avenida e não de uma rua ou uma travessa. O mesmo acontece

com a marca <telefone tipo="CELULAR"> que, além de definir seu conteúdo como sendo o número de um telefone, ainda garante tratar-se de um telefone celular e não de um telefone fixo ou via satélite ou público.

Essa capacidade de representação do significado confere ao XML grande poder de compatibilidade com diversos tipos de sistemas e ambientes de trabalho, possibilitando a integração e a troca de informações entre sistemas distintos; torna mais ágil e efetiva a utilização de ferramentas de busca; permite a total padronização ou a total personalização do conteúdo representado através de suas *tags*. No exemplo acima, todas as marcas foram criadas para atender uma necessidade especificamente proposta na descrição do problema: a agenda de alunos de uma escola. Analisando o código do exemplo acima, veremos nele refletidas as seguintes características:

Interoperabilidade e compatibilidade - por tratar-se de um código de texto sem formatação, pode ser lido e interpretado pelas mais diferentes ferramentas e pode circular entre os mais diversos sistemas operacionais existentes.

Ferramentas de busca - por estabelecer o significado de seu conteúdo, torna-se fácil extrair uma determinada informação contida no documento. No exemplo mostrado, poderíamos realizar uma consulta em busca dos pais de um determinado aluno através dos dados contidos nas marcas <filiacao>; poderíamos consultar com facilidade quantos e quais alunos moram em apartamento ou quais alunos possuem celular. A consulta poderia, também, ser delimitada pelo atributo "NOMEDAAGENDA" da marca <agenda> forçando a busca a se limitar à agenda de alunos ou ao contrário, a abranger diversas agendas diferentes.

Personalização - possibilita que a escolha de um conjunto de *tags* seja feita de acordo com a necessidade encontrada especificamente naquele cenário.

Padronização - a escolha adequada das *tags* que serão utilizadas em um determinado sistema facilita a comunicação entre os personagens envolvidos, estabelecendo um cenário onde todos os personagens e sistemas utilizarão as mesmas *tags*.

Representação Semântica - pelo fato das *tags* indicarem o significado de cada conteúdo, é possível sua utilização para a localização de idéias contidas no documento, bem como informações bem específicas acerca do conteúdo.

Legibilidade - por tratar-se de uma arquivo contendo apenas um texto subdividido segundo seu significado e não segundo sua aparência ou apresentação, permite a fácil leitura e edição por qualquer

pessoa, sem a necessidade extrema de uma ferramenta de edição mais complexa que um simples editor de textos.

Baixo custo computacional - também por tratar-se de arquivo de texto, o desenvolvimento de ferramentas computacionais é bastante facilitado, sendo que hoje é possível encontrar, não só APIs para o desenvolvimento de *software* que usam XML, quanto ferramentas de edição específicas para XML, ambos de domínio público.

Para se tornar possível o uso do XML e obter a correta troca de dados entre os sistemas envolvidos, é necessário que todas as partes conheçam as *tags* utilizadas no código em questão. Assim, essas *tags*, previamente acordadas entre os personagens que terão acesso ao documento, deverão ser registradas em uma espécie de contrato denominado DTD (*Document Type Definition*) ou Definição do Tipo de Documento, que deverá ser conhecido por todos os usuários do documento, sejam eles sistemas ou pessoas.

Uma DTD define como as *tags* deverão ser utilizadas no que se refere a sua estrutura, hierarquia e tipo de dado que contenham, sendo que cada *tag* define um **elemento** que, por sua vez, pode ser especificado por um **atributo.**

Mesmo um código HTML tem uma DTD usada internamente pelo *browser* que lê e interpreta o arquivo HTML.

A DTD serve, não só para orientar a construção de um novo documento, como também dá, aos sistemas que a utilizam, meios para verificar e validar sua consistência gramatical. Cabe ao sistema decidir por rejeitar um documento inválido ou tentar o aproveitamento das informações válidas nele existentes.

Para o caso proposto em nosso exemplo, a DTD seria algo parecido com:

<!ELEMENT agenda (aluno) >
<!ATTLIST agenda nome NMTOKEN #REQUIRED >
<!ATTLIST agenda valor NMTOKEN #REQUIRED >
<!ELEMENT aluno (sobrenome, nome, filiacao, endereco, telefone+) >
<!ATTLIST aluno id NMTOKEN #REQUIRED >
<!ELEMENT area (#PCDATA) >
<!ELEMENT bairro (#PCDATA) >
<!ATTLIST bairro tipo NMTOKEN #REQUIRED >
<!ELEMENT cep (#PCDATA) >

<!ELEMENT complemento (#PCDATA) >

<!ATTLIST complemento tipo NMTOKEN #REQUIRED >

<!ELEMENT endereço (logradouro, numero, complemento+, bairro, cep, municipio, estado, pais) >

<!ELEMENT estado (#PCDATA) >

<!ELEMENT filiacao (mae, pai) >

<!ELEMENT logradouro (#PCDATA) >

<!ATTLIST logradouro tipo NMTOKEN #REQUIRED >

<!ELEMENT mae (sobrenome, nome) >

<!ELEMENT municipio (#PCDATA) >

<!ELEMENT nome (#PCDATA) >

<!ELEMENT numero (#PCDATA) >

<!ELEMENT pai (sobrenome, nome) >

<!ELEMENT pais (#PCDATA) >

<!ELEMENT sobrenome (#PCDATA) >

<!ELEMENT telefone (area, numero) >

<!ATTLIST telefone tipo NMTOKEN #REQUIRED >

Através da DTD acima, podemos entender como seu XML deve ser estruturado, pois a DTD nos informa quais são os elementos que podem ser utilizados e quais são os elementos a eles subordinados através das marcas <!ELEMENT>. Por exemplo, podemos ver uma definição para os elementos "nome" e "sobrenome" e que estes são subordinados aos elementos "aluno" (*<!ELEMENT aluno (sobrenome, nome, filiacao, endereco, telefone+)*>), "mae" (*<!ELEMENT pai (sobrenome, nome)*>) e "pai" (*<!ELEMENT pai (sobrenome, nome)*>).Também podemos perceber que o elemento "pai", por sua vez, exige os elementos "nome" e "sobrenome".

Outro dado importante é que a DTD também informa a existência de atributos para um determinado elemento, como ocorre com "telefone" em nosso exemplo, no qual a marca *<!ATTLIST telefone tipo NMTOKEN #REQUIRED >* indica a presença do atributo "tipo" para o elemento "telefone", informando também tratar-se de um atributo de uso obrigatório (*#REQUIRED*).

4.1 Uso de XML na representação de partituras de música

O XML tem apresentado grande eficiência na implementação de sistemas de educação à distância baseados na *web* pois, possibilita uma representação estruturada dos dados, podendo encapsular qualquer tipo de informação, inclusive as de controle da apresentação ou visualização dos dados no *browser* do usuário. Também possibilita a identificação semântica, permitindo o desenvolvimento de sistemas capazes de entender os dados do ponto de vista de seu significado [SAN99]. Essa grande capacidade de representação da informação através da XML a coloca como uma excelente alternativa no desenvolvimento de soluções para educação musical à distância.

O uso de XML para a representação de partituras associado ao uso de *applets* Java para sua edição e visualização em páginas HTML pode simplificar o trabalho de autoria e criação de *sites* voltados à educação musical. Existe já à disposição, algumas soluções em XML para a representação de partituras e arquivos MIDI. Entre elas podemos citar:

- Xscore [GRI98];
- Music and Lyrics Markup Language [4ML];
- MusicXML [REC00];
- Music Markup Language [STE99];
- MusiXML [CAS98];
- FlowML [SCT00];
- MusicML [ROT97];

As grandes vantagens da utilização do XML para a construção de documentos contendo música, principalmente se empregados em ambientes de ensino mediados por computador, são:

- a possibilidade de se inserir no documento o conteúdo textual, criando uma forte associação entre a informação escrita e a informação musical;
- a inserção de controladores para a realização de exercícios;
- a inserção parâmetros para efetuar a correção automática de exercícios;
- a fácil conversão para outros formatos;
- a fácil transferência via *web* e aplicação em documentos HTML através do uso de *applets* Java.

Apesar de algumas das soluções acima citadas ressaltarem a importância do desenvolvimento de um código XML para troca eficiente de partituras via *web* em formato genérico e não proprietário [GRI98], [STE99], existe ainda o problema da visualização e edição das partituras. Mesmo considerando a simplicidade do código XML, além de sua legibilidade, a tarefa de transcrever uma partitura para qualquer sistema textual pode ser muito árdua, uma vez que o trabalho final pode vir a conter algumas dezenas ou até centenas de linhas de texto e a estruturação manual da XML com o auxílio único de um editor de textos é bastante sujeita a erros, apesar de estar longe de ser uma tarefa impossível, uma vez que, a estrutura da XML definida pela DTD, deve ser seguida à risca. Além do mais, mesmo tendo em mãos o arquivo XML (transcrito de uma partitura), para sua efetiva utilização será necessária sua apresentação em forma de partitura ou sua audição em forma sonora. Portanto, concluímos haver a necessidade de se desenvolver ferramentas para a edição das partituras e sua posterior visualização e audição.

A proposta MusicML da *The Connection Factory* © inclui em seu trabalho, não somente uma DTD para a representação musical em XML, mas também um *applet* através do qual é possível a visualização das partituras. Porém, a tarefa de construção do documento XML tem que ser feita manualmente, além de que o *applet* proposto não é capaz de tocar o conteúdo musical nem convertê-lo para outros formatos, como MIDI, o que permitiria sua execução (audição) em outros aplicativos.

Quando falamos de educação musical, não podemos deixar de considerar a necessidade da existência de um mínimo de ferramentas para o desenvolvimento das atividades didáticas. Dentre essas ferramentas, certamente a partitura constitui-se uma das mais importantes, pois tem a capacidade de transportar a codificação gráfica do conteúdo musical. É essa representação dos sons das notas musicais que será o alvo das próximas discussões, tendo em vista sua capacidade de congelar a idéia musical para que possa ser estudada, analisada e compartilhada.

Se considerarmos um ambiente gráfico como a Internet, perceberemos que uma partitura pode ser facilmente distribuída através de arquivos de imagem. Nesse contexto, o autor do material tem apenas que digitalizar as partituras que serão usadas em seu material didático e inseri-las nos documentos HTML que compõem o material. Também pode gerar arquivos de imagem através da captura direta da imagem da partitura a partir de editores de partitura existentes no mercado.

No entanto, este processo, apesar de permitir a troca de partituras e a composição de material didático para a *web*, não faz, desta, mais do que um simples livro eletrônico. O grande diferencial da Internet em relação a um livro que o aluno leva para casa e estuda, é que a Internet permite um nível de

interação e comunicação não somente com o conteúdo exposto, mas também com o autor do conteúdo e até mesmo com os outros usuários do conteúdo.

A Internet possibilita o uso de mídias diversas que podem expandir o potencial pedagógico do material em estudo, pois, além de mostrar as partituras ao aluno, também pode fazê-las tocar. Mas, para tanto, de nada adianta a simples inclusão de imagens digitalizadas das partituras: é preciso fornecer também a gravação dos sons representados por essas imagens. Isso aumenta a complexidade do processo de criação do material para se utilizar todos os recursos disponíveis na *web*. Não seguindo este caminho, resta como solução utilizar-se do recurso de disponibilizar as partituras em um dos formatos existentes atualmente no mercado, possibilitados pelo uso de editores como Finale [CODA], Sibelius [SIBE], Encore[24], entre outros, para que sejam baixadas pelos usuários. Porém, essa alternativa, além de obrigar o aluno e autores de conteúdo a optarem por um formato proprietário, ainda descarta o grande potencial multimídia existente na Internet.

A utilização de código XML para esse processo de troca, visualização e audição das partituras na *web* é a solução que melhor se adequa ao problema.

Com o uso de um único *applet* Java é possível a interpretação do arquivo XML contendo uma partitura, sua apresentação visual e por fim sua audição. Isso sem falar que a flexibilidade do XML permite que haja um total controle, tanto da forma como a partitura será mostrada, quanto como será tocada pelo *applet* Java.

Não que o uso de *applets* Java seja a única forma de possibilitar a apresentação e audição da XML musical na Internet pois, existe ainda a possibilidade do desenvolvimento de *browsers* com recursos para interpretar e exibir a partitura codificada ou o desenvolvimento de *plugins* capazes de conferir a um *browser* de uso comum tais recursos. Porém a utilização de *applets* Java apresenta uma larga gama de vantagens em relação a estas outras duas possibilidades.

A construção de um novo *browser*, especificamente voltado para a leitura de material musical ou mesmo a criação de *plugins* engloba um alto custo de implementação (projeto, programação, testes, etc.), além do quê tais *software* ficam limitados ao sistema operacional para o qual foram desenvolvidos. Ao contrário, após o desenvolvimento de um *applet*, seu uso se estende além dos limites do sistema operacional, poupa ao usuário instalação de *software* e reduz os elementos a serem configurados em seu sistema para obter um correto funcionamento.

Também apresenta vantagens do ponto de vista do autor do material: primeiro, a simplicidade com que um *applet* é incluído no documento HTML; segundo, a possibilidade de uso de versões

[24] Encore, software para edição de partituras da Passport Designs, saiu do mercado no início de 2001.

diferentes de *applets* para cada necessidade, o que é totalmente transparente para o cliente (aluno); por fim, a capacidade de um *applet* "conversar" com seu servidor de origem, o que permite o desenvolvimento de sistemas mais complexos associando *applets* (do lado do cliente) a *servlets* (do lado do servidor), no qual o autor ou o professor poderá coletar, remotamente, dados sobre o desempenho de seus alunos durante o processo de aprendizado ou mesmo interagir com eles em tempo real [SOA99].

Existem alguns fatores que dificultam o trabalho de representação de uma partitura pois, além da grande gama de elementos gráficos (notas, hastes, barras, claves, ligaduras, etc.) - que têm que ser representados não somente no que se refere ao seu desenho em si, mas também ao seu posicionamento e alinhamento, uma vez que a estética da escrita contribui para a transmissão da informação musical - as informações sobre a execução sonora destes elementos também têm que estar presentes. Por estas razões, notamos que existem no mercado inúmeras soluções gratuitas para arquivo e processamento de imagens, onda sonora e MIDI e pouquíssimas soluções gratuitas para o tratamento de partituras [BAR98].

4.2 Análise da DTD adotada por este trabalho

No decorrer da presente pesquisa, observou-se a necessidade de se propor uma DTD que possibilitasse a representação integrada de três elementos essenciais da partitura: os sinais gráficos de significado musical, o posicionamento espacial destes sinais e os parâmetros de controle da execução.

O início dos trabalhos foi delineado pela análise de um dos formatos XML já existentes, tendo sido escolhido o formato MusicML [ROT97], principalmente em função de já existir um *applet* que permitiria sua apresentação gráfica.

4.2.1 Analisando o MusicML

O princípio do MusicML é estruturar as informações segundo seu significado hierárquico dentro da representação musical. A proposta MusicML tem uma implementação de um *applet* Java - inclusive com as fontes disponíveis para estudo - bastante simples, vindo de encontro às necessidades deste projeto e, assim, tendo sido tomado como ponto de partida.

Vamos analisar alguns exemplos extraídos do MusicML para entendermos mais tarde como o modelo usado neste trabalho foi definido.

```
<sheetmusic>
    <musicrow size="two">
    </musicrow>
    <musicrow size="two">
    </musicrow>
</sheetmusic>
```

Neste exemplo, encontramos duas *tags*: a primeira, *<sheetmusic>*, define o corpo do documento, contento todos demais elementos; a segunda, *<musicrow size="two">*, define um pentagrama[25] com dois compassos[26].

```
<segment>
    <subsegment position="one">
        <note beat="quarter" name="e"/>
        <note beat="eighth" name="g"/>
        <chord>
            <note beat="sixteenth" name="f"/>
            <note beat="quarter" name="a"/>
        </chord>
        <rest size="half"/>
    </subsegment>
</segment>
```

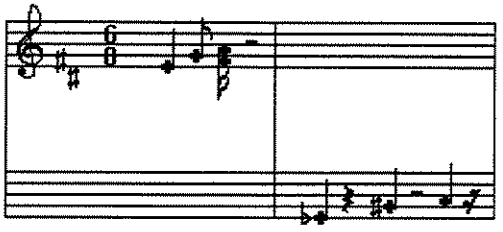


Figura 3: Partitura desenhada pelo *applet* MusicML

[25] Pentagrama: conjunto de cinco linhas paralelas onde a música é escrita.
[26] Compasso: unidade métrica na qual a música é dividida, responsável pela definição rítmica da música.

No trecho de código acima, encontramos ** que se refere ao primeiro compasso e *<subsegment position="one">* referindo-se à primeira linha (pentagrama). A *tag <note beat="quarter" name="e"/>* nos diz que a primeira nota desta música é um "mi" (*name="e"*) e sua duração é uma "semínima" (*quarter*). Mais adiante temos *<chord>* que agrupa duas notas em um acorde[27] e *<rest size="half">* indica uma pausa de "mínima" (*half*).

Observando este pequeno trecho de código, percebemos que não existe nenhum elemento ou atributo capaz de controlar a posição em que cada nota deverá ser desenhada. Também não há nada que indique como estas notas deveriam ser tocadas (andamento, intensidade, timbre, articulação, etc.). Apesar disso, o *applet* usado para renderizar a partitura calculou automaticamente a distância entre as notas, mas se o autor desejasse um espaçamento maior ou menor não haveria como indicá-lo. Também podemos supor que, se o *applet* que possibilitou a visualização desta partitura tivesse a capacidade de tocá-la, este proporia um andamento mediano (entre 80 e 100 batidas por minuto), uma intensidade *mezzo-forte*, uma articulação destacada, mas isto dependeria da implementação do programa usado para tocar o XML, pois no modelo proposto pelo MusicML não existem parâmetros capazes de interferir na execução sonora.

Uma vez que o objetivo deste trabalho é obter um modelo de representação da partitura que permita o controle tanto da visualização quanto da audição, a proposta MusicML teve que ser descartada e uma nova solução foi proposta, apesar do modelo examinado acima continuar servindo de referência para a busca de uma nova solução. Outro problema em relação ao MusicML é que o fluxo do texto não traduz com clareza o significado musical nele contido, o que dificulta sua edição manual e sua conversibilidade para outros formatos.

4.2.2 Proposta de uma nova DTD

Antes de entrarmos no aspecto da representação da escrita musical, vamos analisar a estrutura geral da DTD proposto por este trabalho.

O novo modelo para representação de partituras em XML pode ser dividido em duas partes básicas: a primeira, é usada para identificar a música e seu autor e as dimensões da partitura; a segunda, contém os elementos musicais representados em forma textual.

A primeira *tag* que encontramos é *<music>* e sob ela se subordinam as demais *tags* previstas

[27] Acorde: conjunto de notas tocadas simultaneamente.

nesta DTD. A *tag* <*music*> possui um atributo "*version*" onde é informada a versão da DTD usada. Esse atributo foi adotado pois o *software* que está sendo usado atualmente para editar e ler os documentos XML gerados não faz a validação do documento contra a DTD. Um documento XML, segundo o modelo adotado neste trabalho, poderá conter somente um elemento "*music*", o que significa dizer que, para cada partitura, deverá ser criado um novo arquivo XML.

Abaixo do elemento "*music*" estão os elementos "*inits*" e "*staf*". Ao elemento "*inits*" se subordinam dois outros elementos: "*document*" e "*author*", sendo que o primeiro deverá conter o nome da música à que se refere a partitura e o segundo conterá o nome do autor da música. A *tag* <*inits*> pede também dois atributos de uso obrigatório: "*length*", cujo valor corresponde ao comprimento dos pentagramas em *pixels* e "*numberofstaves*", que armazena a quantidade de pentagramas que deverá ser usada na partitura. Os elementos "*staf*" contêm todos os elementos musicais que descrevem a partitura. Sua *tag* (<*staf*>) pede um atributo "*number*" que representa o número do pentagrama.

A estrutura de um documento XML descrevendo uma partitura tem a seguinte aparência:

```
<?xml version="1.0"?>
<!DOCTYPE music SYSTEM "Javamusic.dtd">
<music version="1.0">
    <inits length= "400" numberofstaves= "2">
        <document></document>
        <author>desconhecido</author>
    </inits>
    <staf number="1">
        <elementos_musicais atributo="valor"/>
        ...
    </staf>
    <staf number="2">
        <elementos_musicais atributo="valor"/>
        ...
    </staf>
</music>
```

Todos os elementos musicais serão tocados e renderizados na ordem em que aparecerem no documento XML.

