Bibliotecas Java Aplicadas a Computação Musical

Leandro L. Costalonga, Evandro M. Miletto, Luciano V. Flores, Rosa M. Vicari

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{llcostalonga,miletto,rosa}@inf.ufrgs.br, lflores@cpovo.net

Abstract. *Today, there are many programming languages and libraries available for the development of computer music applications. This paper presents a comparison of Java technologies for computer music, as a quick reference for interested developers. The main available musical libraries are listed, described and compared.*

Resumo. *Atualmente existem dezenas de linguagens de programação e bibliotecas destinadas ao desenvolvimento de aplicativos para computação musical. Este artigo apresenta um estudo comparativo sobre tecnologias Java para computação musical visando servir como referência rápida para desenvolvedores que queiram utilizar essa tecnologia. As principais bibliotecas musicais disponíveis são relacionadas, descritas e comparadas.*

1. Introdução

Um dos primeiros passos em um projeto de software é a escolha da linguagem que melhor possa codificar os requisitos identificados. Neste ponto, características da linguagem devem ser criteriosamente observadas e ponderadas.

No paradigma orientado a objetos, onde o reuso é fortemente favorecido, deve-se observar, além das características nativas da linguagem, as opções de expansão da mesma através de bibliotecas, APIs (*Application Programming Interfaces*) ou *add-on*s de terceiros que possam facilitar o desenvolvimento. Neste quesito Java está muito bem amparada e, em se tratando de bibliotecas e ferramentas de auxílio para softwares musicais, não é diferente.

O fato de Java estar sendo amplamente utilizada na programação de aplicações musicais não significa que é a única ou a melhor linguagem para este tipo de programação. Entretanto, alguns fatores que fazem com que Java se destaque como linguagem de programação para softwares musicais são os mesmos que fizeram de Java uma linguagem bem sucedida em áreas mais tradicionais, dentre os quais se cita a robustez, portabilidade, facilidade de aprendizado, bom suporte pela indústria, fácil depuração, bem projetada, etc. Há, contudo, algumas características que fazem de Java uma das linguagens mais usadas para desenvolvimento de softwares musicais como, por exemplo, o fato de incluir uma biblioteca para tratamento de som e mensagens MIDI (Java Sound).

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

postura fez crescer o número de APIs disponíveis e, conseqüentemente, a dificuldade na escolha. Atualmente, muitos dos problemas iniciais do Java Sound já foram corrigidos.

Comparando Java com linguagens especialmente desenvolvidas para programação musical como Max/MSP, SuperCollider, Nyquist e KeyKit, pode-se afirmar que, por ser uma linguagem aberta, Java pode combinar música com outras funcionalidades da linguagem, como rede, gráficos e banco de dados.

Obviamente Java tem limitações, principalmente de performance, por ser uma linguagem tão completa e abrangente. Essas limitações motivam empresas e desenvolvedores independentes a criarem suas bibliotecas com soluções alternativas. Este trabalho pretende apresentar algumas das tecnologias disponíveis, facilitando a escolha e o reuso de componentes já criados por terceiros. Não é o objetivo deste trabalho ensinar ou esgotar todas as possibilidades de uso das ferramentas analisadas, e sim mostrar suas potencialidades.

O artigo está organizado como se segue: a seção 2 apresenta uma breve descrição de cada biblioteca analisada, ficando o estudo comparativo e classificatório para a seção 3. Uma breve conclusão é apresentada na seção 4.

2. Descrição das Bibliotecas

Foram analisados 14 bibliotecas que agregam ao Java a capacidade de trabalhar com áudio e dados musicais. São eles: Java Sound (Java Sound Resources, 2005), JMSL (Didkovsky e Burk, 2005), jMusic (Sorensen e Brown, 2005), Wire/Wire Provider (Wire, 2005; Gehnen, 2005), JavaMIDI (Marsanyi, 2005), NoSuch MIDI (NoSuch MIDI, 2005), MIDIShare (Grame, 2005), MIDI Kit (McNabb, 2005), jFugue (Koelle, 2005), Tritonus (Tritonus, 2005), JSyn (JSyn, 2005; JSynthLib, 2005), JScore (JScore, 2005), JASS (Doel, 2005) e Xemo (Project Xemo, 2003).

Procurou-se ordenar a apresentação das APIs por sua relevância (do ponto de vista dos autores), funcionalidade e data de criação. Todas as informações que constam na análise foram retiradas da documentação fornecida pelos fabricantes, o que nem sempre foi suficiente. Experimentos também foram realizados a fim de testar os códigos que constam nos exemplos. Infelizmente não há espaço suficiente no artigo para incluir código de exemplos e descrever em detalhes as APIs, mas tais informações são facilmente obtidas junto aos fabricantes.

Com base nos exemplos e na opinião própria dos autores (dada a subjetividade da questão), chegou-se à conclusão que as melhores APIs para trabalho com som (de maneira geral) são o JSyn, jMusic e Java Sound. Para trabalhar com MIDI, as melhores APIs são jMusic, JMSL e Java Sound. O destaque desta análise foi a jMusic que se mostrou completa, estável e de fácil uso em todas as categorias.

Este artigo é um resumo de um estudo muito mais extenso, realizado ao final de 2003, quando então as classificações e descrições básicas das bibliotecas foram elaboradas. As melhorias das novas versões vêm sendo acompanhadas e, quando significativas, são descritas neste artigo.

2.1. Java Sound (Versão 1.5)

Java Sound é uma API que provê suporte para operações de áudio e MIDI com alta qualidade, tais como: captura, mixagem, gravação, seqüenciamento e síntese MIDI. A Java Sound vem junto com o J2SE desde a versão 1.3 e por isso é considerada por alguns como a fonte do sucesso do Java para a programação musical. Na versão J2SE 1.5 teve grande avanço e correção e alguns de seus principais *bugs*, como a comunicação MIDI com dispositivos externos.

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

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

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

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

Java Sound suporta tanto áudio digital como MIDI. Ainda, permite que terceiros criem e distribuam componentes de software customizados para estender suas capacidades.

Java Sound pode ser usado na Web via *applets*, entretanto deve-se ficar atento às restrições de segurança definidas na classe *AudioPermission*. O padrão é que uma *applet* executando sobre restrições de segurança pode tocar, mas não gravar sons. Isso pode ser modificado caso o usuário permita.

Apesar das restrições, o Java Sound permite que as aplicações escritas com suas classes funcionem da mesma forma nas diversas plataformas suportadas pelo Java.

2.2. JMSL – Java Music Specification Language (Versão 1.03)

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

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

- Total integração com a linguagem (Java) e recursos da mesma, incluindo conectividade com banco de dados, ferramentas de rede, *servlets*, etc.;
- Comunicação direta com outras bibliotecas e dispositivos implementados em Java, como o JSyn, JavaMIDI, MidiShare e Java Sound;
- Incorporação do pacote JScore que permite edição da notação musical;
- Permite ao compositor distribuir as aplicações localmente ou através de *applets*;
- Gratuito em uma versão mais restrita (*Lite*).

Baseada no HMSL – *Hierarquical Music Specification Language*, o JMSL mantém a idéia original de seu predecessor ao apoiar-se no conceito de hierarquias ao desenvolver aplicações musicais, que nada mais são do que relacionamentos pai-filho.

Hierarquias podem ser previamente definidas ou criadas durante a execução de uma peça musical. Dois tipos de hierarquias podem ser criadas no JMSL: seqüencial e paralela. Hierarquizar, no contexto do JMSL, não significa apenas posicionar os objetos em uma visão *top-down*, definindo assim a importância dos "objetos" (entidades) no contexto musical. No JMSL, posicionar um objeto na hierarquia significa definir o momento em que o mesmo será executado. Desta forma, o objeto mais alto na hierarquia solicita aos seus "filhos" (objetos ligados diretamente a ele) que sejam executados todos juntos (hierarquia paralela) ou seqüencialmente (hierarquia seqüencial).