4.2.3 A representação das notas

Para se atingir um modelo para a DTD que atendesse as expectativas deste projeto, primeiramente foram analisados os elementos da partitura do ponto de vista da informação neles contidas.

A começar pelas as notas musicais: temos sua representação gráfica na partitura nos que traz em primeiro plano duas informações distintas: a altura (freqüência ou afinação) e a duração. A informação sobre a intensidade pode estar presente ou não, mas entende-se que uma nota musical, em seu âmago, contém altura, duração e intensidade (o quão fraco ou forte ela soa).

O gráfico abaixo dá-nos uma amostra de como estes três atributos estão presentes nas notas musicais, ressaltando a importância de serem representados explicitamente pelo código XML:

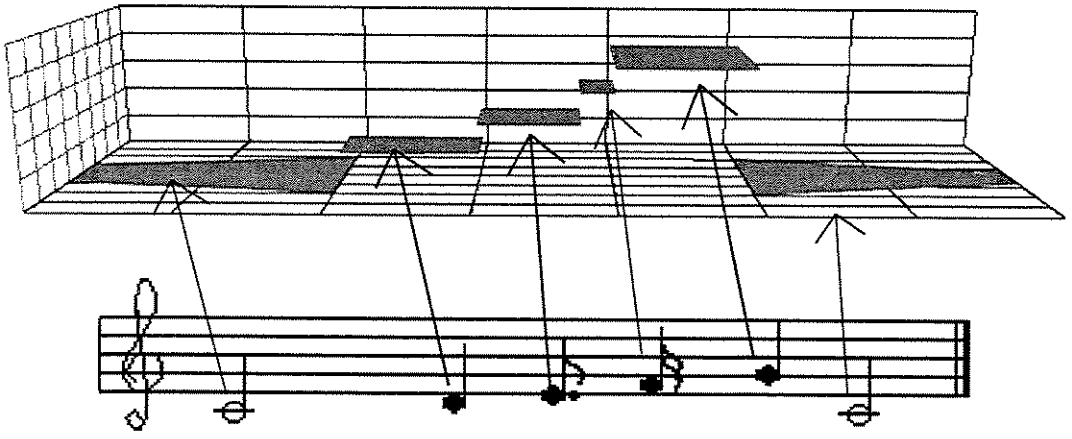


Figura 4: Notas musicais projetadas num gráfico 3D

Como podemos ver na figura acima, uma nota musical representa um objeto com três dimensões bem definidas: a duração no eixo *x*, a altura no eixo *y* e a intensidade no eixo *z*. Transferindo estas três dimensões para nosso modelo XML, podemos supor que o elemento "nota" terá pelo menos três atributos para representar as três dimensões da nota. Porém, antes de conhecermos esses atributos, é preciso discutir um pouco mais sobre a natureza específica de um deles: a altura.

O entendimento musical da altura de uma nota não está exatamente no domínio da freqüência da onda que irá compor a nota musical. Seu entendimento é totalmente abstrato uma vez que cada freqüência é reconhecida por um nome específico - que se repete a cada oitava da freqüência - a saber: dó, ré, mi, fá, sol, lá e si.

Para melhor entendimento desta questão, consideremos a nota lá. Por padrão, foi adotada a freqüência de 440Hz para esta nota. A freqüência de 880Hz corresponde à mesma nota lá, uma oitava acima e a freqüência 220Hz corresponde à nota lá uma oitava abaixo. Além disso, uma nota musical também pode sofrer elevação ou rebaixamento em sua freqüência provocada por um fenômeno chamado **acidente**. Existe cinco tipos de acidentes capazes de elevar ou rebaixar a freqüência das notas. São eles:

♯ **sustenido** - eleva a nota em um semitom[28];

𝄪 **dobrado-sustenido** - eleva a nota em um tom (dois semitons);

♮ **bequadro** - retorna a nota a sua freqüência ou altura original;

♭ **bemol** - rebaixa a nota em um semitom;

𝄫 **dobrado-bemol** - rebaixa a nota em um som (dois semitons).

Assim, percebemos que precisamos definir três atributos para descrever a dimensão da freqüência de uma nota: nome da nota, oitava e acidente (também chamado de sinal).

Considerando o argumento acima, foi definida para as notas musicais a *tag* "*note*" com os atributos "*sound*", "*eighth*" e "*signal*" para descrever a altura da nota em termos de nome, oitava e acidente respectivamente, "*duration*" para a duração da nota e "*attack*" para a intensidade, sendo a duração representada pelo nome das figuras musicais SEMIBREVE, MÍNIMA, SEMINIMA, COLCHEIA, SEMICOLCHEIA, FUSA E SEMIFUSA e a intensidade representada numericamente[29] por uma valor entre 0 e 127.

Além dos atributos acima citados para o elemento <*note*> foram definidos também os atributos "*start*" e "*end*" que interferem, respectivamente, no instante de ataque (início) da nota (podendo

---

[28] Em termos de freqüência, corresponde a um doze avos de uma oitava, ou seja, dada uma nota e sua oitava superior (dobro de sua freqüência), podemos subdividir a distância entre elas em doze partes aproximadamente iguais (sistema temperado) onde se encaixam as sete notas mais seus cinco sustenidos. Fazendo uma correlação com o teclado do piano, entre uma nota dó e a nota dó imediatamente seguinte (uma oitava acima) teremos sete notas brancas (dó, ré, mi, fá, sol, lá, e si) e cinco notas pretas (dó# ou ré*b*, ré# ou mi*b*, fá# ou sol*b*, sol# ou lá*b* e lá# ou si*b*).

[29] A adoção de um sistema numérico para a representação da intensidade foi a alternativa encontrada para correlacionar a intensidade diretamente com o sistema MIDI que utiliza valores entre 0 e 127 (0x00 a 0x7F). À semelhança da altura das notas, a representação da intensidade pelas palavras "*forte*" ou "*piano*" não expressa um valor definido de intensidade em db ou em qualquer outro sistema de medida, pois trata-se apenas de uma abstração. Usando uma escala numérica, como no sistema MIDI, o autor poderá usar como abstração algo entre 0 e 10 para "*pianíssimo*" (fraquíssimo) e algo superior a 120 para uma nota "*fortíssimo*", por exemplo.

adiantá-lo ou atrasá-lo em até 100% do valor de duração da nota) e no instante da finalização da nota (podendo, igualmente, alterá-lo em ± 100% da duração da nota).

Para assegurar o correto posicionamento de cada nota, foi criado o atributo "*position*", cujo valor corresponde à distância, em *pixels* da margem esquerda da área de apresentação.

Também foi criado um atributo chamado "*status*" que permite determinar se a nota deve ou não ser desenhada e se deve ou não ser tocada. Os valores possíveis para este atributo são: "*h*", "*m*" e "*hm*" que significam *hide, mute* e ambos, para esconder a nota, não tocar a nota ou escondê-la e não tocá-la, respectivamente.

A descrição final de uma nota pode se parecer então com:

<note sound="G" eighth="5" signal="SHARP" duration="SEMINIMA" attack="51" position="105" start="10.0" end="-50.0" status="h" />

Aqui vemos representada a nota Sol sustenido, com duração de SEMÍNIMA e intensidade 51 na graduação padrão MIDI. Ao ser executada pelo programa esta nota se iniciará com um décimo de semínima de atraso e durará metade de seu valor (*stacatto*).

Os atributos "*signal*", "*attack*", "*start*", "*end*" e "*status*" são de uso opcional.

4.2.4 A representação das pausas

A representação de uma pausa pode ser muito simples, uma vez que, não havendo som pois representam o silêncio na música, precisamos apenas de atributos para representar sua duração e os aspectos relativos à sua apresentação gráfica.

A *tag* que representa as pausas recebeu o nome de *<pause>* e possui os atributos: "*duration*" para a duração, "*status*" para controlar sua exibição (podendo receber o valor "*h*" quando for desejado que não seja desenhada) e "*position*" para determinar seu posicionamento.

O código abaixo exemplifica a descrição de uma pausa:

<pause duration="SEMINIMA" status="h" position="105"/>

4.2.5 As claves

As claves indicam, na escrita musical, quais as notas que serão representadas em cada uma das linhas do pentagrama. Para descrevê-las, precisamos apenas saber seu nome e sua posição. Assim, temos os seguintes atributos para a *tag* <*clef*>: "*name*" para indicar o nome da clave, (podendo assumir os valores "G_CLEF", "F4_CLEF",. "C3_CLEF", "C4_CLEF" e "P_CLEF"); "*position*" para a posição da clave e "*status*" para exibi-la ou não.

<clef name="G_CLEF" position="42"/>

4.2.6 Armaduras de claves

Como as armaduras de clave referem-se à tonalidade da música, sua descrição é baseada no nome das tonalidades. A *tag* que as identifica é <*keysignature*> e possui os atributos: "*tone*", "*status*" e "*position*". No exemplo a seguir, vemos representada a armadura de clave da tonalidade de mi bemol maior e/ou dó menor:

<keysignature tone="EbM/Cm" position="57"/>

4.2.7 Fórmula de compasso

A fórmula de compasso traz duas informações musicais importantes: quantos tempos têm cada compasso (unidade de compasso) e qual a figura representa a duração de um tempo. A *tag* que a representa é <*timesignature*> e possui os seguintes atributos: "*rhythm*" para representar a fórmula de compasso assumindo exatamente o valor expresso na partitura (C, 2/4, 3/4, 9/8, etc.), "*status*" e "*position*". Exemplo:

<timesignature rhythm="3/4" position="84"/>

4.2.8 Representação dos andamentos

Os andamentos indicam em que velocidade a música será executada, ou seja, quantos tempos serão tocados por minuto, podendo ter seu valor alterado no decorrer da música. A *tag* <*tempo*> foi escolhida para descrevê-los e possui os atributos "*bpm*" (*beats per minute*), "*unit*", para indicar qual figura musical representa um tempo (*beat*), "*height*" para indicar a posição vertical em que a indicação do tempo será desenhada (distância em *pixels* do topo da partitura), "*position*" e "*status*". Exemplo:

<tempo bpm="85" unit="SEMINIMA" heigth="89" position="118"/>

4.2.9 Barras de compasso

As barras de compasso indicam: onde começa e termina cada compasso da música; as divisões da idéia musical ou frases musicais (barras duplas), as repetições (barras duplas seguidas de pontos) e o final da música (uma barra fina e uma barra grossa). A *tag* adotada é <*barline*> e possui o atributo "*type*" que pode assumir os valores "*Beguin*", "*End*", "*Single*", "*Double*", "*RepeatLeft*", "*RepeatRight*", "*RepeatDouble*" e "*EndScore*", além dos atributos "*position*" e "*status*". Exemplo:

<barline type="Single" position="50"/>

4.2.10 Textos

Nesta primeira versão da DTD, a representação de textos dentro de uma partitura, sejam eles referentes à letra da música ou indicação de elementos musicais como metrônomo, dinâmica, acelerando e retardando, intenção da frase e outros, trata todos os textos como textos simples que não têm nenhum significado a não ser para o usuário.

A *tag* usada é <*text*> e possui os parâmetros "*string*" cujo valor corresponde ao texto a ser inserido no documento, "*height*" para indicar a posição vertical em que o texto será mostrado (distância em *pixels* do topo da partitura) além de "*position*" e "*status*". Exemplo:

<text string="crescendo" height="144" position="47"/>

4.2.11 Descrevendo grupos de notas

Os grupos de notas podem representar, musicalmente falando, o sentido de uma frase, a marcação rítmica ou podem simplesmente facilitar a leitura da partitura.

A seguir, temos duas figuras representando o mesmo trecho de música, sendo que na primeira figura as notas aparecem separadas e na figura seguinte aparecem agrupadas:

Figura 5: Notas separadas

Figura 6: Notas agrupadas

Para representar os grupos de notas foi criada a *tag* *<group>*. Além dos atributos "*position*" e "*status*", esta *tag* possui também o atributo "*size*" que informa quantas notas fazem parte do grupo. Vejamos, a seguir, um exemplo de como será usada a *tag* *<group>*:

```
...
<group size="2" position="324"/>
<note sound="B" eighth="5" duration="COLCHEIA" position="325"/>
<text string="pp" heigth="339" position="342"/>
<note sound="A" eighth="5" duration="COLCHEIA" position="347"/>
...
```

Como podemos ver, no código acima, temos os elementos "*group*", "*note*", "*text*" e "*note*". O atributo "*size*", com o valor 2, indica ao leitor ou ao *software* que irá renderizar a partitura, que as duas notas imediatamente seguintes à *tag* "*group*" devem ser agrupadas, independente de haver entre as *tags* *<note>* outras *tags* de qualquer natureza. Esta construção permitiu maior simplicidade na arquitetura do *applet* Java desenvolvido para a visualização da partitura codificada em XML, evitando conflitos com a

tag "*quialtera*" que será apresentada na seção seguinte. Para futuras versões, no entanto, pretende-se alterar o formato da *tag* "*group*" de modo que esta contenha as demais *tags* numa estrutura mais hierarquizada.

4.2.12 Representando as quiálteras

As quiálteras na música têm a função de alterar a duração de um conjunto de notas de modo a fazer com que, num determinado período de tempo, ocorram mais ou menos notas. Para entender melhor, imaginem um ritmo musical onde ocorre uma seqüência de notas agrupadas duas a duas e, no meio desta seqüência, aparece um grupo com três notas, mas que duram o mesmo período de tempo que duas notas da mesma seqüência.

A *tag* <*quialtera*> possui os atributos "*duration*", que informa o tempo de duração total de todas as notas da quiáltera, sendo representado pelas figuras de tempo das notas (semibreve, mínima, semínima, etc.); "*size*", que informa quantas notas fazem parte da quiáltera; "*index*", que representa a relação temporal entre a duração das notas efetivamente escritas e o tempo total de duração da quiáltera; "*height*", "*position*" e "*width*", que definem respectivamente a distância do topo da área de apresentação, a distância da margem esquerda da área de apresentação e a largura da ligadura de quiáltera que deverá ser desenhada.

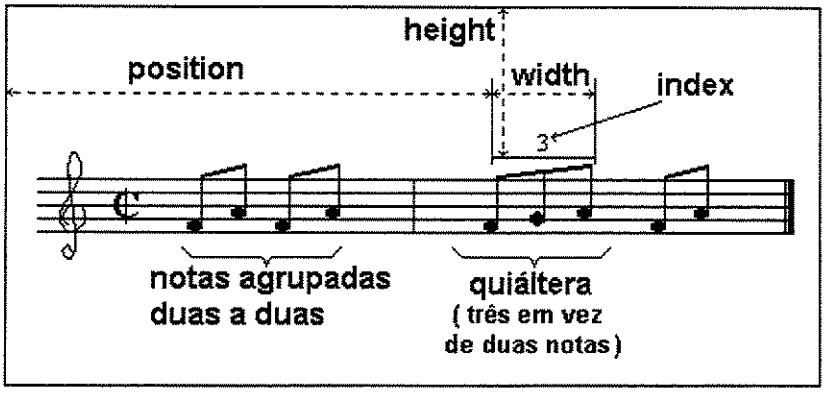


Figura 7: quiáltera

O código a seguir descreve em XML a quiáltera mostrada na figura.

<group size="3" position="252"/>

<quialtera duration="SEMINIMA" size="3" index="3" height="90" width="56" position="252"/>

<note sound="F" eighth="5" duration="COLCHEIA" position="253"/>

<note sound="G" eighth="5" duration="COLCHEIA" position="278"/>

<note sound="A" eighth="5" duration="COLCHEIA" position="303"/>

No apêndice A pode ser visto a integra do DTD para o modelo de XML adotado neste trabalho e descrito acima e no apêndice B é mostrada uma partitura e o código XML que a descreve.

5 O *software* JavaMusic

No capítulo "Educação Musical e a Informática" foi abordado o uso de *software* educacional no ensino de música. Vimos que grande parte dos programas existentes no mercado é, em geral, relacionada com o treinamento auditivo e que existe também um grande número de programas desenvolvidos para a edição e impressão de partituras, geração de arquivos MIDI, seqüenciação e processamento de sons. Também discutiu-se sobre a carência de *software* voltado a construção de *websites* para ensino de música, nos quais os poucos recursos atualmente explorados estão ligados também ao treinamento auditivo através do uso de *applets* Java ou ao uso de figuras de partituras digitalizadas e *download* de arquivos MIDI. Tratamos em seguida do uso do padrão XML para criação de documentos contendo partituras, analisamos o formato MusicML e a partir deste, foi proposto um novo formato de partitura em XML mais adequado aos objetivos desta pesquisa. Neste capítulo, será apresentado o JavaMusic, uma ferramenta experimental para autoria de partituras em XML e integração com HTML através de *applets* Java, permitindo exibir e tocar uma partitura no ambiente da *web*.

A Sun Microsystems[30] disponibilizou a API JavaSound [JSO99] no início de 2000 como uma alternativa para o desenvolvimento de aplicações que necessitem dispor dos recursos de som para *wave* e para MIDI. O uso da API JavaSound está condicionado à instalação do *plugin* JRE 1.3 (*Java Runtime Environment*) [JRE]. Este *plugin* permite que *applets* executados no *browser* possam acessar os recursos de sistema como a placa de som através de uma interface interligando a máquina virtual do Java, onde o *applet* é executado, com o *hardware* responsável pelos recursos de som do computador.

[30] http://www.sun.com

A API JavaSound foi adotada neste trabalho e atende plenamente suas necessidades. No entanto, existem outras APIs disponíveis para a interconexão entre a máquina Java e os recursos de áudio do sistema. A proposta JSyn da SoftSynth.com [JSYN] é uma das APIs disponíveis que poderia ter sido usada no desenvolvimento do JavaMusic, porém o JSyn não oferece recursos para a criação de eventos MIDI, atuando somente no domínio da freqüência. Os exemplos de *applets* criados com essa tecnologia se mostraram bastante interessantes e eficientes em termos de velocidade de execução, mas nenhum exemplo foi encontrado que demonstrasse sua eficiência no tratamento da música tradicional. Já a API JavaSound, apesar de ser muito mais lenta na execução dos programas, traz diversos exemplos abrangendo seqüenciação de eventos MIDI, *jukebox* (caixa de música) e bateria (*Rhythm Groove Box*), sendo, portanto, adotada para o desenvolvimento deste projeto.

O JavaMusic é constituído de dois acessórios básicos que trabalham em conjunto: o aplicativo JavaMusic e o *applet* JM.

Veremos, a seguir, como funcionam o aplicativo e o *applet*, como eles podem ser usados na construção de *sites* para educação musical, os detalhes de sua implementação, de seus conceitos e de seus algoritmos internos.

5.1 Visão geral

O JavaMusic é um editor de partituras desenvolvido em linguagem Java, que permite escrever uma partitura de um trecho musical e gerar um arquivo XML baseado na descrição apresentada no capítulo anterior, bem como sua conversão para *midifile*. O *software* é composto de dois elementos: a aplicação, que consiste no editor onde o material é criado e um *applet* que permite a visualização da partitura dentro de uma página HTML.

A escolha da linguagem Java deve-se ao fato de possibilitar a portabilidade com os diversos sistemas operacionais, permitindo que uma mesma versão do *software* esteja disponível para diferentes plataformas e também para propiciar a reutilização de classes na arquitetura do *applet* que integra a ferramenta.

No desenvolvimento da API buscou-se uma estrutura de classes que permitisse representar, com o máximo de fidelidade, o significado dos elementos musicais para que a mesma estrutura usada na construção do JavaMusic pudesse ser reutilizada no desenvolvimento de outros programas, com fins educacionais ou não, para a área de música.

A API utilizada na arquitetura do *software* é dividida em quatro partes básicas:

- **classes de significação musical** - onde foram definidos elementos como notas, duração, claves, entre outros, sob o ponto de vista do seu comportamento musical;
- **classes de representação gráfica** - que geram a saída gráfica a partir dos objetos musicais a elas apresentados;
- **classes de conversão MIDI** - classes que convertem os objetos musicais em eventos MIDI e possibilitam sua audição e a geração de arquivos *midifile*;
- **classes acessórias** - classes destinadas ao processamento dos arquivos XML e às estruturas da interface gráfica (botões, janelas, *buffers*, etc).

Estas classes estão distribuídas em seis pacotes:

- **music.music**: contém as classes de representação dos elementos da notação musical;
- **music.graphics**: contém as classes de representação gráfica;
- **music.midi**: classes de conversão para MIDI;
- **music.util**, **music.beans** e **music.event**: classes acessórias.

Este *software* é experimental e ainda não contempla todos os recursos necessários à escrita e à execução musical. O objetivo do seu desenvolvimento é testar a aplicabilidade e a eficiência de *applets* Java e XML em *sites* para educação musical, estudar a representação computacional dos principais elementos da música e sua notação dentro do padrão XML.