2.3. JMusic (Versão 1.4.2)

jMusic é uma biblioteca de programação musical de alto nível escrita em Java, desenvolvida na Universidade de Tecnologia de Queensland (Brisbane – Autrália). Assim como algumas outras linguagens e ambientes de programação, o jMusic foi projetado para ser usado por músicos e não programadores, com a função de auxiliá-los no processo composicional. Entretanto, muitos programadores fazem uso do jMusic como uma poderosa API para desenvolver aplicações musicais, em especial instrumentos digitais, ambientes de educação e análise musical.

Pode-se citar como vantagens do jMusic:

- Por ter sido escrito em Java, o jMusic consegue manter as principais virtudes da linguagem como sua flexibilidade e portabilidade, além de usar conhecimento prévio de Java no aprendizado do jMusic;
- JMusic é gratuito e aberto distribuído sobre a GNU General Public License;
- Fácil aprendizado e uso, uma vez que foi construído de acordo com as convenções musicais tradicionais;
- Material musical construído em outros programas e interfaces musicais pode ser importado ou exportado com facilidade;
- Grande variedade de ferramentas utilitárias para visualização e audição da composição em construção.

As informações musicais são organizadas e armazenadas tal qual a pauta em papel, ou seja, a partitura é formada de partes, que por sua vez, é constituída de frases musicais onde estão contidas as notas. A nota é a estrutura básica usada no jMusic e traz consigo uma série de atributos, como: altura, volume, figura de tempo, controle de estéreo, duração e os acidentes (sustenido e bemol). Frase pode ser vista como vozes de uma parte – por exemplo, no piano cada mão tocaria uma voz. A frase só tem um atributo realmente importante, a lista de notas.

jMusic pode ser visto como um código para descrever música. A música pode ser representada em diversas notações diferentes levando-se em consideração aspectos como: conhecimento musical do usuário, quantidade de frases e partes musicais, objetivo do software em desenvolvimento, etc. A forma mais comum de representar a música é através na da notação clássica, denominada no jMusic como *Common Practice Notation* (CPN). Atualmente, a implementação da CPN permite que somente frases musicais sejam exibidas e outras pequenas operações como: salvar em MIDI, modificar notas e formulas de compasso, tocar, etc. Apesar de bastante útil, a CPN possui restrições de implementação e possui difícil leitura para leigos em música. Uma das maiores desvantagens é fato de não consegui exibir duas frases ao mesmo tempo em uma única janela. Uma notação que vem ajudar a transpor estas restrições é a *ShowScore*, um ponto intermediário entre a CPN e a *Piano Roll* (*Scratch*).

O jMusic possui um completo pacote de áudio que envolve síntese sonora, construção de instrumentos virtuais e uso de *samples* em composições.

Instrumentos virtuais são classes usadas para renderizar partituras do jMusic em arquivos de áudio (ou saída de áudio em tempo real) e são feitos a partir de objetos de áudio organizados em uma estrutura hierárquica denominada "corrente" (*chain*). Estas correntes de objetos de áudio definem todas as propriedades do som que se ouve e o tipo de síntese a ser utilizada para gerá-lo.

A organização hierárquica de objetos de áudio (e música) não é algo recente. Uma das primeiras linguagens de programação para música, Music V, escrita por Max Mathews nos Laboratórios da Bell – EUA, estabelecia a convenção de usar diagramas de fluxos de sinais. Outra opção dos instrumentos virtuais é usar amostras de som (*samples*). Nesta técnica, pequenos arquivos de som são gravados e definidos como notas que podem ter sua freqüência (*pitch*) e duração modificadas e seqüenciadas para gerar uma composição.

2.4. Wire/Wire Provider (Versão 0.97)

As primeiras versões do Java Sound não conseguiam trocar mensagens MIDI com dispositivos externos, o que só foi resolvido na versão JDK1.4.1 (em 2003). Para resolver este problema, alguns fabricantes e programadores disponibilizaram bibliotecas que permitem tal comunicação, porém muitas não eram escritas em Java, o que limita a portabilidade das aplicações que as usam.

Duas versões do Wire foram disponibilizadas. A primeira chama-se simplesmente Wire e foi escrita por uma pequena empresa alemã chamada Bonneville. É um exemplo típico de JNI (*Java Native Interface*), ou seja, classes escritas em C++ e aproveitadas pelo Java, no Windows. A outra versão foi escrita por Gerrit Gehnen e foi baseada na versão de Niel Gorisse (Bonneville) e no pacote para Linux da Tritonus.

2.5. JavaMIDI (Versão 5)

JavaMIDI é um conjunto de classes, escritas por Robert Marsanyi, que permite uso do MIDI sobre a plataforma Java assim como o Wire e o Wire Provider com a diferença de executar sobre Windows e Macintosh.

2.6. NoSuch MIDI (Versão Única)

Esta biblioteca e capaz de lidar com dispositivos MIDI externos, mensagens exclusivas, escalonamento em tempo real da saída, escrita e leitura de MIDI. Não precisa do Java Sound por ser uma implementação independente, o que a faz não executar em *browsers*. É gratuita para uso não comercial.

2.7. MIDI Share (Versão 1.9)

Feita a um tempo em que não existia suporte a MIDI no Java, mas sobreviveu ao Java Sound por não apresentar seus problemas com dispositivos MIDI externos e sobressaiu-se das demais bibliotecas similares por rodar em *applets.* Entretanto, é necessário instalar dois arquivos (bibliotecas nativas) no cliente JMidi e JPlayer.

2.8. MIDI Kit (Versão 1.0)

Esta biblioteca pode ser uma boa alternativa para quem precisa de portabilidade nas aplicações MIDI. Com ela é possível desenvolver com facilidade aplicações MIDI simples e locais, bem como servidores de processamento MIDI distribuído em rede que se configura dinamicamente para atender as necessidades de cada cliente.

Da mesma forma que outras bibliotecas que usam rotinas não escritas em Java, o MIDI Kit necessita que arquivos sejam postados no cliente para funcionar em *applets*.

Apesar do nome, o MIDI Kit também consegue processar áudio. Até o momento, somente o processamento de arquivos de som e execução dos mesmos estão disponíveis. O modo de funcionamento do MIDI Kit é baseado na arquitetura do NeXT Music Kit (Stanford – CCRMA, 1998).

2.9. JFugue (Versão 1.0)

JFugue é um conjunto de classes Java para programação musical. Esta biblioteca usa simples *strings* (cadeias de caracteres) para representar dados musicais, incluindo notas, acordes e mudanças de instrumentos. JFugue também permite a definição de música através de padrões que podem ser transformados para criar novos segmentos musicais derivados de peças musicais já existentes. Atualmente na versão 2.1, o JFugue já inclui suporte a controladores MIDI e melhorou o *parse* interno dos acordes, aumentando a gama de acordes trabalháveis.

2.10. Tritonus (Versão 1.x)

Tritonus é uma implementação com o código aberto do Java Sound 1.0 para Linux. É distribuída de acordo com os termos da GNU Library General Public License.

Tritonus é mais estável que o Java Sound, mas ainda não está completo e perfeito. Apesar de sua limitação de plataforma (somente Linux), o Tritonus é quase uma unanimidade entre aqueles que programam em Linux.

Existe uma relação unilateral do Java Sound em relação ao Tritonus, ou seja, o que for feito para o Java Sound, através do pacotes de SPI, deve funcionar no Tritonus.

Tendo em vista a estreita relação entre Tritonous e Java Sound, a seguir algumas funcionalidades são apresentadas:

- Suporta leitura e gravação de arquivos no formato .au, .aiff e .wav, assim como Java Sound, mas diferentemente deste não possui pacotes para *Service Providers* estenderem sua capacidades, tendo os mesmos que trabalhar direto no código fonte;
- Uma outra diferença do Java Sound é o fato do Tritonus não possuir um *mixer*, tendo que buscar algum disponível no sistema. Isto facilita a integração do código com a infra-estrutura de hardware da máquina mas também dificulta a criação de novos dispositivos de áudio, que têm que prover implementações para as interfaces Mixer, SourceDataLine, TargetDataLine e Clip;
- Standard linear: mono e estéreo, big e little endian, 8, 16, 24 e 32 bits, codecs A-law e mu-law;
- MP3 decoder (Javalayer 0.0.7 incorporado ao Tritonus, Java MP3 Player Project criação de MP3 usando a biblioteca LAME). O Java Sound não provém mais suporte a MP3 por problemas autorais.