É importante ressaltar que os critérios usados na escolha dos elementos que foram implementados nesta versão foram: primeiro, a importância ou necessidade de cada elemento para a representação da notação musical; segundo, a relação custo-benefício entre a importância de um determinado elemento e o grau de dificuldade para sua implementação. Portanto, elementos importantes dentro da representação musical como a escrita de acordes, melodias múltiplas e ligaduras foram descartados nesta primeira implementação tendo em vista o alto custo computacional para sua implementação. Nesta versão inicial do JavaMusic, somente é possível escrever melodias em uma única voz, sem acompanhamento e sem acordes.

A construção das interfaces do usuário procurou privilegiar a intuitividade, a visibilidade dos comandos e a clareza da interface para permitir que o JavaMusic pudesse ser facilmente utilizado, mesmo por pessoas que não tenham domínio sobre ferramentas computacionais.

Ao contrário dos *software* encontrados nas prateleiras, o JavaMusic não usa nenhum sistema de validação da escrita, permitindo que o usuário possa tanto escrever de maneira estritamente correta, do ponto de vista da notação da música tradicional, quanto carregar seu trabalho de erros de notação, possibilitando a criação de exemplos musicais bastante variados.

Também se buscou desenvolver recursos diferenciais em relação ao modo como o JavaMusic executa ou toca as melodias nele criadas. Tais recursos consistem de comandos para controlar a intensidade, a velocidade e a articulação dos sons, permitindo ao usuário autor do material a simulação de *swing* através da inserção de atrasos e antecipações na execução de cada nota. Esses recursos usados no JavaMusic serão tratados em detalhes nas seções seguintes onde o programa estará sendo descrito com mais profundidade.

5.2 Características e funcionalidades

Como já foi dito, o JavaMusic permite a criação de um arquivo XML segundo o padrão apresentado no capítulo anterior através de uma interface gráfica de modo que todo o processo de criação do documento XML se torne totalmente transparente para o autor. Isso significa que o usuário do JavaMusic não precisará conhecer uma única linha sobre XML para usar o JavaMusic.

O *software* permite escrever melodias em pentagramas com largura entre 300 e 600 *pixels*[31].

Quanto aos recursos de saída gráfica, o JavaMusic possibilita:

- escrever as claves de sol, fá, dó na terceira linha, dó na quarta linha e percussão;
- escrever as figuras de tempo e de pausa, de semibreve até semifusa, pontuadas ou não;
- escrever os sinais de alteração (sustenido, dobrado-sustenido, bequadro, bemol e dobrado-bemol);
- traçar oito tipos de barras de compasso: barra simples e dupla, barra de início e de fim de linha, barra de repetição à esquerda, à direita e dupla e barra de final;

[31] Considerando o fato de que a maioria dos usuários de computador trabalha com definições de vídeo entre 640X480 a 800X600, larguras de pentagrama de até 600 *pixels* estão dentro da área de visibilidade de, praticamente, qualquer configuração de vídeo.

- escrever as armaduras de claves em todas as tonalidades;
- inserir qualquer fórmula de compasso;
- inserir indicações de metrônomo;
- inserir textos diversos;
- formar quiálteras com qualquer combinação de tempos;
- agrupar ou não as bandeirolas das notas;
- tornar qualquer um dos símbolos acima invisíveis, quando visto no *browser* do aluno, através da função "*hide*";
- visualizar o conteúdo preparado no modo "tela cheia";
- inserir o conteúdo em páginas HTML.

Quanto aos recursos de execução da melodia, o JavaMusic possibilita:

- o controle sobre a dinâmica;
- o controle sobre a articulação;
- atrasar ou adiantar uma nota ou conjunto de notas;
- variar o andamento;
- bloquear a execução de uma nota ou conjunto de notas no *browser* do aluno através da função "*mute*".

Com essas características o JavaMusic supre ao professor de teoria musical recursos para criação de exemplos musicais em XML visíveis e audíveis em páginas HTML através do *applet* JM, parte integrante do JavaMusic. Estes exemplos poderão se constituir de aulas teóricas, exercícios de solfejo e ditados musicais.

5.3 Componentes

Uma abstração sobre a arquitetura do JavaMusic para facilitar a compreensão de seu funcionamento pode ser feita através da divisão do *software* em cinco grandes componentes, cada um com funções bem específicas.

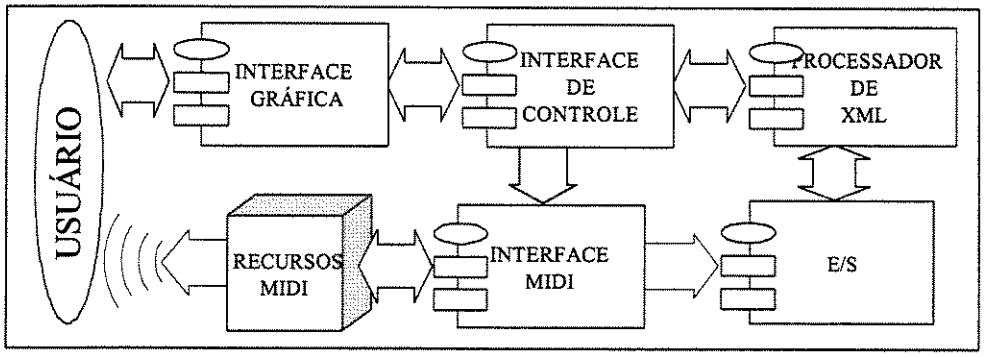


Figura 8: Diagrama de ligação entre os componentes internos do JavaMusic

Observando a Figura 8, percebemos que o JavaMusic se comunica com o usuário através da Interface Gráfica onde estão disponíveis todos os comandos necessários à sua operação. A Interface de Controle é o núcleo central do programa fazendo a ponte de ligação entre os comandos enviados pelo usuário e os demais componentes. O Processador de XML é o componente responsável pela redação e pela interpretação do código XML. O componente E/S tem como função básica a comunicação entre o programa e as unidades de disco, possibilitando a leitura e a gravação dos arquivos gerados. A Interface MIDI converte os dados do *buffer* interno do programa para uma seqüência de eventos MIDI e a encaminha tanto para a placa de som, onde é executada, quanto para a interface de saída (E/S) para salvamento em disco.

5.3.1 Interface Gráfica

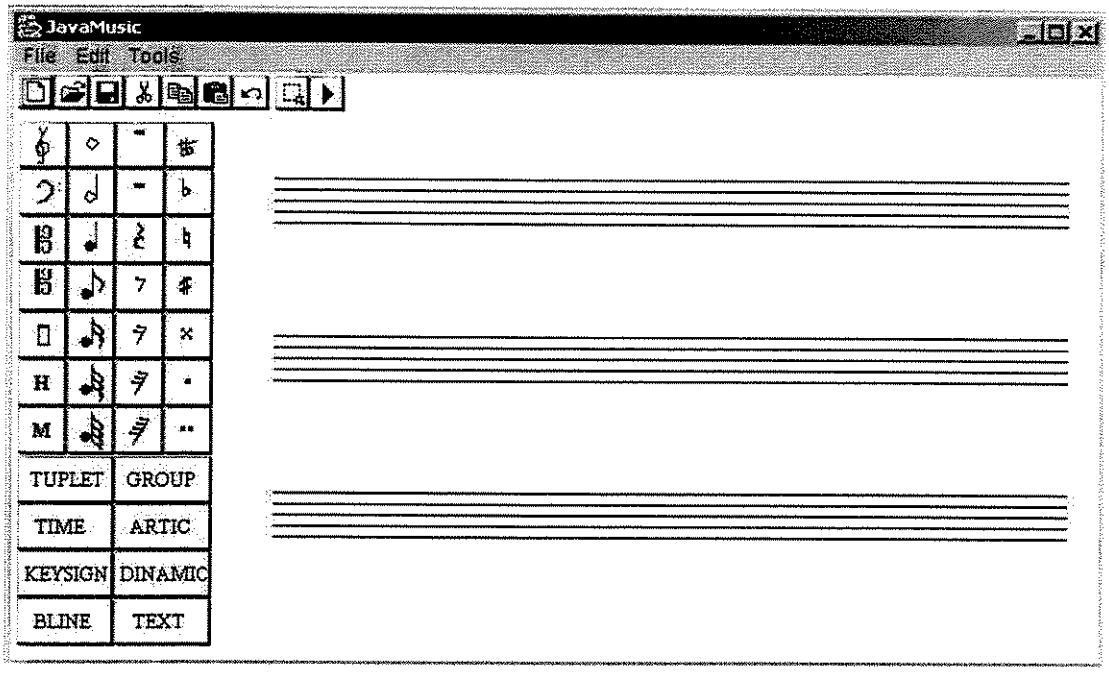


Figura 9: Interface do *software* JavaMusic

A maior preocupação com a interface gráfica do JavaMusic foi a de se obter uma disposição tal para os *widgets* (elementos da interface) que propiciasse ao usuário a maior comodidade possível durante o trabalho de edição. A idéia foi dispor todos os botões numa mesma janela e agrupá-los segundo sua utilidade. A inserção dos elementos gráficos principais como notas, pausas, claves e acidentes é feita através de um mecanismo de *drag and drop* (arrastar e soltar) diretamente no primeiro plano da interface. Os elementos secundários, ou de menor utilização, foram agrupados em caixas de diálogo e também operam através de *drag and drop*. São as barras de compasso, os textos, armaduras de clave e fórmula de compasso, além dos elementos de configuração da execução da partitura como dinâmica, articulação, quiálteras. As funções de agrupamento de notas, nota escondida e notas mudas são acessadas através de botões simples.

Para a construção da interface do usuário, foi usado o AWT (*Abstract Windowing Toolkit*)[32] da linguagem Java, ponto de partida para a elaboração da janela principal, dos menus, das caixas de diálogo e para a renderização de todas as figuras [DEI97].

[32] Vide http://java.sun.com/api

Para a construção dos botões foi usada a tecnologia JavaBeans[33], o que permitiu a personalização de cada botão a partir de um mesmo componente de *software*, bem como, sua reutilização na construção das barras de ferramentas [MOR99]. Construindo os botões em JavaBeans, foi possível definir personalizações para nome do botão, imagem de primeiro plano, dimensões, cor de foco, além de permitir a definição de dois modos de operação: *push button* e liga/desliga. Também é possível definir o botão para toque rápido ou convencional (no primeiro modo o botão é acionado apenas passando o ponteiro do *mouse* sobre ele, no segundo modo é preciso clicar o *mouse* para acionar o botão), apesar desta funcionalidade não ter sido usada, já que todos os botões do JavaMusic estão operando no modo convencional. O botão está definido na classe music.beans.MButton.

O princípio do JavaBeans consiste em troca de informações entre os componentes através de eventos. Existem dois tipos principais de eventos em JavaBeans: o primeiro é aquele gerado pela ação do usuário (como clicar um botão) ou pelo próprio componente (como um relógio); o segundo é gerado quando uma propriedade interna do componente (como estado, cor ou valores de variáveis) é alterada. Através desses eventos, os botões permitem ao sistema reconhecer ações como movimento, estado interno do botão (ligado ou desligado), duplo clique do *mouse*, entre outros.

As barras de ferramentas também foram construídas com JavaBeans. Um *container* é usado para encapsular os botões. As barras de ferramentas permitem personalizar sua disposição (vertical ou horizontal), a conexão ou não entre seus botões, cores de fundo e de foco, dimensão dos botões, imagem de primeiro plano para cada botão e a quantidade de botões. Permite ainda agrupar botões com características diferentes, de modo que, por exemplo, pode-se ter na mesma barra de ferramentas botões operando no modo *push button* outros no modo liga/desliga, sendo que para estes últimos é possível conectar dois ou mais botões para simularem botões de rádio, isso sem necessariamente conectar todos os botões. O *bean* barra de ferramentas está definido na classe music.beans.MBar.

O uso de JavaBeans na construção da interface gráfica simplificou muito o trabalho de montagem do *layout* dos elementos gráficos e permitiu uma eficiente conexão entre a interface gráfica e os demais componentes do JavaMusic através dos eventos gerados por cada ação sobre os *beans*.

Esta implementação do JavaMusic propõe duas classes para a interface gráfica. A primeira, music.JavaMusic, é a interface gráfica do aplicativo JavaMusic e a ela estão conectados todos os *beans*. A segunda classe é a music.JM que define a interface gráfica do *applet* que será distribuído via *web*. Esta classe não possui nenhum *bean* conectado a ela pois, nesta versão, o *applet* não permite a edição do material, limitando-se à sua apresentação.

---

[33] Vide http://java.sun.com/products/javabeans

No aplicativo, foram usadas duas barras de ferramentas, uma contendo os botões padrões como abrir, salvar, copiar e colar, etc. e outra contendo os botões para a notação musical.

As funções da barra de ferramentas padrão podem ser acessadas também através dos menus e dos atalhos de teclado atendendo, assim, diversos perfis de usuários.

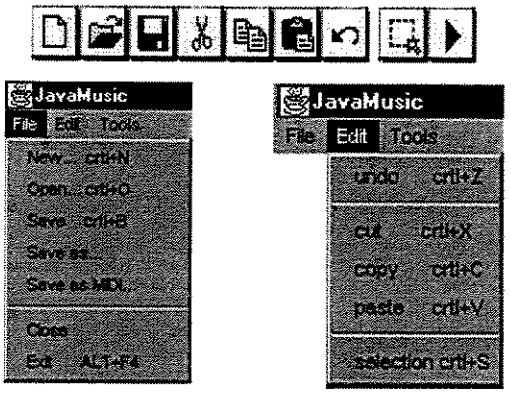


Figura 10: Barras de ferramenta padrão e menus

A barra de ferramentas de edição musical contém todos os comandos e botões necessários para a notação musical, segundo os recursos abrangidos neste programa. As três fileiras de botões pequenos (da esquerda para a direita) são botões do tipo *push button* e ao serem apertados definem qual símbolo deverá ser impresso na partitura. A fileira mais à direita contém botões do tipo liga/desliga e são usados em combinação com os outros botões.

Os botões maiores abrem caixas de diálogo contendo outras funções e símbolos. Vide figura a seguir.

Outro *bean* foi construído para integrar a interface gráfica. Trata-se do botão linear, uma espécie de potenciômetro linear virtual, definido pela classe music.beans.LinearButton. Este botão é usado para definir valores dentro de uma escala cujos limites podem estar entre os infinitos negativo e positivo e que são estabelecidos no momento da instanciação do *bean*. O LinearButton também permite a definição das propriedades de dimensão, nome, legenda, borda, com ou sem saída numérica (monitor) e cor de fundo. Foi usado na construção das caixas de diálogo de dinâmica e articulação (figura 11).

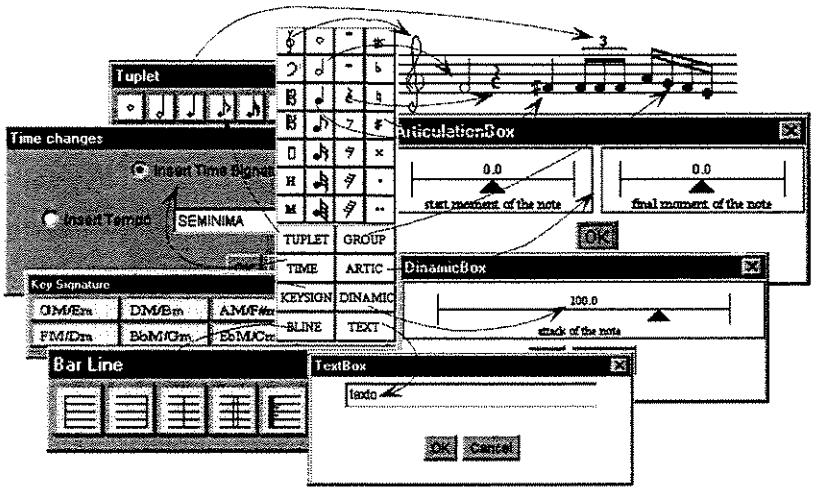


Figura 11: Barra de edição musical e caixas de diálogo

5.3.2 Interface de Controle

A Interface de Controle é a seção do programa responsável pelo processamento das ações do usuário e sua integração entre todos os demais componentes do programa. Duas interfaces de controle foram implementadas, uma para o aplicativo, definido em music.Music e outra para o *applet*, definido em music.MusicPanel.

O JavaMusic utiliza diversas classes que ao serem instanciadas se tornam *buffers* de informações necessárias ao processamento. Além disso, para cada conjunto de símbolos musicais, existe uma classe que desenha os símbolos na saída gráfica usando as informações contidas nesses *buffers* e uma classe que armazena o significado musical de cada símbolo.

Tanto as classes de *buffer* como as classes que geram as saídas gráficas são usadas no corpo do aplicativo JavaMusic (classes music.JavaMusic e music.Music) e do *applet* (classes music.JM e music.MusicPanel), mas as classes de saída gráfica precisam ter conhecimento se estão gerando a saída através do aplicativo ou através do *applet*. Isso porquê, no caso dos aplicativos, são usadas cores para diferenciar os símbolos que deverão ser desenhados pelo *applet* dos que não deverão ser desenhados, apesar de estarem presentes na partitura, e também para diferenciar os símbolos que deverão ser tocados daqueles que não deverão ser tocados, sendo que, no caso do aplicativo, esta diferenciação através das cores é imprescindível para a orientação do usuário autor do material. No caso do *applet* que será usado pelo usuário aluno não há diferenciação de cores.

5.3.2.1 Os *buffers*

O JavaMusic utiliza três classes que atuam como *buffers* de dados ao serem armazenados em vetores (java.util.Vector). Estas classes encapsulam dados complexos sobre a exibição dos símbolos e contribuem para manter a unidade dos significados atribuídos a cada dado. Essas classes são:

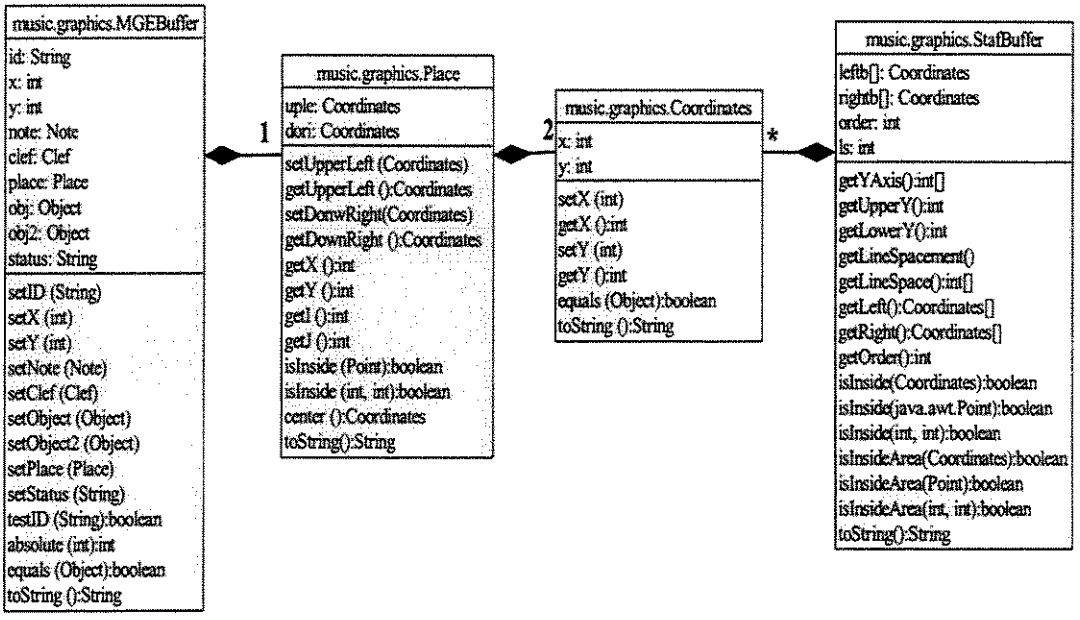


Figura 12: os *buffers*

music.graphics.StafBuffer:

Encapsula as coordenadas de cada linha de um pentagrama, o espaçamento entre as linhas e o número do pentagrama.

music.graphics.Coordinates:

Armazena as coordenadas x e y de um ponto.

music.graphics.Place:

Representa uma área quadrada encapsulando as coordenadas x e y do canto superior esquerdo e do canto inferior direito da área.

music.graphics.MGEBuffer:

Sendo o mais genérico e mais importante dentre todos os *buffers*, cada instância desta classe pode encapsular um objeto musical, de qualquer natureza, sua localização espacial na saída gráfica e seu *status* (*hide* ou *mute*).

5.3.2.2 Os pentagramas

Quando o JavaMusic é carregado pelo sistema, após a inicialização da interface gráfica, o programa gera automaticamente um pentagrama. O usuário pode alterar este procedimento selecionando a opção "*New*" no menu "*File*" ou na barra de ferramentas ou com o atalho "crtl + n". Com isso, uma caixa de diálogo é aberta e o usuário pode escolher a quantidade de pentagramas e a largura na faixa entre 300 e 600 *pixels*.

Considerando que a notação musical é baseada na inserção de símbolos sobre um sistema de cinco linhas chamado pauta ou pentagrama, o JavaMusic utiliza as informações sobre os pentagramas para posicionar corretamente cada símbolo musical a ser desenhado. Para desenhar um pentagrama é preciso criar um objeto GPentagram, da classe music.graphics.GPentagram. Este objeto pede como argumento o comprimento do pentagrama, a largura da margem esquerda, o espaçamento entre os pentagramas, a distância entre uma linha e outra e pode receber, também, um argumento booleano que, se for *true*, permitirá o ajuste automático do espaçamento entre os pentagramas. Quando o método drawStaf (java.awt.Graphics) é chamado, o objeto GPentagram desenha um pentagrama ou no topo da página ou imediatamente abaixo do último pentagrama e armazena informações sobre a posição de cada linha num objeto StafBuffer, da classe music.graphics.StafBuffer. Cada pentagrama é alocado em um objeto StafBuffer que, por sua vez, é alocado em um vetor (java.util.Vector) dentro do objeto GPentgram.

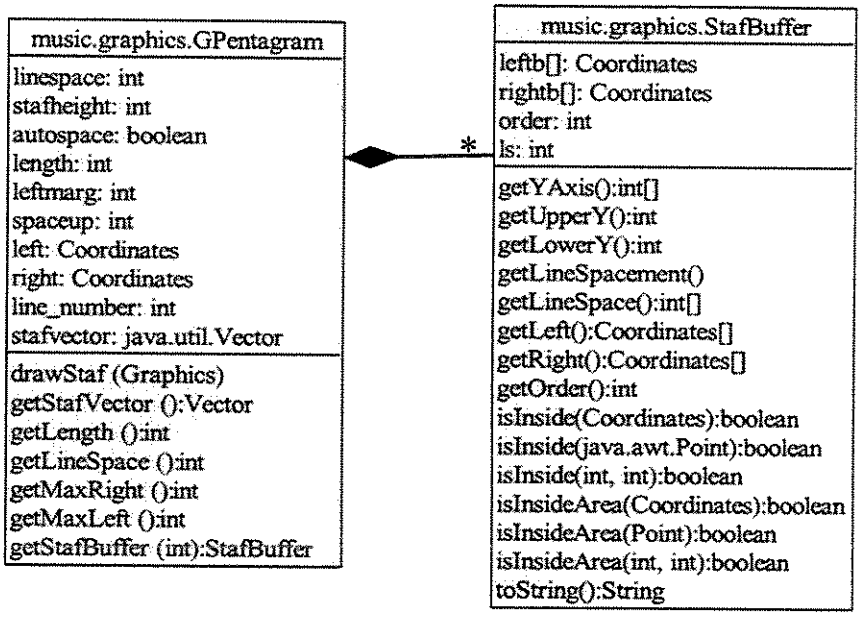


Figura 13: As classes GPentagram e StafBuffer

O método getStafBuffer(int i) retorna um objeto StafBuffer referente ao pentagrama especificado no argumento (i), sendo suas informações utilizadas por outras classes para posicionar graficamente os símbolos musicais.

5.3.2.3 As classes de significação musical

As classes de significação musical são classes cujos dados por elas encapsulados descrevem características relativas aos símbolos que representam conceitos musicais e/ou conceitos de notação musical.

music.music.MusicObject:

Interface herdada diretamente de java.io.Serializable. É o topo da hierarquia de todos as outras classes utilizadas pelo JavaMusic, sendo também a mais genérica de todas as classes.

music.music.Sound:

Classe abstrata da qual são herdadas as classes que representam os sons. Nela estão as constantes que definem os nomes e valores das notas musicais.

music.music.SoundNote:

Herdada da classe music.music.Sound, esta classe é uma abstração sobre o som das notas musicais.

music.music.Pitch:

Esta classe também é herdada de music.music.Sound e tem como membro principal um objeto SoundNote. Determina a nota exata em qualquer oitava da extensão do piano. Possui métodos para reconhecimento da armadura de clave (tonalidade), transposição de notas e o valor da nota dentro do padrão MIDI.

music.music.Note:

Das classes que representam os sons musicais, Note é a mais específica. Adota a representação das notas em três dimensões: altura, duração e intensidade. Possui métodos para a transposição de notas, alteração na duração, variação na dinâmica, reconhecimento de sinais de alteração e armadura de clave.

music.music.Time:

Classe abstrata que serve de referência para as demais classes que representam abstrações do tempo e da duração dos sons na música.

music.music.Duration:

Encapsula o valor de duração de uma nota musical.

music.music.TimeSignature:

Encapsula informações sobre a fórmula de compasso.

music.music.Tempo:

Encapsula as informações sobre o andamento da música.

music.music.BarLine:

Interface que contém as variáveis que representam as barras de separação entre compasso.

music.music.MObject:

Classe abstrata da qual são estendidas as classes que encapsulam informações sobre as alterações na escrita musical. Estas alterações abrangem acidentes, claves, escalas e armaduras de clave.

music.music.Sign:

Esta classe serve de ponteiro de referência à presença de sinais de alteração nas notas.

music.music.KeySignature:

Encapsula as armaduras de clave.

music.music.Scale:

Classe na qual são definidas todas as escalas tonais.

music.music.Clef:

Encapsula uma clave musical.

music.music.Intension:

Desta classe são geradas as classes cujas funções são representar variações de intenção na execução da música. Nesta versão, foram implementadas classes para articulação e dinâmica.]

music.music.Articulation:

É usada para definir a articulação de uma nota através de parâmetros que determinam o quanto o ataque (início do som) e a terminação de uma nota deverá ser atrasado ou adiantado, possibilitando o encavalamento (ligadura) ou separação (staccato) entre as notas.

music.music.Dinamic:

Encapsula a intensidade de uma nota, podendo conter tanto um valor único (*velocity*) quanto uma *array* de valores (variações de intensidade sobre uma mesma nota musical).

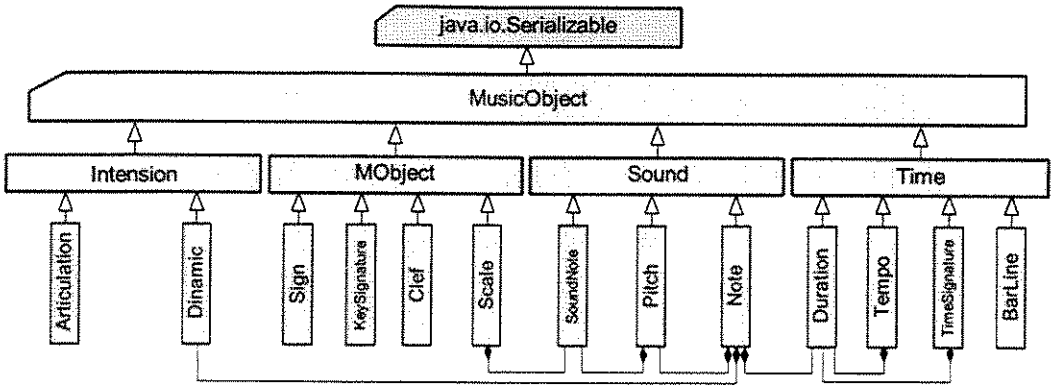


Figura 14: Classes de significação musical

5.3.2.4 As classes de representação gráfica

As classes de representação gráfica são classes que contêm métodos para desenhar os símbolos musicais.

O desenho de cada símbolo é feito pela sucessão de vetores (segmentos de reta) formando as figuras. A referência de medida de cada vetor é feita em função da distância entre duas linhas de um pentagrama. Assim, para desenhar uma figura musical, é preciso informar o valor em *pixels* desta distância, definido na variável linespace da classe GPentagram.

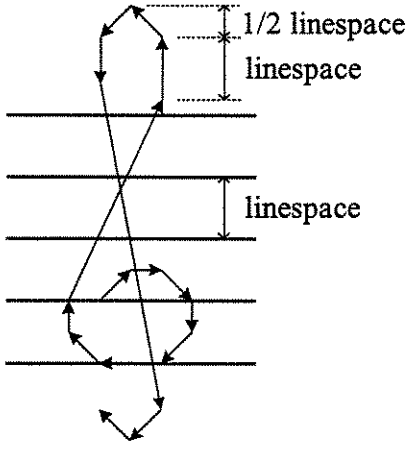


Figura 15: Vetores foram usados para formar uma clave de Sol tendo linespace como referência

As classes de representação gráfica precisam conhecer o pentagrama onde deverá desenhar cada símbolo ou as coordenadas x e y da posição da tela, saber se a classe está sendo chamada a partir do aplicativo ou do *applet* e se o símbolo terá estado *hide* (escondido), *mute* (sem som) ou ambos.

As informações sobre o tamanho e a posição onde a figura deve ser desenhada podem ser passadas de duas formas: para desenhar um símbolo numa posição qualquer na saída gráfica, basta informar os valores das coordenadas x e y da posição e a medida referência de tamanho; para desenhar o símbolo em um determinado pentagrama, basta passar como argumento para o método de desenho da classe o objeto GPentagram contendo os pentagramas usados, o número do pentagrama e a posição horizontal em que se deseja desenhar o símbolo. O objeto GPentagram permite a classe de representação gráfica conhecer a altura (posição vertical ou coordenada y) e a variável linespace (referência de tamanho para os vetores).

Quando o usuário selecionar um botão da interface gráfica para selecionar um determinado símbolo e, em seguida, clicar numa posição qualquer de um pentagrama, a interface de controle automaticamente chama o método de desenho da classe gráfica referente àquele símbolo e passa os argumentos devidos. A classe gráfica por sua vez reconhece em qual pentagrama deverá desenhar, qual a distância entre suas linhas e a posição em que o *mouse* foi clicado, renderizando a imagem do símbolo e em seguida retornando um objeto Place contendo a área real onde o símbolo foi desenhado. Neste momento, é criado um objeto MGEBuffer que encapsula a posição do símbolo na tela, o objeto musical por ele representado e o objeto Place. Este objeto Place, gerado pela classe gráfica, é usado pela interface de controle para cruzar as coordenadas do ponteiro do *mouse* com a posição real do símbolo na tela quando o usuário precisar executar qualquer ação sobre ele. O objeto MGEBuffer, por sua vez, é armazenado num vetor de objetos, onde poderá ser consultado pelo sistema a qualquer tempo. O processo de renderização das figuras tanto a partir da ação do usuário quanto a partir da leitura de um arquivo XML, será mostrado com detalhes na seção "Inserindo os símbolos musicais no pentagrama". Agora vamos conhecer as classes de representação gráfica:

music.graphics.GClef

As claves são desenhadas pela classe music.graphics.GClef. Esta classe, que não precisa ser instanciada por tratar-se de uma classe estática, contém métodos que possibilitam desenhar as claves em condições diversas, de acordo com os argumentos informados.

Para desenhar uma clave numa posição específica da tela, usa-se o método draw passando como argumento o dispositivo gráfico usado (java.awt.Graphics), o nome da clave, as coordenadas x e y onde se deseja que a clave seja renderizada e o tamanho em *pixels*[34].

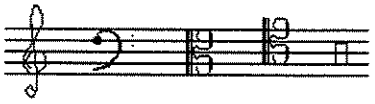


Figura 16: exemplo de claves

Para desenhar uma clave sobre um determinado pentagrama é preciso passar ao método draw, além do dispositivo gráfico e do nome da clave, também o objeto GPentagram, o número do pentagrama e a posição na linha horizontal (distância em *pixels* da borda esquerda da janela de visualização) onde a clave deverá ser desenhada, uma string chamada de *status* que poderá conter as letras "h", "m" ou "hm", que significam respectivamente "*hide*", "*mute*" e "*hide and mute*" e um valor booleano que indicará se a saída gráfica ocorre no aplicativo ou no *applet*.

music.graphics.GNote:

Mais complexa de todas as classes de representação gráfica da API JavaMusic, a classe GNote oferece uma grande variedade de opções ao método draw(), permitindo o máximo de flexibilidade para o desenho das figuras das notas.

É possível desenhar uma figura de nota em qualquer posição da tela indicando-se a saída gráfica, um objeto Duration com o tipo de figura desejado, as coordenadas x e y e o tamanho desejado.

Para desenhar uma nota sobre um pentagrama é preciso fornecer: a saída gráfica, o objeto GPentagram, o objeto Note, um objeto Sign com o sinal da nota (bemol, sustenido ou bequadro), o objeto Clef indicando qual clave está sendo usada naquele trecho da música, a posição horizontal (em vez das coordenadas x e y), o número do pentagrama, o *status* e a indicação de *applet*.

---

[34] É importante observar que o tamanho aqui referido não é o tamanho da figura em si, mas sua proporção com a distância entre duas linhas consecutivas de um suposto pentagrama, uma vez que essa distância é a medida de referência para a dimensão das figuras musicais. Essa relação de dependência entre as figuras e as linhas do pentagrama assegura que todas as figuras serão renderizadas numa dimensão tal que se ajustem as proporções do pentagrama.

Figura 17: figuras de notas

É importante observar que, apesar do objeto Note ser capaz de encapsular a altura real da nota musical, seja ela natural, bemol ou sustenido, a classe Note representa as notas do ponto de vista do seu entendimento musical e não sua notação gráfica. Assim, quando deseja-se imprimir um sinal de sustenido ou bemol junto com a nota, este sinal tem que ser explicitamente solicitado à classe GNote pelo envio do objeto Sign como um dos argumentos do método draw() pois o JavaMusic separa o significado musical da sua representação gráfica.

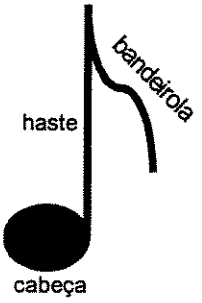


Figura 18: nomenclatura dos elementos de uma nota musical

A classe GNote implementa um algoritmo que identifica e decide se a haste de uma nota deverá ser voltada para cima ou para baixo.

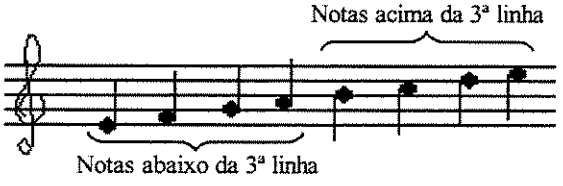


Figura 19: o GNote desenha as hastes voltadas para cima nas notas colocadas abaixo da terceira linha e desenha as hastes para baixo nas notas a partir da terceira linha do pentagrama

Também possui um método projetado para desenhar grupos de notas em que, notas, claves, sinais, posição horizontal e status, são passados em *arrays*. Estes *arrays* são processados para que cada elemento seja desenhado separadamente, exceto as bandeirolas, para as quais um algoritmo calcula a melhor disposição para a correta ligação dos grupos de notas.

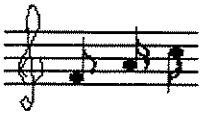


Figura 20: notas não agrupadas

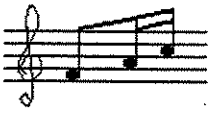


Figura 21: notas agrupadas

Este algoritmo é capaz de agrupar qualquer conjunto de notas, mesmo com símbolos como pausas, sinais de alteração ou claves intercalados às notas.

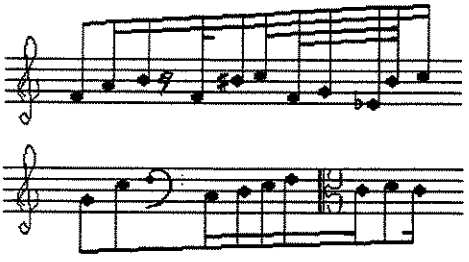


Figura 22: grupos de notas com outros símbolos intercalados

music.graphics.GPause:

Esta classe é responsável pelo desenho das figuras de pausa. Para desenhar uma pausa existem dois modos: primeiro, usando o dispositivo gráfico, a duração da pausa (objeto Duration), as coordenadas x e y e o tamanho, pode-se desenhar a pausa em qualquer posição da saída gráfica; segundo, usando o dispositivo gráfico, um objeto GPentagram, a duração da pausa, o número do pentagrama, a posição horizontal, o *status* e a indicação de *applet*, pode-se desenhar num pentagrama.

Figura 23: figuras das pausas

music.graphics.Gbarline:

Classe usada para desenhar as barras de compasso. Devem ser passados os seguintes argumentos para seu método draw(): o dispositivo gráfico em uso (java.awt.Graphics), o objeto GPentagram, o tipo de barra de compasso, o número do pentagrama, a posição horizontal, o *status* (*hide* e/ou *mute*) e um *boolean* para indicar se a chamada da classe foi feita a partir de um *applet* (*true*) ou do aplicativo (*false*).

Os valores referentes ao tipo de barra de compasso estão definidos na classe music.music.BarLine pelas constantes:

```
public static final int BARLINE_BEGUIN = 0;
public static final int BARLINE_END = 1;
public static final int BARLINE_SINGLE = 2;
public static final int BARLINE_DOUBLE = 3;
public static final int BARLINE_LEFTREPEAT = 4;
public static final int BARLINE_RIGHTREPEAT = 5;
public static final int BARLINE_DOUBLEREPEAT = 6;
public static final int BARLINE_ENDSCORE = 7;
```

Figura 24: barras de separação de compasso

music.graphics.GText:

Esta classe é usada para inserir textos dentro da partitura. Os textos são enviados a esta classe no formato de cordões (*string*) e juntamente com o dispositivo gráfico e as coordenadas x e y da posição inicial pode-se inserir textos em qualquer lugar da saída gráfica.

Os textos são impressos, por padrão, em tipos "TimesRoman", normal, preto e fonte 12, porém, a classe GText reconhece alguns comandos capazes de alterar a formatação do texto. Estes comandos não estão implementados na interface do usuário do JavaMusic em modo gráfico, nem foram incluídos no código XML, mas podem ser acessados através da simples inclusão dos comandos antes do texto desejado, separando por ponto e vírgula. Os comando disponíveis são:

- **c = xxx**: para alterar as cores, onde x pode ser "1" ou "0" representando as cores: preto (000), azul (001), ciano (011), verde (010), vermelho (100), amarelo (110), magenta (101) e cinza (111);
- **s = x**: para alterar o tamanho da fonte, onde x é o tamanho desejado em paica;
- **b**: para tornar a fonte negrita;
- **i**: para tornar a fonte itálica;
- **f = nome**: para alterar o tipo de fonte, onde nome é o nome da fonte desejada.

andante
andante andante **andante** *andante*
andante

Figura 25: exemplos de textos reformatados

O recurso de alteração das fontes não foi incluído na interface de comandos do JavaMusic, pois foram implementados quando os trabalhos de desenvolvimento do JavaMusic já estavam finalizados. Deverá ser incluído nas versões futuras do programa.

music.graphics.GTimesign:

Classe que desenha as indicações de fórmula de compasso e de alteração de andamento (music.music.Tempo).

Para inserir uma indicação de tempo, basta informar ao método draw() da classe GTimesign o dispositivo gráfico, o objeto Tempo desejado, as coordenadas x e y, o *status* e a indicação de *applet*.

Para inserir uma armadura de clave, substitui-se o objeto Tempo pelo objeto TimeSignature, e para inseri-lo num pentagrama, é preciso passar o objeto GPentagram, o número do pentagrama e a posição horizontal.

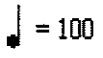


Figura 26: indicação de tempo (andamento)

Figura 27: fórmulas de compasso

music.graphics.GKeysign:

Desenha a armadura de clave (indicação da tonalidade pela justaposição de bemóis ou sustenidos), tomando como argumento: o dispositivo gráfico, o objeto GPentagram, o objeto Clef

indicando qual clave está sendo usada naquele trecho da música, o objeto KeySignature com a armadura de clave desejada, um objeto KeySignature indicando qual era a armadura de clave anterior, a posição horizontal, o número do pentagrama, o *status* e a indicação de *applet* ou aplicativo.

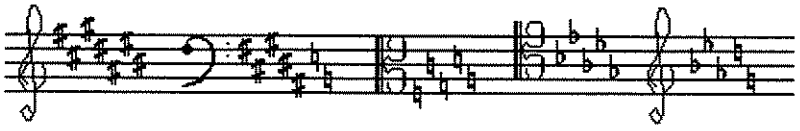


Figura 28: armaduras de clave

music.graphics.GTuplet:

A classe GTuplet faz a indicação gráfica da quiáltera. Ela desenha uma barra horizontal sobre o grupo de notas que constituirá a quiáltera (vide figura) e insere o valor de equivalência das durações das notas sobre a barra.