As partes principais do suporte ao MIDI estão implementadas baseadas no seqüenciador ALSA. Uma implementação baseada no MIDI Share está em desenvolvimento.

- Escrita e leitura de arquivos MIDI;
- Um sistema de síntese de software (TiMidity) não muito estável;
- Interface para sintetizadores em hardware (com restrições);
- MIDI IN/OUT (acesso a dispositivos MIDI externos).

2.11. JASS (Real-time Audio Synthesis - Versão 2.x)

JASS (Java Audio Synthesis System) é uma unidade geradora baseada em ambientes para programação de síntese de áudio. Escrita em Java puro, o ambiente baseia-se em um pequeno número de interfaces e classes abstratas que implementam a funcionalidade necessária pra criação de *patches*. *Patches* são criados juntando unidades geradoras em complexas estruturas e podem renderizar sons em tempo real. A comunicação com o hardware de áudio foi escrita com o Java Sound e, em algumas plataformas, JNI (*Java Native Interface*).

A biblioteca se propõe a ser uma alternativa ao Java Sound com baixa latência devido a métodos nativos. Atualmente possui implementação para Linux (ALSA e OSS), Macintosh (OS/X) e Windows (DirectX, ASIO *blocking* API, ASIO *callback* API). Todas as implementações, exceto ASIO *callback*, utilizam uma classe de entrada/saída de áudio em tempo real escrita em C++ por Gary P. Scavone. JASS tem o código fonte aberto e está disponível para uso não comercial.

Atualmente a biblioteca está na versão 2.012 e é atualizada constantemente. Em 2004 lançou um plug-in para IDE Eclipse, o que facilitou o desenvolvimento de

programadores terceiros. Entre as inovações da versão 2.x está o *Trace-Assertion* que ajuda a especificar o comportamento dinamico do sistema em tempo de execução. Essa funcionalidade, antes obrigatória, passa a estar desabilitada por *default* na última versão lançada.

2.12. JSyn (Versão 14.2)

JSyn permite o desenvolvimento de programas Java para computação musical. Pode-se rodar como aplicações *stand-alone* ou como *applets* em *webpages* (usando o *plug-in*). Escrito em C para prover síntese em tempo real, o JSyn pode ser usado para gerar efeitos de som, ambientes de áudio ou música. JSyn se baseia no tradicional modelo de unidades geradoras que juntas podem gerar sons complexos. Seguem as principais características:

- Síntese em tempo real e de alta fidelidade usando a CPU;
- Biblioteca de unidades geradoras incluindo osciladores, filtros, envelopes, geradores de ruídos e efeitos;
- Todas a operações usam precisão de 32 bits;
- Pode-se combinar *samples* e sons sintetizados em tempo real;
- Fácil uso das classes Java para criar, conectar e controlar as unidades geradoras;
- Uso de *time-stamping* para programação de eventos no tempo;
- Uso de fila de *samples* e dados do envelope para programar repetição e colagem;
- Suporte a entrada de áudio para gravação e processamento de voz;
- Suporte para dispositivos multi-canal;
- Suporte para plataformas Windows, Macintosh e Linux;
- Editor gráfico (Wire) que permite gerar sons conectando unidades geradoras inteiramente, podendo exportar o código Java resultante;
- SDK e *plug-in* gratuitos, para uso não comercial.

2.13. JScore (Acompanha o JMSL 1.03)

JScore é um analisador sintático em Java para dados de música, que gera as partituras no formato XML e o respectivo código MIDI. O primeiro objetivo do JScore era demonstrar potencialidade do JLex (gerador de analisadores sintáticos), integrado ao JMusic que descreve a música no modo que o JLex deve entender. Atualmente o JScore integra o JMSL.