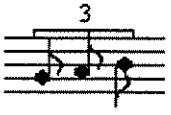


Figura 29: quiáltera

Pede como argumento um objeto Place que corresponde à área ocupada pelo grupo de notas e o valor da equivalência, além do dispositivo gráfico, *status* e indicação de *applet*.

5.3.2.5 Inserindo os símbolos musicais no pentagrama

Quando o usuário clica numa linha de um pentagrama para inserir uma nota ou qualquer outro símbolo, é desencadeada uma seqüência de ações dentro da interface de controle.

A primeira ação realizada pelo sistema é reconhecer qual botão foi clicado e, logo após, a posição do ponteiro do *mouse*. Ao clicar num dos botões da interface gráfica, o nome do botão é armazenado na variável lastbutton da classe music.Music. Ao clicar num dos pentagramas, é gerado um evento mouseClicked que encapsula as coordenadas do ponteiro do *mouse*, o número de cliques e qual botão (direito ou esquerdo) do *mouse* foi usado.

O método clickedOn() da classe Music é então chamado para processar estes eventos.

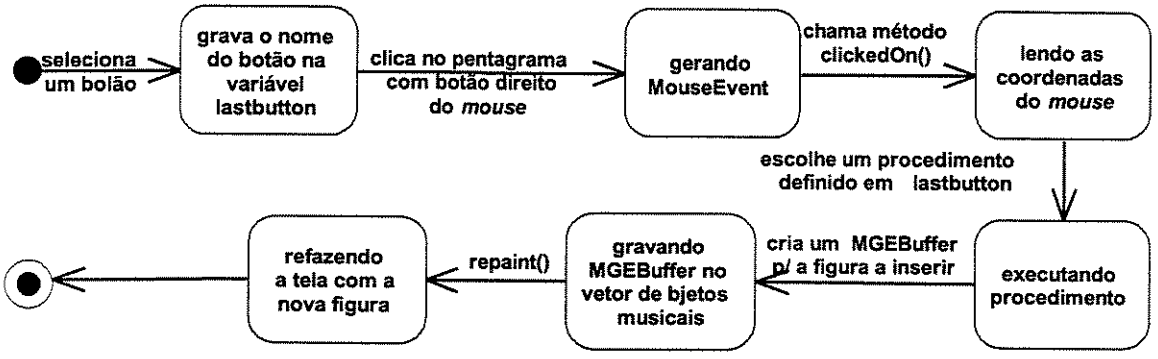


Figura 30: Inserção de uma figura

O método clickedOn() recebe como argumento três informações: a posição do ponteiro do *mouse*, o botão usado e o número de toques no botão. Inicializada suas variáveis locais, este método verifica o conteúdo da variável global lastbutton para saber qual ação deverá executar, então executa a ação que consiste em criar um objeto MGEBuffer encapsulando a figura que deseja inserir e, em seguida, armazena este objeto no vetor de objetos musicais. As ações podem ser:

Inserção de pausas:

Usa o método pauseTest(lastbutton) para selecionar a ação de inserir uma pausa e o método defineDuration(lastbutton) para verificar qual a duração da pausa escolhida e criar um objeto MGEBuffer.

Inserção de claves:

Usa o método clefTest(lastbutton) para selecionar a ação de inserir uma clave e o método defineClef(lastbutton) para verificar qual a clave escolhida.

Inserção de barras de compasso:

Usa o método barlineTest(lastbutton) para selecionar a ação de inserir uma barra de compasso e o método defineBarline(lastbutton) para verificar qual a barra escolhida.

Inserção de fórmula de compasso:

Se o conteúdo de lastbutton for uma *string* "timesign", então, é selecionada a ação de inserir uma fórmula de compasso. Neste caso, quando o usuário clica no botão "TIME", é aberta uma caixa de

diálogo na qual pode optar por definir a fórmula de compasso ou o andamento. Ao escolher fórmula de compasso, é criada uma instância de TimeSignature que fica armazenada na variável global timesignature da classe Music. Essa variável é usada para criar o objeto MGEBuffer com a fórmula de compasso escolhida.

Inserção de alteração de andamento:

Se o conteúdo de lastbutton for uma *string* "tempo", então, é selecionada a ação de inserir uma alteração de andamento. Neste caso, quando o usuário clica no botão "TIME", é aberta uma caixa de diálogo onde pode optar pela alteração de andamento, criando uma instância da classe Tempo que fica armazenada na variável global tempo da classe Music. Essa variável é usada para criar o objeto MGEBuffer com a alteração de andamento escolhida.

Inserção de textos:

Aqui também é usada uma caixa de diálogo onde o usuário entra com o texto. Ao clicar em "OK", o conteúdo digitado é armazenado na variável global textfield, da classe Music, que, por sua vez, é usada para criar um objeto MGEBuffer contendo o texto desejado.

Inserção de notas:

Esta função é uma das mais complexas de toda a interface de controle do JavaMusic, pois o programa precisa cruzar três informações diferentes: a posição do ponteiro do *mouse*, onde se deseja inserir a nota, a clave mais próxima a esta posição e as coordenadas do pentagrama.

O método clickedOn se serve de uma variedade de outros métodos para executar esta ação, sendo que inicialmente chama o método noteTest(lastbutton) para saber que o usuário pretende inserir uma nota musical.

A seguir chama o método searchLastClef e lhe passa como argumento o número do pentagrama e a posição horizontal onde ocorreu o clique do *mouse*. Com estes dados, o método searchLastClef faz uma varredura no vetor de objetos musicais (MGEBuffer) e procura identificar qual foi a última clave usada na partitura. O vetor de objetos musicais armazena todos os MGEBuffers em ordem espacial, da esquerda para a direita e de cima para baixo. Assim, para o sistema saber qual a última clave usada, basta procurar (da última posição do vetor para a primeira) um MGEBuffer contendo uma clave cuja localização (coordenadas x e y) seja mais à esquerda ou mais acima do ponto onde se deseja inserir uma nota.

O próximo passo para a inserção da nota é identificar qual é a nota em si, uma vez que, até este momento, o sistema conhece apenas sua posição espacial, mas não sabe que nota aquela posição representa. Para isso, o sistema precisa fazer o cruzamento entre a posição espacial das linhas do pentagrama, as coordenadas do ponteiro do *mouse* e a clave que está sendo usada. É chamado, então o método searchPitch que recebe como argumento um *array* contendo a altura em *pixels* de cada linha do pentagrama, a clave encontrada por searchLastClef e a altura do ponteiro do *mouse*. Do cruzamento destes três dados é retornado, então, um objeto Pitch que representa a altura da nota.

Identificada a altura da nota, o método defineDuration usa a variável lastbutton para criar um objeto Duration correspondente a duração da nota.

Para a criação do MGEBuffer correspondente à nota a ser inserida, ainda são coletadas as variáveis start e end para definir a articulação da nota, a variável velocity para definir a intensidade da nota e a variável signal para definir o sinal da nota. Também são chamados os métodos refreshGroups e refreshQuialtera para atualizar as notas agrupadas e as quiálteras respectivamente.

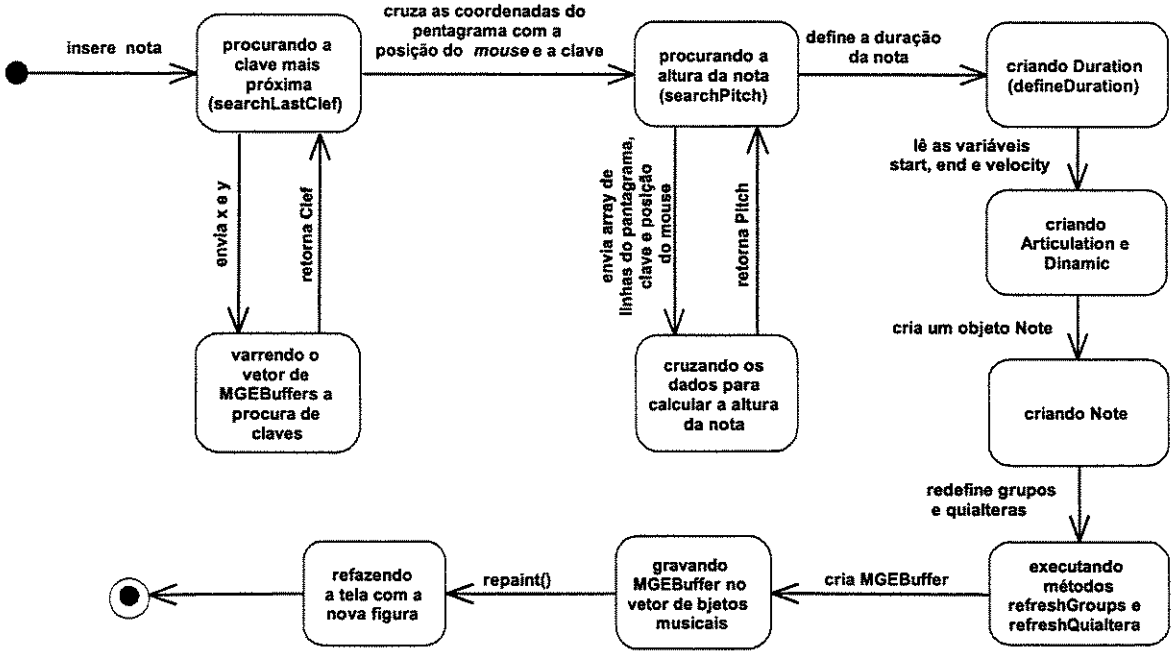


Figura 31: Inserção de notas

Inserção de armaduras de clave

A inserção de uma armadura de clave é reconhecida pelo método keySignTest(lastbutton) chamado por clickedOn.

Inicialmente o método searchLastClef é chamado para identificar qual a última clave usada. Em seguida, o método searchPitch é chamado para se obter a altura correta para cada sinal da armadura de clave e defineKeySign(lastbutton) é, então, invocado para construir o objeto KeySignature referente à armadura desejada.

Antes de armazenar o KeySignature no MGEBuffer e este, por sua vez, no vetor de objetos musicais, é feita uma varredura no vetor de objetos musicais para procurar a armadura de clave imediatamente anterior e a imediatamente subseqüente para efetuar eventuais correções na tonalidade apresentada pela armadura que se deseja inserir.

5.3.2.6 Deletando símbolos musicais

O processo de deletar figuras é também realizado no método clickedOn da classe Music, porém, a diferença está no uso do botão direito do *mouse*.

Quando o botão direito do *mouse* é identificado pelo método clickedOn, uma sub-rotina identifica a posição do *mouse* e faz uma varredura no vetor de objetos musicais procurando um MGEBuffer contendo um objeto Place que represente uma área que abranja as coordenadas do ponteiro do mouse. Então, este objeto MGEBuffer é apagado do vetor de objetos musicais.

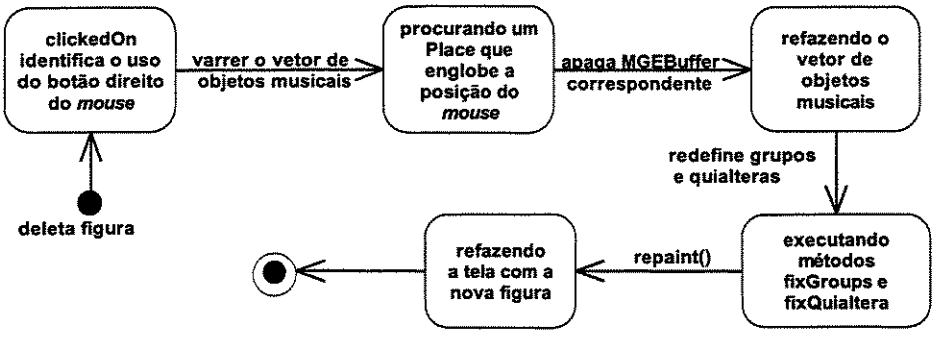


Figura 32: Deleção de figuras

O método clickedOn da classe Music é responsável tanto por inserir novos símbolos quanto por apagar os símbolos não desejados no pentagrama. Este método é usado somente pelo aplicativo pois,

no caso do *applet*, seu uso não faz sentido se considerarmos que nesta versão do JavaMusic o *applet* é usado apenas para a visualização das partituras e não para sua edição.

A saída gráfica é feita através do método paint() tanto no aplicativo (classe Music) quanto no *applet* (classe MusicPanel) que, como veremos detalhadamente na próxima seção, "lê" o conteúdo do vetor de objetos musicais e desenha um a um usando as classes de representação gráficas já comentadas. O vetor de objetos musicais é uma variável global do tipo java.util.Vector denominada elements.

O diagrama abaixo mostra um resumo do funcionamento do método clickedOn().

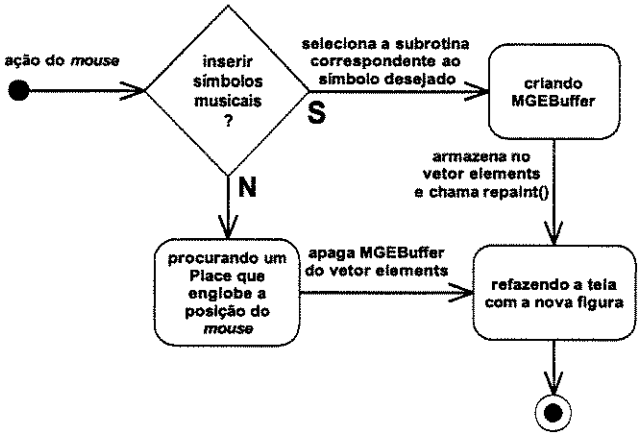


Figura 33: Método clickedOn()

5.3.2.7 Gerando a saída gráfica

Para gerar a saída gráfica, ou seja, para imprimir na tela os símbolos musicais, pentagramas e textos, é usado o método paint (java.awt.Graphics), padrão da linguagem Java.

Este método usa um laço para fazer a leitura dos elementos gravados na variável elements, que é usada tanto no aplicativo (classe Music) quanto no *applet* (classe MusicPanel) e consiste em uma instância da classe java.util.Vector.

Cada elemento encontrado no vetor elements é um objeto MGEBuffer que encapsula um objeto ou símbolo musical contendo as seguintes informações: uma instância do objeto musical, as coordenadas da tela onde ele será desenhado, o status, a clave anterior, os sinais de alteração das notas musicais e a área ocupada pela figura (Place).

Quando uma figura é inserida, o método clickedOn gera um MGEBuffer correspondente à figura. Porém, o campo Place do MGEBuffer permanece vazio até que a figura seja renderizada, momento em que sua área é calculada e o MGEBuffer é atualizado.

O método paint() também tem a função de identificar as marcações de grupos de notas ou de quiálteras para gerar a impressão correta.

A escolha da classe de representação gráfica responsável pela impressão de cada figura é feita por uma sucessão de comandos "*if*" que testam o conteúdo do MGEBuffer e, tendo encontrado o procedimento correto, desmembram os dados contidos no *buffer* e os fornecem como argumento às classes de representação gráfica. Estas, por suas vezes, renderizam as imagens e retornam um objeto Place contendo a área ocupada pela figura. O último passo do método paint é atualizar o MGEBuffer e retorná-lo ao vetor de objetos.

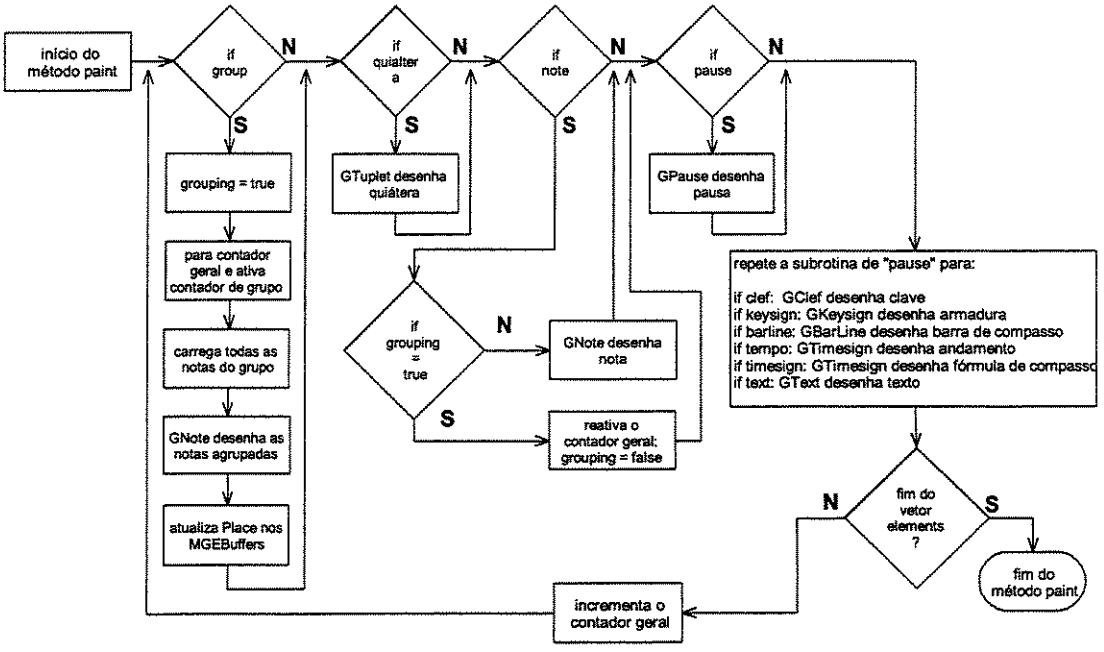


Figura 34: Fluxograma resumindo método paint

5.3.2.8 Selecionando elementos na tela

A interface gráfica do aplicativo JavaMusic possui um botão usado para ativar o modo de seleção de objetos. Quando o botão de seleção está ativado, somente os eventos mouseMoved e mouseDragged são reconhecidos pela interface de controle.

Ao clicar e arrastar o *mouse* sobre uma figura da partitura, um retângulo vermelho é desenhado sobre a figura indicando a área selecionada. Diversas ações podem acontecer sobre a seleção:

Cortar, copiar e colar:

Ao selecionar um ou mais elementos da partitura e clicar no botão copiar, o método copy da classe music.Music cria um objeto Place representando a área selecionada e faz uma varredura no vetor de objetos musicais a procura dos objetos cujo posicionamento na saída gráfica coincida com a área selecionada. Os objeto encontrados são copiados num vetor temporário.

Selecionando uma área da tela onde se deseja inserir os elementos copiados e clicando no botão colar, o método paste da classe music.Music atualiza as coordenadas de cada objeto selecionado e armazena estes objetos no vetor de objetos musicais. Quando o método paint é invocado para refazer a tela, os elementos selecionados aparecem na nova posição.

A diferença no tratamento dos métodos cut e copy é que em cut, os objetos selecionados são removidos do vetor de objetos musicais e transferidos para o vetor temporário e, no método copy, são apenas copiados.

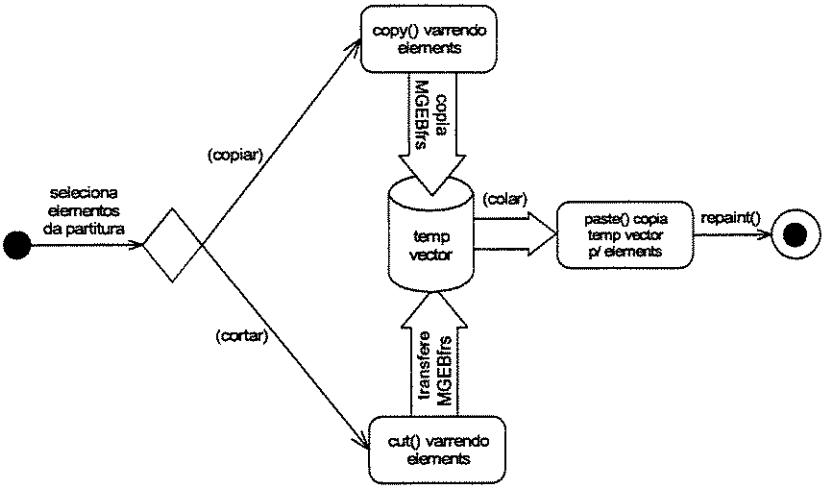


Figura 35: diagrama de atividades para cut, copy e paste

"hide" e "mute":

A função *hide* foi criada para permitir que o usuário possa fazer com que determinados elementos da partitura não apareçam na tela. Esta função poderá ser usada na preparação de ditados musicais: o aluno ao ver a partitura através do *browser* (e do *applet*) não poderá ver todos os elementos escritos, apesar deles estarem presentes e serem inclusive tocados pelo *applet*.

A função *mute* foi criada para impedir que determinadas notas musicais (ou mesmo elementos musicais como sinais de alteração, mudanças de andamento, etc.) possam ser tocados.

O objetivo desta função é permitir ao autor a criação de exercícios de leitura onde o aluno poderá ver a partitura, porém, não poderá fazer tocar todos os elementos nela presentes e deverá então supor o seu significado entoando-os ou tocando-os num instrumento qualquer.

As duas funções recebem um tratamento diferenciado no que se refere à apresentação na tela, dependendo de onde estão sendo visualizadas. Quando uma nota musical recebe o status de *hide* e está sendo usado o aplicativo JavaMusic, esta nota é desenhada na cor cinza, para que o autor que esteja editando a partitura possa reconhecer tal nota como escondida. Quando esta partitura for visualizada através do *applet*, a nota escondida será desenhada na mesma cor do fundo ficando, portanto, invisível. Isso explica porque as classes de representação gráfica precisam receber entre os parâmetros um argumento informando se a saída gráfica é para o aplicativo ou para o *applet*.

No caso da função *mute*, as notas marcadas com esse status aparecem em azul no aplicativo e em preto no *applet* (isso porque o aluno não precisa saber quais notas não serão tocadas pois ele deverá perceber isso auditivamente). Se um objeto musical for marcado com *hide* e *mute*, ele aparecerá na cor ciano se visto no aplicativo e invisível no *applet*.

Ao selecionar com o mouse um ou mais elementos da tela e clicar no botão *hide* ou *mute*, os métodos hide e mute da classe music.Music fazem uma varredura no vetor de elementos musicais e selecionam todos os elementos cuja posição na tela esteja dentro da área selecionada. Em seguida, altera a variável status de cada um dos MGEBuffers encontrados no vetor para "h" ou "m" ou "hm".

Quando os objetos selecionados já estão com status *mute* ou *hide*, o status é alterado para normal.

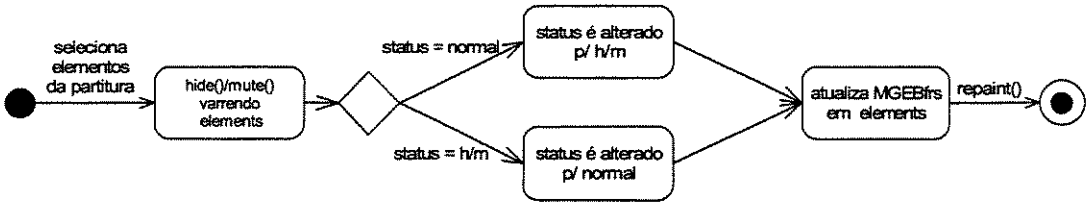


Figura 36: diagrama de atividades para os métodos hide e mute

5.3.3 Processador de XML

A interface responsável pela geração do arquivo XML e pela interpretação deste é composta por duas classes:

- **music.util.Parse:** sua função é extrair do documento XML o conteúdo de cada marca e de cada parâmetro.
- **music.util.XMLProcessor:** classe que possui um método para escrever o XML a partir do vetor de objetos musicais e outro que reconstrói o vetor usando o conteúdo extraído do documento XML.

Estas classes foram criadas para gerar e ler os arquivos XML na fase de teste das demais classes que compõem o JavaMusic. Elas seriam, a princípio, substituídas pela API SAX [SAX] no decorrer do trabalho de desenvolvimento. No entanto, observou-se que estas duas classes funcionaram com eficiência o bastante para permanecerem nesta implementação. Apesar das classes XmlProcessor e Parse não possibilitarem a validação do código XML com sua DTD, a princípio, os arquivos XML gerados pelo JavaMusic não devem ser manipulados diretamente pelo usuário e a classe XMLProcessor garante a construção do código sem nenhuma inconsistência em relação a DTD.

Testes efetuados com o uso do *software* XML Spy [SPY] demonstraram total confiabilidade no código gerado pelo JavaMusic.

5.3.3.1 A classe music.util.Parse

Um objeto Parse é criado a partir de um *string* contendo todo o conteúdo do documento XML.

Quando a classe Parse é instanciada, esta cria um vetor de *strings* onde cada elemento corresponde a uma marca do documento XML.

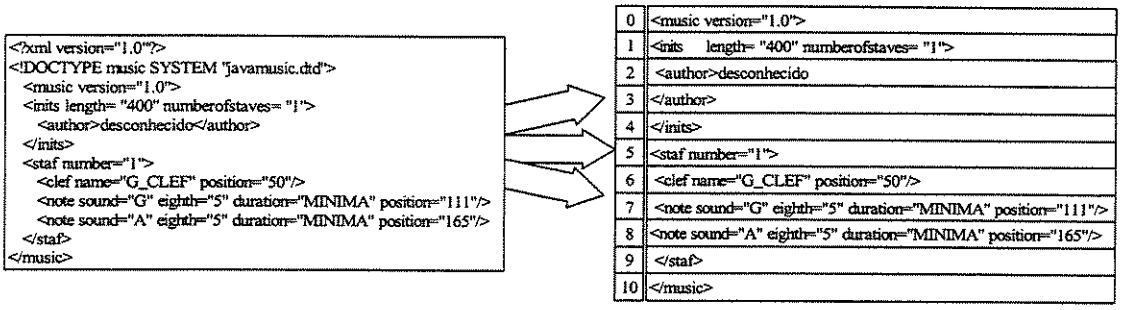


Figura 37:as marcas do XML são copiadas num objeto java.util.Vector

Um contador interno é usado para garantir a leitura dos elementos do vetor na ordem em que aparecem no XML original. Porém, o contador pode ser paralisado para extrair todos os parâmetros de uma marca contida em um dos elementos do vetor.

Um objeto Parse também pode ser criado a partir de um vetor contento em cada elemento uma marca do XML. Assim, é possível construir uma árvore hierárquica para as marcas do XML.

Foram criados os seguintes métodos para extrair o conteúdo das marcas:

- **public String getTag():** Retorna o nome da marca da posição corrente do contador;
- **public String getContentOfTagS(String tag):** retorna o conteúdo da marca "tag";
- **public String getDataOf (String tag):** retorna o conteúdo de todas as marcas contidas pela marca "tag";
- **public Vector getContentOfTag (String tag):** retorna o conteúdo de todas as marcas contidas pela marca "tag" distribuído dentro de um objeto Vector. Com esse vetor é possível gerar um novo objeto Parse para extrair o conteúdo de suas marcas;
- **public Vector getParam (String tag):** retorna a lista de parâmetros da marca "tag";
- **public String getAttribute (String tag, String param):** retorna o valor do parâmetro "param" da marca "tag";
- **public boolean hasMoreElements ():** retorna "*true*" se ainda houverem marcas a serem lidas;
- **public boolean isEndTag (String s):** retorna "*true*" se a próxima marca for uma marca de fechamento (ex.: </marca>).

5.3.3.2 A classe music.util.XmlProcessor

A classe XmlProcessor possui somente dois métodos:

- **public static StringBuffer XmlGenerator(Vector music, String name, String author, int length, int amount):** escreve o documento XML a partir do vetor de objetos musicais. Pede ainda como argumento o nome da obra, o nome do autor, a largura dos pentagramas em *pixels* e a quantidade de pentagramas. Retorna um objeto java.lang.StringBuffer contendo o documento XML pronto;
- **public static Object[] getMusic(String m):** interpreta o arquivo XML contido no argumento "m" criando um objeto Parse para extrair o conteúdo de todas as marcas. Retorna um *array* com cinco objetos contendo em cada posição: nome da obra, autor, quantidade de pentagramas, largura em *pixels* e um objeto Vector que corresponde ao vetor de objetos musicais.

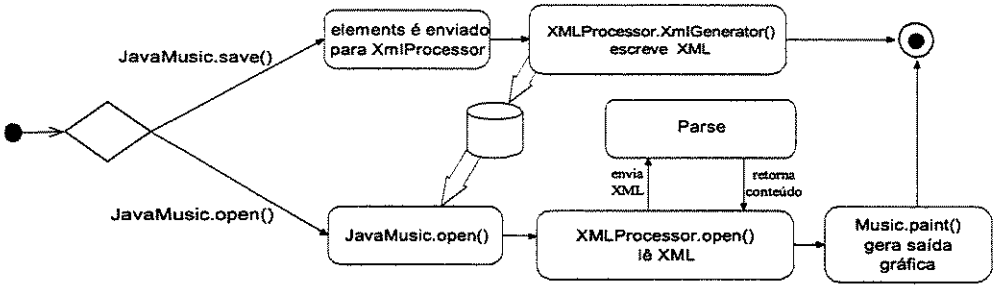


Figura 38: XmlProcessor e Parse é usado para ler e gerar os arquivos XML

5.3.4 Interface MIDI

Duas classes foram criadas para processar eventos MIDI gerando a saída de áudio e arquivos .mid. São elas a: interface music.midi.Playable e a classe music.midi.Player.

A interface Playable define três métodos que são implementados em Player: play(), saveAsMidi() e stop().

Quando um objeto Player é instanciado, seu construtor pede uma cópia do vetor de elementos musicais. O método play() é chamado para executar a seqüência de elementos musicais armazenados no

vetor. O método makeSequence() é responsável pela leitura de cada elemento do vetor, desencapsulando o conteúdo de cada MGEBuffer e alocando-o em um vetor de eventos MIDI. A seguir o método play() invoca o método createEvent(), responsável por gerar os eventos MIDI propriamente ditos e enviá-los para uma trilha (classe javax.sound.midi.Track). O método sequencer.start() aciona o seqüenciador padrão do sistema e a música começa a tocar.

Quando a música chaga ao final, isto é, quando o último evento MIDI da seqüência é tocado, a trilha contendo a seqüência MIDI é automaticamente descartada da memória. Se for desejado interromper a execução antes do seu término, o método stop() deve ser invocado para que a seqüência seja interrompida.

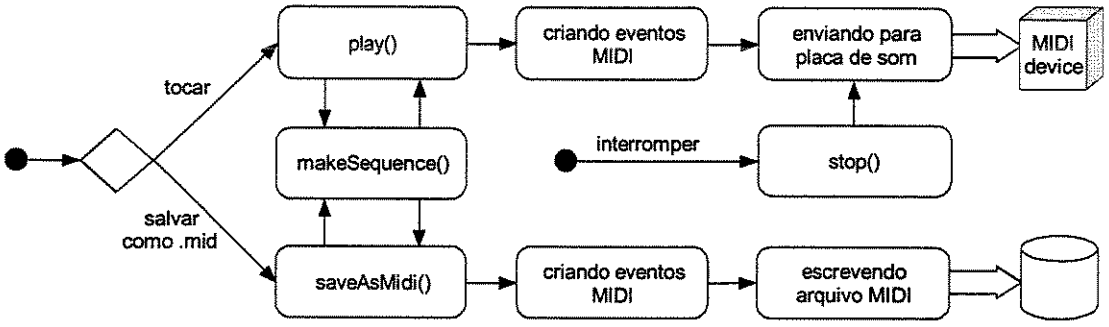


Figura 39: classe music.midi.Player

O método saveAsMidi() é similar ao método play(), exceto que, ao invés de usar o método sequencer.start() para tocar a seqüência MIDI, chama o método MidiSystem.write() para escrever a seqüência num arquivo .mid.

5.3.5 Entrada e saída

O mecanismo de entrada e saída foi implementado na classe music.util.Record.
Foram definidos os seguintes métodos:

- **public void setArchive(File name):** define um objeto java.io.File referente ao arquivo que se deseja escrever ou abrir;
- **public File getArchive():** recupera o arquivo aberto;

- **public void setDocument(StringBuffer doc):** define um objeto java.lang.StringBuffer contendo o documento XML;
- **public StringBuffer getDocument():** resgata um objeto java.lang.StringBuffer contendo o documento XML que se deseja abrir;
- **public void openFile():** abre o arquivo definido pelo método setArquive;
- **public void saveFile():** salva o conteúdo definido por setDocument no arquivo definido por setArchive.

A idéia de centralizar todos os recursos de escrita e leitura numa mesma classe permitiu padronizar todas as ações que envolvem o acesso ao disco rígido e simplificar a estrutura de todo o programa.

5.4 O applet JM

As classes music.JavaMusic e music.MusicPanel são usadas na arquitetura do *applet* JM.

O *applet* JM desempenha uma função muito importante na estrutura do JavaMusic. Ele é a ferramenta que permite a visualização dos arquivos XML em forma de partitura na *web*. Sem ele, a ferramenta JavaMusic não faria sentido, pois seu objetivo maior é a criação de material musical para a Internet.

O funcionamento das classes que compõem este *applet* é muito similar ao funcionamento do aplicativo JavaMusic. Na verdade, as classes music.JavaMusic e music.Music foram adaptadas e se transformaram nas duas classes que compõem o *applet*.

Basicamente, a diferença entre o aplicativo e o *applet* está no fato do *applet* não possuir uma interface gráfica de edição e não permitir o acesso ao disco local.

A classe music.JM possui apenas dois métodos. O primeiro é o método init(), padrão da classe *applet*, que cria uma instância de music.MusicPanel com as dimensões definidas no corpo do documento HTML pelos parâmetros width e height da marca *<applet>*. O segundo método, openXmlFile(), carrega o documento XML definido também no corpo do HTML pelo parâmetro music da marca <param>.

É a classe music.MusicPanel que tem a maior responsabilidade na estrutura do *applet*. Sua função é interpretar o código XML e renderizar todos os elementos na saída gráfica, bem como permitir que o arquivo XML possa ser tocado no computador do usuário.

Abaixo podemos ver um exemplo do código usado para carregar o *applet* JM na página HTML:

```
< applet code=music.JM archive=jm.jar width=586 height=343
        codebase="http://sheratan.mc21.fee.unicamp.br/~craraujo/Mestrado/jm/">
        <param name="music" value="cancao.xml">
</ applet>
```

Com esta explanação, concluímos a análise da estrutura básica do *software* JavaMusic e do *applet* JM.

6 Usando o JavaMusic na aula de música

A apresentação de partituras em *sites* para educação musical, como já foi analisado anteriormente, depende da digitalização das partituras para sua inclusão nas páginas HTML. Com a utilização do *software* JavaMusic, esse trabalho é substituído pela construção das partituras com o aplicativo JavaMusic e sua inserção nas páginas HTML via *applet*. Vamos analisar agora como o JavaMusic pode ser usado num contexto de aula de música de um curso superior de música semelhante ao da UNICAMP, propondo o uso da Internet como uma ferramenta acessória às aulas presenciais.

Apesar da ferramenta JavaMusic ter sido criada para auxiliar a preparação de material para a *web*, ela também pode ser usada pelo professor dentro da sala de aula. Desse modo, diversas situações podem ser encontradas de acordo com a criatividade do professor.

Imaginemos uma aula de percepção, na qual os alunos devam solfejar um exercício proposto pelo professor. Ao invés do material ser impresso e copiado para todos os alunos, o professor pode preparar os exercícios de solfejo usando o JavaMusic e, com o auxílio de um *DataShow*, pode projetar os exercícios num telão, onde serão vistos por toda a classe.

O professor deve preparar os exercícios antes da aula. Ele tanto pode salvar os exercícios em disquete e levar para a sala de aula quanto pode alocá-los no *site* de sua disciplina que será acessado pelos próprios alunos para repetirem os exercícios em casa. No primeiro caso, o professor poderá usar a função "tela cheia" (ctrl+f) durante a projeção para facilitar a visualização. Já no segundo caso, será necessário que a sala possua uma conexão com a Internet para acessar o *site*.

6.1 Criando um exercício de solfejo com o auxílio do JavaMusic

Abrimos o JavaMusic e selecionamos, no índice "File", um novo documento ("New") ou também podemos clicar no ícone  , ou ainda, usar o atalho de teclado "crtl+n". Aparecerá a seguinte caixa de diálogo:

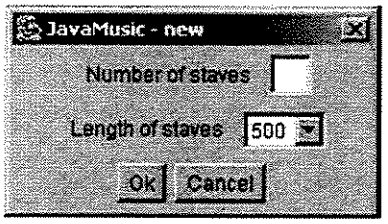


Figura 40: Novo Documento

Digitamos a quantidade de pentagramas necessários ao exercício e o comprimento desejado.

Podemos escrever o exercício selecionando a figura desejada na barra de ferramentas musicais e clicando em seguida sobre o pentagrama. Repetimos a operação até terminarmos de escrever todo o exercício.

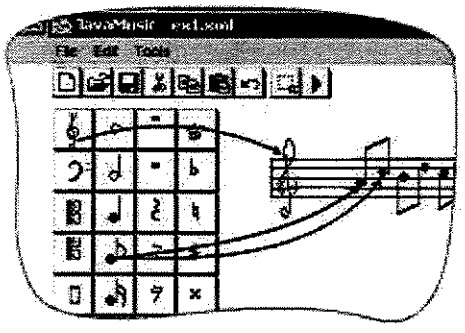


Figura 41: Inserindo elementos no pentagrama

O botão "BLINE" deve ser usado para inserir uma barra de compasso escolhida dentre as oito barras disponíveis na caixa de diálogo:

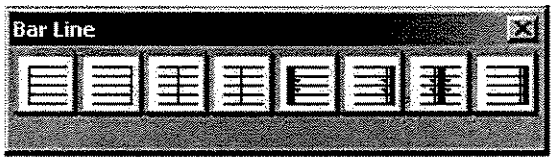


Figura 42: Barras de compasso

Já o botão "KEYSIGN" deve ser usado para selecionar uma armadura de clave:

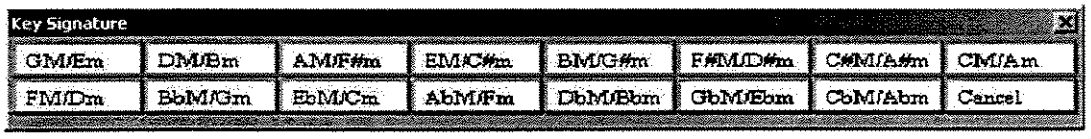


Figura 43: Selecionando a armadura de clave

Para escrever as fórmulas de compasso, usamos o botão "TIME". A fórmula de compasso desejada deve ser digitada como aparece numa partitura, ex.; 3/4, C, 6/8. Para usar "C" cortado digitamos "ç".

A mesma caixa de diálogo, pode ser usada para inserir mudanças de andamento na música. Basta selecionar a figura de referência e indicar o tempo desejado.

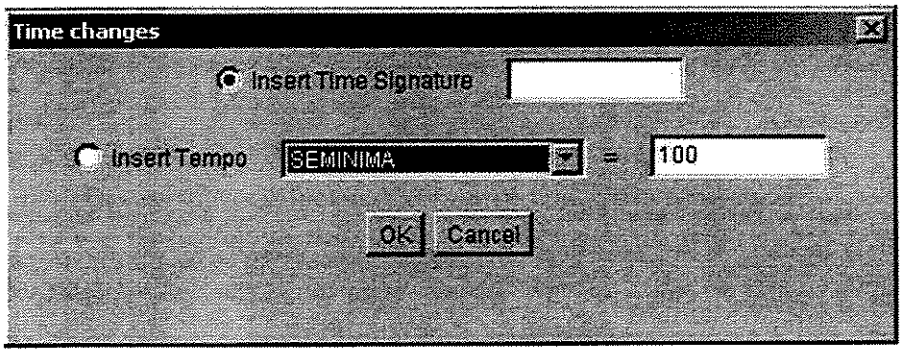


Figura 44: Fórmulas de compasso e alterações de andamento

Uma vez tendo escrito todo o exercício, o autor pode usar o botão  ou o atalho "ctrl+s" para entrar no modo de seleção de objetos. Neste modo de seleção, é possível selecionar um ou mais objetos sobre os quais pode-se aplicar os comandos:

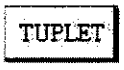
 recortar ("ctrl+x");

 copiar ("ctrl+c");

 colar ("ctrl+v");

 mudo ("ctrl+m");

 escondido ("ctrl+h");

 quiáltera ("ctrl+t");

 agrupar ("ctrl+g");

Depois de pronto o exercício, pode-se clicar em  ou teclar a barra de espaço para tocar.

No entanto, o professor pode querer impedir que todas as notas sejam tocadas e, assim, obrigar os alunos a cantarem o exercício (solfejar). Para isso, basta selecionarmos as notas que não devem ser tocadas e clicar no botão "M".