2.14. Xemo

Xemo é um projeto que visa desenvolver um *framework* para composição e notação musical. Em princípio o projeto não tem relação com qualquer tecnologia, ou seja, apenas define o que tem que ser feito. A API define interfaces para layout e renderização de símbolos musicais em alta resolução em dispositivos 2D, incluindo monitores e impressoras. Com isto, pode-se desenvolver programas de composição interativa, editores de notação musical, jogos e softwares educacionais.

Atualmente, a API é simplesmente um conjunto de classes gráficas vazias, ou seja, não possuem nenhuma funcionalidade para armazenar ou ler arquivos de áudio ou MIDI ou mesmo executar o que foi escrito.

A API contém funcionalidades específicas para representação musical, execução e composição interativa. Seus elementos base são pacotes para notação musical, representação de estruturas musicais e execução, e performance via MIDI. Apesar de terem sido lançados alguns pacotes, o projeto foi aparentemente descontinuado.

3. Comparativo

A seguir é apresentado um quadro comparativo entre as APIs analisadas dentro das classificações sugeridas.

3.1. Processamento de Áudio

Esta categoria caracteriza-se pela capacidade das APIs em processarem dados de áudio, de forma genérica. As funcionalidades desejáveis podem ser vistas na Tabela 1.

Tabela 1. Comparação entre APIs que processam áudio.

Critério/API	Java Sound	jMusic	MIDI Kit	JSyn	Tritonus
Gravação e reprodução de áudio	X	X	X	X	X
Conversão e suporte a diferentes formatos de áudio	X	X		X	X
Manipulação de *samples* (“samplear”)		X		X	
Manipulação e/ou processamento de dados de áudio	X	X			X

3.2. Síntese de Áudio

Esta categoria caracteriza-se pela capacidade das APIs em gerar/modificar sons baseando-se em algoritmos de síntese, de forma genérica. As funcionalidades desejáveis podem ser vistas na Tabela 2.

Tabela 2: Comparação entre APIs que sintetizam áudio.

Critério/API	JSyn	JASS	JMusic
Geração de áudio a partir da conexão de unidades geradoras (osciladores, envelopes, amplificadores etc.)	X	X	
Geração de áudio modificando parâmetros físicos das ondas sonoras		X	
Uso de instrumentos virtuais com parâmetros de áudio pré-estabelecidos e devidamente encapsulados	X		X
Suporte a diversos tipos de síntese, como *wavetable*, aditiva, subtrativa, FM, etc.	X	X	X

Vale lembrar que o JSyn integra o pacote do JMSL.

3.3. Seqüenciamento MIDI

Esta categoria caracteriza-se pela capacidade das APIs em escalonar eventos MIDI no tempo. As funcionalidades desejáveis podem ser vistas na Tabela 3.

Tabela 3: Comparação entre APIs que seqüenciam eventos MIDI.

Critério/API	Java Sound	JMusic	JMSL	MIDI Share	Tritonus
Possuir seqüenciador virtual	X			X	
Possuir mecanismo que permita programar/escalonar ações MIDI para serem executadas em determinado momento de tempo		X	X		X
Possuir mecanismo de sincronização	X		X	X	X
Leitura de gravação de arquivos MIDI	X	X	X	X	X
Controles da execução da música (andamento, posicionamento no tempo, dinâmica)	X	X	X	X	X
Controle dos canais (dinâmica, *mute*, solo, timbre)	X	X	X	X	X

3.4. Comunicação MIDI com Dispositivos Externos

Esta categoria caracteriza-se pela capacidade das APIs em enviar eventos MIDI a dispositivos externos. As funcionalidades desejáveis podem ser vistas na Tabela 4.

Tabela 4: Comparação entre APIs que trabalham com dispositivos externos.

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

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

3.5. Síntese a Partir de MIDI

Na classificação adotada por este trabalho, este item diferencia-se da síntese de áudio (Sub-seção 3.2), a qual permite também o uso de áudio como entrada, modificando-o.

Esta categoria caracteriza-se pela capacidade das APIs em gerar sons a partir de eventos MIDI. As funcionalidades desejáveis podem ser vistas na Tabela 5.

Tabela 5: Comparação entre APIs que geram sons a partir de eventos MIDI.