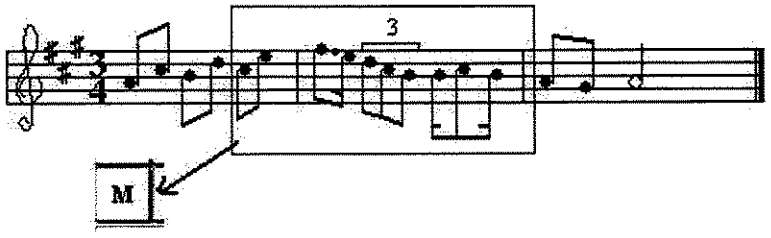


Figura 45: Usando a função MUTE

No exemplo acima, as quatro primeiras notas serão tocadas para dar a referência da afinação e do andamento, mas as notas selecionadas não serão tocadas. A função "MUTE" também pode ser usada para encobrir a armadura de clave, as alterações de tempo e as quiálteras.

A opção do menu "File", "Save" ou o atalho "ctrl+b" ou o botão  deve ser usada para salvar o documento e no menu "Tools", opção "Create HTML file", permite gerar um documento HTML que será usado para publicar o material criado na *web*. Será mostrada a seguinte caixa de diálogo:

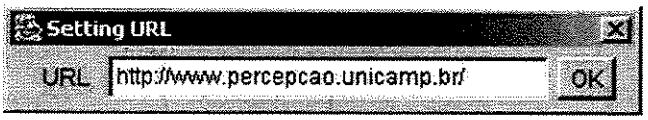


Figura 46: Informando o URL do *site*

Deve-se informar o endereço completo do *site* onde será alocado o material criado, inclusive com seus subdiretórios.

O arquivo HTML gerado em seguida deve ser editado com uma ferramenta apropriada, como MS FrontPage, Netscape Composer, ou outra disponível, sendo que dois ou mais documentos HTML podem ser agrupados numa mesma página, se for desejado que dois ou mais exercícios apareçam na mesma página.

6.2 Criando um exercício de ditado com o auxílio do JavaMusic

Agora vamos imaginar uma nova situação, ainda na aula de percepção: o ditado.

O professor pode preparar os ditados usando o JavaMusic. Com a função *hide*, ele pode esconder elementos da partitura. Esta função pode ser aplicada sobre as notas, sobre a armadura de clave, sobre a fórmula de compasso, sobre as barras de compasso, enfim, sobre qualquer elemento.

Seu uso é bastante simples, bastando selecionar os elementos que se deseja esconder e depois clicar em "H":

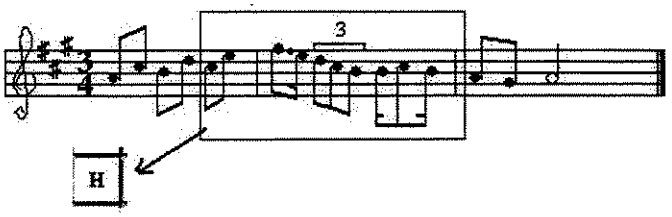


Figura 47: Usando a função "HIDE"

O resultado visível tanto com a função "tela cheia" quanto através do *applet* no documento HTML será:

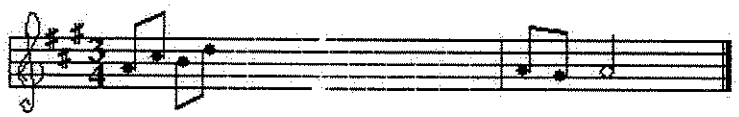


Figura 48: Elementos escondidos pela função "HIDE"

A função *hide* poderia ter sido usada, por exemplo, para esconder a armadura de clave ou ainda as barras de compasso. Assim, os alunos teriam que descobrir qual a tonalidade do exercício ou colocar as barras de compasso.

6.3 Outros exemplos

O JavaMusic permite também alterar o modo como as notas são tocadas. O professor pode, por exemplo, fazer com que as notas soem mais curtas (*staccato*). Para isso deve usar o botão "ARTIC". Uma caixa de diálogo será mostrada para definir a medida da nota:

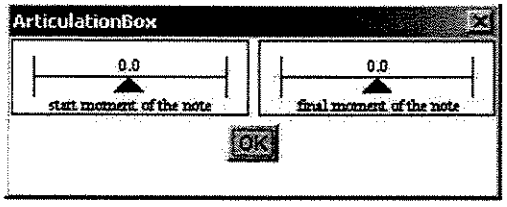


Figura 49: Configurando a articulação

Nesta caixa de diálogo aparecem duas réguas. A da esquerda controla o instante inicial de uma nota. A escolha de valores negativos faz com que a nota inicie adiantada em relação à sua posição na partitura e a escolha de valores positivos faz com que a nota inicie atrasada em relação à sua posição na partitura. A régua da direita controla o instante final da nota. Valores negativos fazem com que a nota termine antes de completar sua duração total e podem ser usados para simular um staccato. Valores positivos fazem a nota terminar depois de completar sua duração total e podem ser usados para ligar duas notas.

Uma vez selecionada uma determinada articulação, todas as notas que forem inseridas a seguir repetirão a mesma articulação. Para alterar a articulação de uma nota já inserida, basta configurar a articulação desejada, clicar no botão "OK" e, em seguida, clicar sobre as notas que se deseja alterar.

O mesmo processo pode ser usado para alterar a dinâmica (intensidade) das notas. O botão "DINAMIC" abre a seguinte caixa de diálogo:

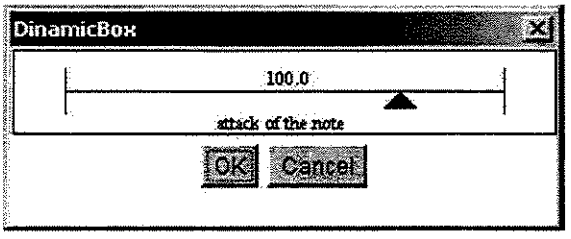


Figura 50: Configurando a intensidade das notas

Nesta caixa, uma régua é usada para escolher um valor entre 0 e 127. Estes valores correspondem aos valores reconhecidos pelas placas de som dos computadores.

O professor também poderá usar o botão "TEXT" para inserir textos na partitura. Uma caixa de diálogo é exibida para que seja digitado o texto desejado. Depois, basta clicar no ponto da partitura onde o texto deve aparecer.

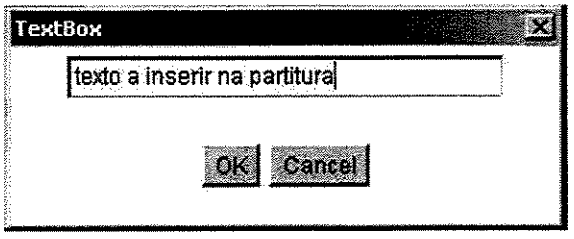


Figura 51: Inserindo textos

Usando o JavaMusic para inserir partituras em páginas HTML, o professor poderá combinar texto explicativo sobre um exemplo musical com as partituras. Para fazer tocar as partituras, basta clicar na palavra "play" que aparece logo acima das partituras. Ao publicar o *site*, o professor não deve se esquecer de enviar para o endereço do *site* as páginas HTML, os arquivos XML contendo as partituras e uma cópia do *applet* JM (arquivo jm.jar).

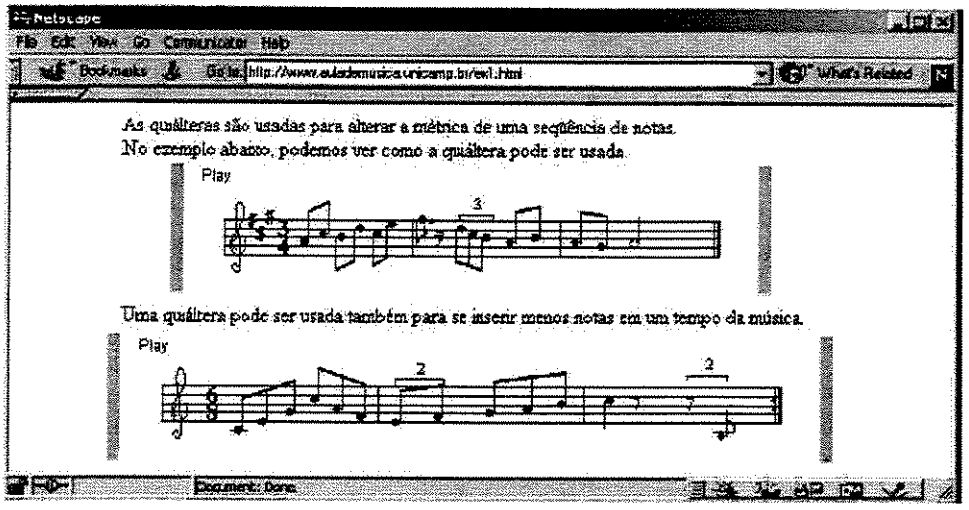


Figura 52: Partituras vistas no *browser* com o *applet* JM

Com este recurso, é possível construir um *site* sobre teoria musical totalmente interativo, sem a necessidade de digitalizar partituras ou gerar arquivos de imagens a partir de editores comerciais.

Em sala de aula, o professor pode observar que alguns de seus alunos com mais dificuldade necessitam de um esclarecimento mais detalhado sobre o assunto estudado. Neste caso, a Internet pode ser usada para esse estudo complementar. O professor, então, constrói um *site* sobre o assunto estudado e escreve os exemplos musicais usando o *software* JavaMusic.

Os alunos poderão acessar o *site* usando seu computador pessoal ou um dos computadores do laboratório da faculdade e, assim, poderão ter uma aula extra sobre a matéria em questão no horário que lhes for mais conveniente. Poderão, também, imprimir o conteúdo para montar uma apostila ou copiar o conteúdo para seu próprio computador.

Esse método de ensino permite o "congelamento" da aula, de modo que o material que for preparado para uma turma poderá ser reaproveitado para as demais aulas ou para outras classes da mesma disciplina, além de que o professor poderá editá-lo a qualquer momento para aprimorá-lo.

Outra situação em que este processo pode ser usado é na criação de material mais aprofundado para alunos que estejam apresentando um destaque em sala de aula.

A utilização da Internet pode caminhar tanto na direção da personalização do ensino quanto na direção da sua generalização, uma vez que, o professor poderá usar os recursos dos *applet* e do aplicativo JavaMusic para a criação de página com conteúdo introdutório às suas aulas, buscando preparar seus alunos para as aulas presenciais de modo a atingir uma maior homogeneidade entre os alunos.

Além destes exemplos de utilização do *software* JavaMusic num curso superior de música, o *software* também poderia ser empregado na construção de *sites* sobre música voltado a estudantes autodidatas, ao treinamento de professores de artes ou, ainda ser utilizado por conservatórios e escolas livres de música.

Compositores e editoras de partituras poderiam usar o JavaMusic em seus *sites* para mostrar trechos de obras em seus catálogos virtuais além de outras aplicações.

7 Conclusão

A utilização do JavaMusic concretiza o cenário anteriormente descrito, no qual o professor poderá usar o aplicativo para gerar seus exemplos musicais para a criação de *sites* de apoio à educação musical enriquecendo o estudo individual dos alunos e/ou as atividades em sala de aula. O *applet* JM, parte integrante do *software*, é capaz de gerar uma saída gráfica, em forma de partitura, para os arquivos XML e permitir a sua execução com um simples toque do mouse e, desta forma, integrar conteúdo textual, gráfico (partituras) e sonoro numa mesma interface.

A implementação do JavaMusic foi uma experiência de integração entre conhecimentos de múltiplas áreas como programação em linguagem Java, XML e música. Este trabalho tem como contribuição importante a afirmação de que a **interdisciplinaridade** pode ajudar a encontrar soluções mais simples para questões ligadas a educação como, por exemplo, o uso da Internet como meio educacional. Não se obteve aqui um *software* acabado, pronto para uso, mas uma demonstração de como podem ser aplicados conhecimentos da área tecnológica em auxílio à área de educação musical.

O JavaMusic propõem alguns recursos para controlar a execução da música como a possibilidade de atrasar ou adiantar o início e/ou o término de uma nota, a criação de quiálteras de qualquer formato, a possibilidade de se esconder notas da partitura ou impedir sua execução. Esses recursos foram desenvolvidos para enriquecer as possibilidades de ditados e exercícios de solfejo.

O desenvolvimento deste *software* teve por objetivo estudar as questões ligadas ao desenvolvimento de *software* para música, o uso de XML para a representação de partituras e a criação

de objetos musicais no computador. Muitos problemas foram mapeados durante os trabalhos e pode-se verificar as dificuldades envolvidas neste tipo de projeto, destacando-se: primeiro, **a conversão de informações do domínio da música para o domínio computacional**: a associação entre a representação gráfica (partitura) e sua representação computacional (neste caso, com o emprego do XML) contraposto à representação computacional e sua conversão em som; segundo, **a apresentação gráfica**: a renderização dos elementos gráficos na tela do computador é um assunto que entra no domínio da computação gráfica exigindo que tais conhecimentos estejam presentes na formação das pessoas envolvidas no projeto do *software*.

Alguns problemas relativos à implementação do JavaMusic foram identificados, particularmente com relação à execução das partituras geradas pelo programa: pode-se ouvir claramente um pequeno atraso na execução de algumas notas, quebrando a fluência ou continuidade da música. Este problema é bastante grave do ponto de vista musical e por isso faz-se necessária a busca por uma solução para que o programa possa então efetivamente ser utilizado. Não houve uma preocupação em solucionar definitivamente este problema neste momento, já que a implementação do JavaMusic foi apenas para visualizar o conceito de utilização da Internet no ensino de música. No entanto, uma mudança na estrutura do programa, buscando um melhor desempenho, poderia ser a criação de um aplicativo (e de um *applet*) independente do aplicativo principal que seria invocado pelo JavaMusic para tocar as partituras. Esse aplicativo não teria que compartilhar recursos da máquina virtual Java na qual o JavaMusic estaria rodando, tendo para isso sua própria instância de máquina virtual e assim poderia se esperar uma eficiência maior em relação estrutura atualmente usada e conseqüentemente reduziria as interrupções na execução.

Outra ação que deverá ocorrer para o aperfeiçoamento do JavaMusic é a otimização das classes de representação gráfica e, principalmente, da classe music.graphics.GPentagram. Uma das possibilidades é transformar a classe GPentagram em um *container* capaz de receber objetos musicais e automaticamente gerar a saída gráfica para esses objetos. Com isso a classe music.graphics.MGEBuffer poderia ser depreciada.

Em relação ao código XML, atualmente todas as marcas contêm um parâmetro chamado "*position*" que informa ao JavaMusic a posição na qual cada elemento deve ser desenhado. Uma evolução no código passa por criar um mecanismo no JavaMusic para posicionar todos os elementos gráficos de forma automática e transparente. Porém, se o usuário desejar forçar o posicionamento de determinado símbolo, ao invés de usar o parâmetro "position", seria mais eficiente a incorporação de recursos de estilo já disponíveis na especificação do XML, como por exemplo, a utilização de folhas de

estilo (CSS – *Cascade Stile Sheet*) ou de XSL (*Extensible Stylesheet Language*). Com isso, seria possível separar totalmente o aspecto conteúdo do aspecto apresentação.

No DTD usado neste trabalho, também pode-se observar alguns elementos que poderiam ser melhorados, como por exemplo, a representação de quiálteras e de notas agrupadas, que poderiam ser definidas dentro de uma estrutura mais hierárquica e mais intuitiva como no exemplo a seguir:

```
<group>
        <note sound="B" eighth="5" duration="COLCHEIA" />
        <note sound="G" eighth="5" duration="COLCHEIA" />
</group>
```

Falta também incorporar a representação de acordes, melodias múltiplas e instrumentação, que não foram implementadas nesta versão.

O JavaMusic poderia ser modificado para operar numa estrutura cliente-servidor, o que lhe permitiria a agregação de novas funcionalidades e competências, tanto do lado do aplicativo (professor/autor), como no lado do *applet* (aluno).

A princípio, o aplicativo poderia ser substituído por um *applet* com as mesmas funcionalidades do JavaMusic, no qual o autor poderia desenvolver o material didático diretamente na *web*, sem se preocupar com os detalhes da publicação do material que poderiam ser gerenciadas pelo próprio servidor. O *applet* JM poderia incorporar as funções de edição do aplicativo JavaMusic e ser conectado a um *servlet* possibilitando uma comunicação bidirecional entre a interface do aluno e o sistema servidor, ou mesmo entre o aluno e a interface do professor. O aplicativo poderia receber funções para controlar as ações dos alunos e acompanhar seus estudos extraclasse, tudo de forma transparente aos usuários.

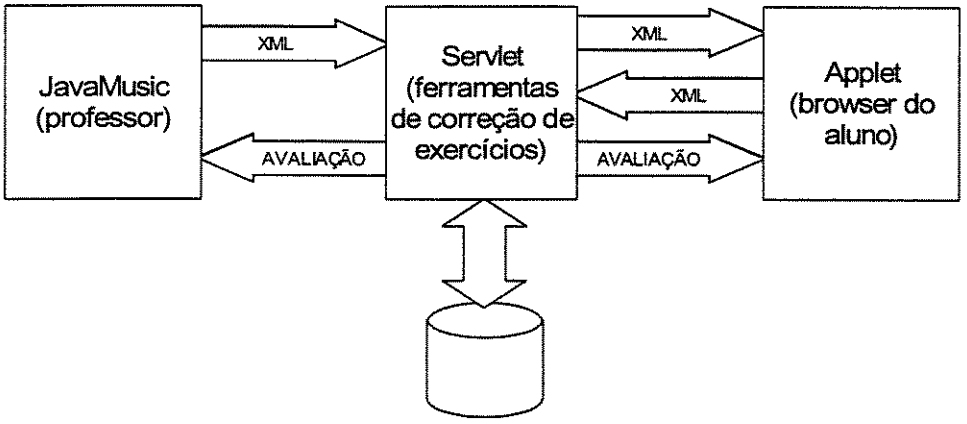


Figura 53: usando *servlet* para interligar interfaces do professor e do aluno

Também poderia ser criado um mecanismo de correção automática para os exercícios, o que permitiria informar ao aluno sobre seu desempenho e permitir uma imediata reflexão sobre seus erros, além de permitir ao professor um acompanhamento *on-line* dos alunos.

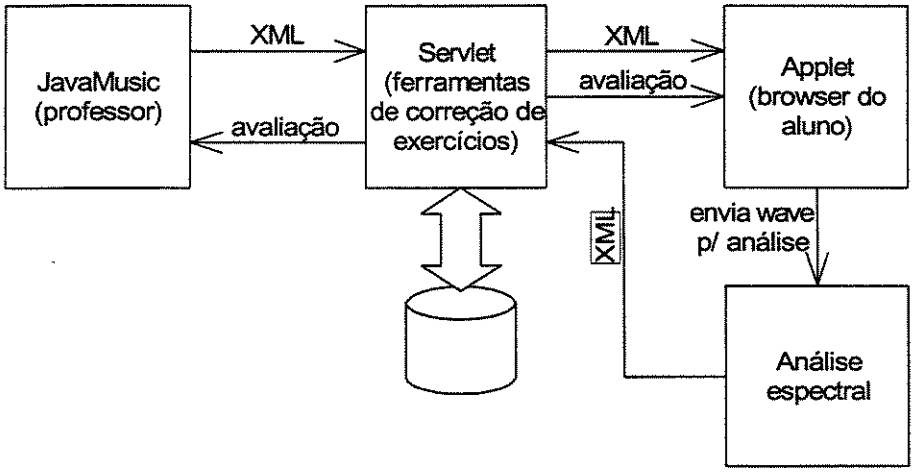


Figura 54: usando análise espectral para correção de exercícios

Outros recursos, como mostrado na figura acima, poderiam ser acrescentados para que o aluno, usando o microfone do computador, pudesse enviar ao professor seus exercícios de leitura. O sistema converteria a gravação do aluno, do domínio do tempo para o domínio da freqüência [OSW99], as informações seriam transferidas para o servidor em formato XML e este faria a correção do exercício comparando o resultado com um gabarito criado pelo professor. O professor, por sua vez, receberia

informações sobre o aluno podendo saber quais as dificuldades de cada aluno em relação à leitura e ao solfejo.

Todo o material criado pelos professores poderia ficar armazenado num banco de dados, podendo ser consultado por todos os alunos cadastrados no serviço, independentemente de sua instituição de origem, caso houvesse um acordo de compartilhamento desses dados, obtendo um arquivo que seria enriquecido pela contribuição de professores de diversas origens. Os professores, por suas vezes, teriam acesso às ações dos alunos, podendo avaliar não somente seu desempenho na realização dos exercícios propostos, bem como conhecer o grau de interesse dos alunos pelas aulas e exercícios armazenados no banco de dados do sistema.

O potencial da linguagem XML para a representação musical é muito grande e permitiria o desenvolvimento de recursos como busca de melodias, ferramentas de análise da estrutura musical, geração automática de acompanhamento e comparação entre trechos da melodia em busca de padrões.