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

3.6. Componentes Gráficos

Esta categoria caracteriza-se pela capacidade das APIs em exibir informações musicais, em diversas notações, através das interfaces gráficas. As funcionalidades desejáveis podem ser vistas na Tabela 6.

Tabela 6: APIs que exibem informações musicais em interfaces gráficas.

Critério/API	JScore	Xemo	JMusic
Exibição de dados MIDI em notação musical tradicional	X	X	X
Exibição de dados MIDI em notação musical alternativa			X
Funcionalidades de execução musical integradas	X		X

3.7. Representação Musical

Esta categoria caracteriza-se pela capacidade das APIs em trabalhar com outros formatos de arquivos usados para armazenar informações musicais. As funcionalidades desejáveis podem ser vistas na Tabela 7.

Tabela 7: APIs que trabalham com formatos de arquivos musicais.

Critério/API	JFugue	Xemo
Permite codificação de dados musicais em formatos ortodoxos, diferentes do convencional MIDI	X	X
MusicXML		X

3.8. Programação Musical (Composição)

Uma API musical pode ser usada em softwares destinados a diversos fins, entretanto possuem maior vocação para alguma atividade musical. Esta categoria é composta pela APIs criadas para uso em composição musical. As funcionalidades desejáveis podem ser vistas na Tabela 8.

Tabela 8: APIs criadas para uso em composição musical.

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

*MusicShape Editor

4. Conclusões

Este trabalho apresentou e analisou ferramentas que agregam ao Java a capacidade de trabalhar com áudio e dados musicais. Foram vistas características, arquiteturas e exemplos destas bibliotecas, que foram classificadas em 8 categorias distintas.

Chegou-se à conclusão que as melhores APIs para trabalho com som (de maneira geral) são o JSyn, jMusic e Java Sound. Para trabalhar com MIDI, as melhores APIs são jMusic, JMSL e Java Sound. O destaque deste trabalho foi a jMusic que se mostrou completa, estável e de fácil uso em todas as categorias. Certamente o usuário deve considerar a natureza da sua aplicação para optar por uma ou outra biblioteca.

Não se espera com este trabalho esgotar ou abranger todas as bibliotecas para programação musical em Java, somente apresentar as principais bibliotecas, visto o grande dinamismo com que são criadas diariamente.

Referências

Didkovsky, N.; Burk, P. (2005) "Java Music Specification Language", http://www.algomusic.com/jmsl/, ago. 2005.

Doel, K. (2005) "Real-time Audio Synthesis using JASS", http://www.cs.ubc.ca/~kvdoel/jass/, ago. 2005.

Gehnen, G. (2005) "Java and Midi Programs by Gerrit Gehnen", http://www.geocities.com/ggehnen/, ago. 2005.

Grame (2005) "Recherche", http://www.grame.fr/Recherche/, ago. 2005.

Java Sound Resources (2005) http://www.jsresources.org/, ago. 2005.

JScore (2005) http://homepages.nyu.edu/~ray208/Jscoring/html/JScore.html, ago. 2005.

JSyn (2005) "Java Audio Synthesis", http://www.softsynth.com/jsyn/, ago. 2005.

JSynthLib (2005) "JSynthLib Home Page", http://www.jsynthlib.org/, ago. 2005.

Koelle, D. (2005) "JFugue - Java API for Music Programming", http://www.jfugue.org/, ago. 2005.

Marsanyi, R. (2005) "JavaMIDI", http://www.softsynth.com/javamidi/ , ago. 2005.

McNabb, M. (2005) "Fantasia and The MIDI Kit", http://www.mcnabb.com/software/fantasia/, ago. 2005.

NoSuch MIDI. (2005) http://www.nosuch.com/nosuchmidi/, ago. 2005.

Project Xemo (2003) http://www.xemo.org/, set. 2003.

Sorensen, A.; Brown, A. (2005) "jMusic: Music Composition in Java", http://jmusic.ci.qut.edu.au/, ago. 2005.

Tritonus (2005) "Open Source Java Sound", http://tritonus.org/, ago. 2005.

Wire (2005) http://www.bonneville.nl/software/Wire/, ago. 2005.