Novos métodos de entrada de dados poderiam ser propostos como: a utilização de um instrumento musical conectado ao computador, o que substituiria o uso do *mouse* para a inserção das notas na partitura; a digitalização de partituras impressas associado ao uso de recursos como análise de padrões e o uso de microfone ou de arquivos de áudio e MIDI.

Finalizando, é importante ressaltar que a proposta do JavaMusic, quando estiver totalmente desenvolvida e pronta para o uso, não prevê a substituição do curso tradicional de música por cursos à distância. O novo paradigma que aqui começa a ser estabelecido propõe, em verdade, a utilização da *web* como uma extensão da sala de aula, provendo aos cursos de música, recursos para seu crescimento em qualidade e, assim, propiciando uma melhor formação aos futuros profissionais da música.

Também, é importante ressaltar que o desenvolvimento de ferramentas educacionais para a Internet, seja em qualquer área do conhecimento, pode promover um maior acesso à formação e à especialização de indivíduos que, eventualmente, tenham dificuldades de freqüentar ou acompanhar uma sala de aula de ensino presencial, possibilitando, portanto, uma maior democratização do ensino.

8 REFERÊNCIAS

[4ML] **4ML**. [*on-line*] disponível em: http://www.4ml.org. 2000. consultado em out/2001.

[BAR98] BARRET, D. **Notation on the web**. Keyboard On-line. [*on-line*] disponível em: http://www.keyboardmag.com/features/netsmarts9812/netsmarts.shtml. 1998. Consultado em out/2001.

[CAS98] CASTAN, G. **Common music notation and computers: motivation**. [*on-line*] disponível em: http://www.s-line.de/homepages/gerd_castan/compmus/MusiXML_e.html. 1998. Consultado em: out/2001.

[CHO01] CHOI, W et all. **Beginning PHP4 – Programando**. São Paulo, Makron, 2001, 719p.

[CODA] CODA Music Technology. [*on-line*] disponível em: http://www.codamusic.com/coda. Consultado em: dez/2001.

[DEI97] DEITEL & DEITEL, **Java how to program**, N. Jersey, Prentice Hall, 1997.

[FRI99] FRITSCH, E. F. et al. **Developing a software for music education: an interdisciplinary project.** Anais do XIX Congresso Nac. da Soc. Bras. de Computação. Rio de Janeiro. 1999. v.3.: p. 251-264.

[FUE00] FUERTES, C. **Music education by computer**. [*on-line*] disponível em: http://www.xtec.es/rtee/eng/tutorial/index.htm. [2000?]. consultado em out/2001

[GRI98] GRIGAITIS, R. J. **Extensible Score Language (Xscore) 0.01**. [*on-line*] disponível em: http://fn2.freenet.edmonton.ab.ca/~rgrigait/xscore. 1998. consultado em out/2001.

[JRE] **JAVARUNTIME**. [*on-line*] disponível em http://java.sun.com/j2se/1.3/jre/index.html.

[JSO99] **JAVA SOUND API programmer's Guide.** Sun Microsystems, Inc. 1999. [*on-line*] disponível em http://java.sun.com/products/jdk/1.3/docs/sound/. consultado em jul/2000

[JSYN] **JSYNTM**: audio synthesis API for Java. [*on-line*] disponível em http://www.softsynth.com.

[LUC99] LUCENA, C. J. P. et al, **AulaNet: ajudando professores a fazerem seu dever de casa**. Anais do XIX Congresso Nac. da Soc. Bras. de Computação. Rio de Janeiro. 1999. v.1. p. 106-117.

[MANZ] MANZOLLI, J., **Rabisco:** From Kandinsky to Stravinsky. [*on-line*] disponível em: http://www.nics.unicamp.br/rabisco. consultado em dez/2000.

[MAR99] MARTINS, J. G. et al. **A transformação do ensino através do uso da tecnologia na educação**. Anais do XIX Congresso Nacional da Soc. Bras. de Computação. Rio de Janeiro. 1999. v.1.: p. 571-579

[MID] **SOURCES of MIDI Information**. [*on-line*] disponível em: http://www.midi.org/about-midi/resource.htm. consultado em: fev/2002

[MOO96] MOORE, F. Richard. **Dreams of Computer Music - Then and Now.** Computer Music Journal. 1996. v. 20 nº 1. p. 25-41.

[MOR99] MORRISON, M. et al. **Como programar em JavaBeans**, Makron, 1999.

[OSW99] OSWALT, E., **Spectral analysis of sound (proposal).** [*on-line*] disponível em: http://www.jao.com/ace/wavs/pca.html. [1999?]. consultado em: ago/2000

[PIV99] PIVA JR., D. **Educador Digital**: Uma introdução à cultura digital para educadores. Itu: Edigital, 1999.

[RAP99] RAPOSO, A. B.; MAGALHÃES, L. P.; RICARTE, I. L. M. **Interação na *WEB***. Anais do XIX Congresso Nac. da Soc. Bras. de Computação. Rio de Janeiro.1999.v.2. p.1-50

[REC00] **RECORDADE – MusicXML definition**, http://www.musicxml.org. 2000. consultado em: out/2001.

[ROT97] ROTTERDAM, J. van. **MusicML: an XML experience**. The Connection Factory. [*on-line*] disponível em: http://195.108.47.160/3.0/musicml. 1997. consultado em: jun/2000.

[SAN99] SANTANCHÈ, A.; TEIXEIRA, C. A. C. **Explorando linguagens de markup extensíveis na construção de sistemas de educação baseados na *web*.** . Anais do XIX Congresso Nac. da Soc. Bras. de Computação. Rio de Janeiro. 1999. v.1.: p. 39-55.

[SAX] SAX. [*on-line*] disponível em: http://www.saxproject.org.

[SCH98] SCHANK, R. C., **Horses for courses**. Communication of ACM. jul/98. v.41 nº 7

[SCT00] SCHIETTECATTE, Bert. **A format for virtual orchestras: FlowML**. [*on-line*] disponível em: http://www.flowml.com. 2000. consultado em out/2001.

[SIBE] **SIBELIUS Music notation *software*.** [*on-line*] disponível em: http://www.sibelius.com. consultado em dez/2001.

[SOA99] SOARES, J. M. et al. **Um framework para educação à distância baseado na *web* usando applets e servlets**. Anais do XIX Congresso Nac. da Soc. Bras. de Computação. Rio de Janeiro. 1999. v.1.: p. 149-165.

[SPY] XMLSPY. [*on-line*] disponível em: http://www.xmlspy.com.

[STE99] STEYN, J. **Music Markup Language**. http://www.mmlxml.org. 1999. consultado em: out/2001.

[UNE95] UNES, W. A., BARRANECHEA, S. A. **A informática aplicada à música através da tecnologia MIDI.** Anais do IV Encontro anual da Associação Brasileira de Educação Musical. 1995. p 87-88

[WOL98] WOLZ, U. et al. **Computer-mediated communication in collaborative educational settings.** ItiCSE'97 Working Group on CMC in Collaborative Educational Settings: Computer-Mediated Communication in Collaborative Educational Settings. 1998. ACM SIGCUE Outlook 25(4): p. 51-69.

9 APÊNDICE A - Document Type Definition (DTD)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 (http://www.xmlspy.com) by Cláudio R. Araújo (Unicamp) -->
<!--DTD generated by XML Spy v3.5 (http://www.xmlspy.com)-->
<!ELEMENT author (#PCDATA)>
<!ELEMENT barline EMPTY>
<!ATTLIST barline
    type (Beguin | End | EndScore | RepeatLeft | RepeatRight | RepeatDouble | Double | Single)
    #REQUIRED
    status (m | h | mh | hm) #IMPLIED
    position CDATA #REQUIRED >
<!ELEMENT clef EMPTY>
<!ATTLIST clef
    name (P_CLEF | F3_CLEF | F4_CLEF | G_CLEF | C3_CLEF | C4_CLEF) #REQUIRED
    status (m | h | mh | hm) #IMPLIED
    position CDATA #REQUIRED >
<!ELEMENT document (#PCDATA)>
<!ELEMENT group EMPTY>
<!ATTLIST group
    size CDATA #REQUIRED
    status (m | h | mh | hm) #IMPLIED
    position CDATA #REQUIRED >
<!ELEMENT inits (document, author)>
<!ATTLIST inits
    length CDATA #REQUIRED
    numberofstaves CDATA #REQUIRED >
<!ELEMENT keysignature EMPTY>
<!ATTLIST keysignature
    tone CDATA #REQUIRED
    status (m | h | mh | hm) #IMPLIED
    position CDATA #REQUIRED >
<!ELEMENT music (inits, staf+)>
<!ATTLIST music
```

version CDATA #REQUIRED >

<!ELEMENT note EMPTY>

<!ATTLIST note

sound (A | B | C | D | E | F | G) #REQUIRED

eighth CDATA #REQUIRED

duration (QUARTIFUSA | SEMIFUSA | SEMIFUSA_DOT | FUSA | FUSA_DOT | FUSA_DOT_DOT | SEMICOLCHEIA | SEMICOLCHEIA_DOT | SEMICOLCHEIA_DOT_DOT | COLCHEIA | COLCHEIA_DOT | COLCHEIA_DOT_DOT | SEMINIMA | SEMINIMA_DOT | SEMINIMA_DOT_DOT | MINIMA | MINIMA_DOT | MINIMA_DOT_DOT | SEMIBREVE | SEMIBREVE_DOT | SEMIBREVE_DOT_DOT | BREVE | BREVE_DOT | BREVE_DOT_DOT | LONGA | LONGA_DOT | LONGA_DOT_DOT) #REQUIRED

signal (DOUBLESHARP | SHARP | NATURAL | FLAT | DOUBLEFLAT) #IMPLIED

attack CDATA #IMPLIED

start CDATA #REQUIRED

end CDATA #REQUIRED

status (m | h | mh | hm) #IMPLIED

position CDATA #REQUIRED >

<!ELEMENT quialtera EMPTY>

<!ATTLIST quialtera

duration (FUSA | FUSA_DOT | FUSA_DOT_DOT | SEMICOLCHEIA | SEMICOLCHEIA_DOT | SEMICOLCHEIA_DOT_DOT | COLCHEIA | COLCHEIA_DOT | COLCHEIA_DOT_DOT | SEMINIMA | SEMINIMA_DOT | SEMINIMA_DOT_DOT | MINIMA | MINIMA_DOT | MINIMA_DOT_DOT | SEMIBREVE | SEMIBREVE_DOT | SEMIBREVE_DOT_DOT | BREVE | BREVE_DOT | BREVE_DOT_DOT) #REQUIRED

size CDATA #REQUIRED

index CDATA #REQUIRED

height CDATA #REQUIRED

width CDATA #REQUIRED

position CDATA #REQUIRED >

<!ELEMENT pause EMPTY>

<!ATTLIST pause

duration (QUARTIFUSA | SEMIFUSA | SEMIFUSA_DOT | FUSA | FUSA_DOT | FUSA_DOT_DOT | SEMICOLCHEIA | SEMICOLCHEIA_DOT | SEMICOLCHEIA_DOT_DOT | COLCHEIA | COLCHEIA_DOT | COLCHEIA_DOT_DOT | SEMINIMA | SEMINIMA_DOT | SEMINIMA_DOT_DOT | MINIMA | MINIMA_DOT | MINIMA_DOT_DOT | SEMIBREVE | SEMIBREVE_DOT |

SEMIBREVE_DOT_DOT | BREVE | BREVE_DOT | BREVE_DOT_DOT | LONGA | LONGA_DOT | LONGA_DOT_DOT) #REQUIRED

status (m | h | mh | hm) #IMPLIED

position CDATA #REQUIRED >

<!ELEMENT staf (clef | barline | keysignature | timesignature | pause | tempo | group | note | quialtera | text)+>

<!ATTLIST staf

number CDATA #REQUIRED >

<!ELEMENT tempo EMPTY>

<!ATTLIST tempo

bpm CDATA #REQUIRED

unit (FUSA | SEMICOLCHEIA | SEMICOLCHEIA_DOT | COLCHEIA | COLCHEIA_DOT | SEMINIMA | SEMINIMA_DOT | MINIMA | MINIMA_DOT) #REQUIRED

heigth CDATA #REQUIRED

status (m | h | mh | hm) #IMPLIED

position CDATA #REQUIRED >

<!ELEMENT text EMPTY>

<!ATTLIST text

string CDATA #REQUIRED

heigth CDATA #REQUIRED

position CDATA #REQUIRED>

<!ELEMENT timesignature EMPTY>

<!ATTLIST timesignature

rhythm CDATA #REQUIRED

position CDATA #REQUIRED >

10 APÊNDICE B - Exemplo de partitura em XML

```
<?xml version="1.0"?>
<!DOCTYPE music SYSTEM "Javamusic.dtd">
<music version="1.0">
    <inits   length= "500" numberofstaves= "3">
        <document>Boa Noite Meu Jesus</document>
        <author>desconhecido</author>
    </inits>
    <staf number="1">
        <clef name="G_CLEF" position="42"/>
        <barline type="Beguin" position="50"/>
        <keysignature tone="EbM/Cm" position="57"/>
        <tempo bpm="100" unit="SEMINIMA" heigth="88" position="82"/>
        <timesignature rhythm="3/4" position="84"/>
        <pause duration="SEMINIMA" position="109"/>
        <pause duration="SEMINIMA" position="134"/>
        <text string="Len - tae" heigth="143" position="170"/>
        <group size="2" position="170"/>
        <note sound="E" eighth="5" duration="COLCHEIA" start="0.0" end="0.0" position="171"/>
        <note sound="F" eighth="5" duration="COLCHEIA" start="0.0" end="0.0" position="201"/>
        <text string="cal -" heigth="143" position="220"/>
        <note sound="G" eighth="5" duration="SEMINIMA" attack="70" start="0.0" end="-27.0"
          position="227"/>
        <text string="ma," heigth="143" position="266"/>
        <note sound="G" eighth="5" duration="SEMINIMA" attack="78" start="0.0" end="-27.0"
          position="272"/>
        <text string="so-brea" heigth="144" position="306"/>
        <group size="2" position="310"/>
        <note sound="B" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="0.0"
          position="311"/>
        <note sound="G" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="0.0"
          position="327"/>
        <barline type="Single" position="366"/>
```

```
        <text string="ter -" heigth="144" position="386"/>
        <note sound="G" eighth="5" duration="SEMINIMA" attack="78" start="0.0" end="-27.0"
            position="395"/>
        <text string="ra" heigth="144" position="427"/>
        <note sound="F" eighth="5" duration="SEMINIMA" attack="78" start="0.0" end="-27.0"
            position="436"/>
        <text string="des - ce a" heigth="144" position="473"/>
        <group size="2" position="477"/>
        <note sound="F" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="11.0"
            position="478"/>
        <note sound="G" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="11.0"
            position="510"/>
        <barline type="End" position="528"/>
    </staf>
    <staf number="2">
        <clef name="G_CLEF" position="42"/>
        <barline type="Beguin" position="53"/>
        <keysignature tone="EbM/Cm" position="63"/>
        <text string="noi -" heigth="243" position="104"/>
        <note sound="A" eighth="5" duration="SEMINIMA" attack="78" start="0.0" end="-27.0"
            position="110"/>
        <text string="te" heigth="242" position="142"/>
        <note sound="C" eighth="6" duration="SEMINIMA" attack="70" start="0.0" end="-27.0"
            position="149"/>
        <text string="c=100; fo - ge a" heigth="241" position="177"/>
        <group size="2" position="181"/>
        <note sound="B" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="0.0"
            position="182"/>
        <note sound="A" eighth="5" duration="COLCHEIA" attack="70" signal="NATURAL"
            start="0.0" end="0.0" position="208"/>
        <barline type="Single" position="245"/>
        <text string="c=100; luz" heigth="242" position="263"/>
        <note sound="B" eighth="5" duration="MINIMA" attack="70" start="0.0" end="-27.0"
            position="270"/>
        <barline type="RepeatLeft" position="301"/>
        <text string="c=001; Quero a -" heigth="237" position="320"/>
```

```
        <group size="2" position="324"/>
        <note sound="B" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="8.0"
            position="325"/>
        <note sound="B" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="8.0"
            position="352"/>
        <barline type="Single" position="375"/>
        <text string="c=001; s=12;go -" heigth="238" position="390"/>
        <note sound="E" eighth="6" duration="SEMINIMA" attack="70" start="0.0" end="-27.0"
            position="397"/>
        <text string="c=001; s=12;ra" heigth="237" position="428"/>
        <note sound="D" eighth="6" duration="SEMINIMA" attack="78" start="0.0" end="-27.0"
            position="435"/>
        <text string="c=001; s=12;des - pe -" heigth="236" position="483"/>
        <group size="2" position="485"/>
        <note sound="C" eighth="6" duration="COLCHEIA" attack="70" start="0.0" end="11.0"
            position="486"/>
        <note sound="B" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="11.0"
            position="511"/>
        <barline type="End" position="529"/>
    </staf>
    <staf number="3">
        <barline type="Beguin" position="33"/>
        <clef name="G_CLEF" position="40"/>
        <keysignature tone="EbM/Cm" position="63"/>
        <text string="c=011; s=12;dir - " heigth="340" position="85"/>
        <note sound="C" eighth="6" duration="SEMINIMA" attack="78" start="0.0" end="-27.0"
            position="96"/>
        <text string="c=011; s=12;me" heigth="338" position="124"/>
        <note sound="B" eighth="5" duration="SEMINIMA" attack="78" start="0.0" end="-27.0"
            position="135"/>
        <text string="c=111; s=15;Bo - a" heigth="339" position="172"/>
        <group size="2" position="178"/>
        <note sound="G" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="0.0"
            position="179"/>
        <note sound="B" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="0.0"
            position="208"/>
```

```
        <barline type="Single" position="228"/>
        <text string="c=101; s=15;noi -" heigth="339" position="233"/>
        <note sound="B" eighth="5" duration="SEMINIMA" attack="70" start="0.0" end="-27.0"
            position="245"/>
        <text string="c=101; s=15;te" heigth="339" position="278"/>
        <note sound="A" eighth="5" duration="SEMINIMA" attack="78" start="0.0" end="-27.0"
            position="287"/>
        <text string="c=101; s=15;meu" heigth="339" position="314"/>
        <group size="2" position="324"/>
        <note sound="B" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="11.0"
            position="325"/>
        <text string="c=010; s=20;Je - " heigth="339" position="342"/>
        <note sound="A" eighth="5" duration="COLCHEIA" attack="70" start="0.0" end="11.0"
            position="347"/>
        <barline type="Single" position="388"/>
        <text string="c=010; s=20;sus" heigth="338" position="426"/>
        <note sound="G" eighth="5" duration="MINIMA_DOT" attack="54" start="0.0" end="-27.0"
            position="436"/>
        <barline type="RepeatRight" position="533"/>
    </staf>
</music>
```

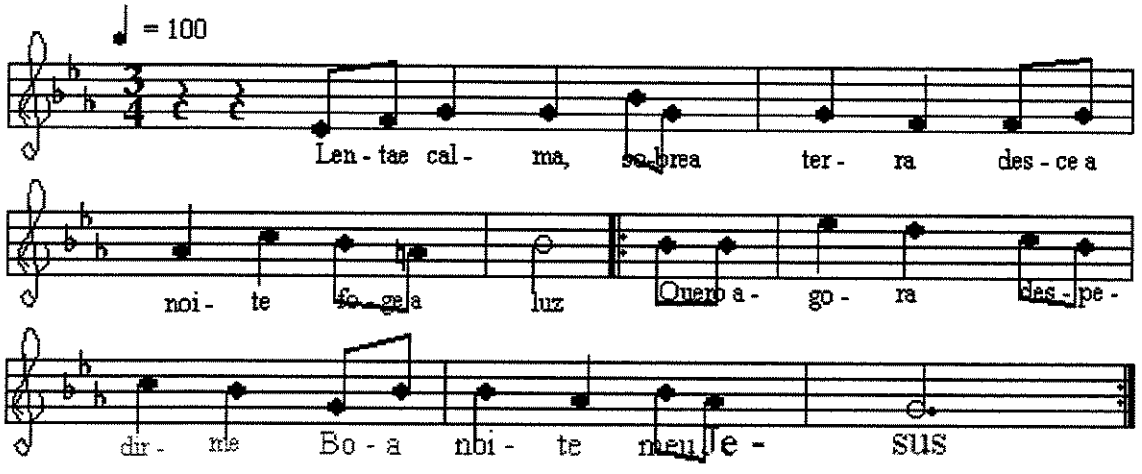


Figura 55: Partitura renderizada com o JavaMusic.

Boa Noite Meu Jesus - Autor desconhecido