Andante - Composição e Performance Musical Utilizando Agentes Móveis

Leo Kazuhiro Ueda*, **Fabio Kon**

Departamento de Ciência da Computação
Instituto de Matemática e Estatística (IME) – Universidade de São Paulo
http://gsd.ime.usp.br/andante

{lku,kon}@ime.usp.br

Novembro de 2003

Abstract. *Across the Centuries, musicians have always had interest in the latest scientific achievements and have used the latest technologies of their time to produce musical material. Since the mid-20th Century, the use of computing technology for music production and analysis has been increasingly common among music researchers and composers. Continuing this trend, in recent years, the use of network technologies in the field of Computer Music turned out to be a natural research goal. Our group is investigating the use of mobile agent technology for the creation and performance of music within a distributed computing environment. We believe that this technology has the potential to foster new ways of composing, distributing, and performing music.*
In this paper, we describe a prototype implementation of Andante, an open-source infrastructure for the construction of distributed applications for music composition and performance based on mobile musical agents. We also describe two sample applications built on top of this infrastructure.

Resumo. *Ao longo da história da música, compositores sempre manifestaram interesse pelas mais recentes descobertas científicas e freqüentemente utilizaram as tecnologias mais avançadas de seu tempo para produzir material musical. Desde meados do século XX, o uso de tecnologias computacionais para a produção e análise musical tem se tornado cada vez mais freqüente entre pesquisadores e compositores. Nos últimos anos, o uso de tecnologias de redes e de sistemas distribuídos na Computação Musical se tornou um objeto natural de pesquisa. Nosso grupo tem investigado o uso da tecnologia de agentes móveis para a criação e execução de música em ambientes computacionais distribuídos. Acreditamos que esta tecnologia tem o potencial de promover novas formas de composição, distribuição e performance musicais.*
Neste artigo, descrevemos a implementação de um protótipo do sistema Andante, uma infra-estrutura de código aberto para a construção de aplicações distribuídas para composição e performance musical baseadas em agentes móveis musicais. Descrevemos também duas aplicações de exemplo construídas utilizando esta infra-estrutura.

*Financiado pela CAPES.

1. Introdução

Compositores têm sempre acompanhado as descobertas e inovações da ciência para criar novas formas de se fazer música. A própria música tradicional foi sendo transformada à medida que novos instrumentos, ou novas formas de se produzir som, foram sendo explorados.

Houve, especialmente no século XX, uma intensificação dessa relação entre música e ciência, principalmente a partir da década de 50, com a música eletroacústica e o uso de computadores. Mais recentemente, devido ao surpreendente avanço da computação, passou a ser possível equipar computadores pessoais com dispositivos relativamente baratos para sintetizar e reproduzir som de alta qualidade. Neste período, desenvolveu-se um vasto leque de técnicas e modelos para composição e execução de música usando computadores [Miranda, 2001, Roads, 1996]. As tecnologias de rede e principalmente a Internet também trouxeram novas possibilidades para produção de música [Kon and Iazzetta, 1998].

Seguindo essa tendência, apresentamos aqui o projeto *Andante*. O desenvolvimento do potencial do modelo de *agentes móveis* em Ciência da Computação é recente, por isso estamos interessados em investigar as aplicações do conceito de agentes móveis na criação de novas formas de composição e performance musical.

O sistema Andante fornece uma infra-estrutura que permite a construção de aplicações distribuídas baseadas em *Agentes Móveis Musicais* para criação, distribuição e execução de música. Usando o Andante, programadores podem implementar seus próprios agentes ou utilizar os agentes básicos da plataforma para construir tais aplicações. Estamos também iniciando colaborações com músicos e pesquisadores visando o desenvolvimento de obras artísticas distribuídas utilizando essa infra-estrutura.

Em [Ueda and Kon, 2003], fizemos uma breve introdução do projeto Andante apresentando as implementações iniciais da infra-estrutura e de uma aplicação de exemplo. Neste artigo, descrevemos em maiores detalhes a arquitetura e a implementação da infra-estrutura, incluindo reformulações e extensões incorporadas nos últimos meses e uma nova aplicação mais sofisticada. Antes, na Seção 2, descrevemos o paradigma de agentes móveis no contexto de sistemas distribuídos para na Seção 3 definirmos o conceito de agente móvel musical. Em seguida, na Seção 4, fazemos uma sucinta descrição de alguns tópicos em computação musical que podem ser explorados pelo sistema Andante. A Seção 5 descreve a implementação da infra-estrutura e a Seção 6 mostra as duas aplicações construídas sobre a infra-estrutura. Na Seção 7 concluímos discutindo os próximos passos da nossa pesquisa.

2. Agentes Móveis

Um agente móvel é um programa de computador capaz de migrar de uma máquina para outra através de uma rede. No processo de migração, o agente suspende sua execução na máquina em que se encontra, gera uma representação do seu estado interno, tem seu código e estado transferidos através da rede para uma outra máquina e, finalmente, continua a execução com o estado que possuía no momento da suspensão [Kotz and Gray, 1999]. Além dessa característica, um agente tem uma certa autonomia,

podendo reagir a mudanças no ambiente onde está sendo executado e decidir por si só o que fazer e quando migrar para outra máquina.

Esse conceito, introduzido no início da década de 90 [Johansen et al., 1994], trouxe um novo paradigma para construção de sistemas computacionais distribuídos e móveis. Mais recentemente, a partir do final da mesma década, trabalhos concretos têm surgido. Já existem várias plataformas de agentes móveis implementadas [Johansen et al., 1995, Johansen et al., 2002, Gray et al., 1998, Lange and Oshima, 1998].

O paradigma de agentes móveis pode trazer uma série de vantagens em relação aos modelos tradicionais de computação distribuída (como os modelos cliente/servidor e de objetos distribuídos baseados em chamadas remotas) [Lange and Oshima, 1999]. Algumas das vantagens são comentadas a seguir.

- *Redução do tráfego na rede e eliminação da latência*: sistemas distribuídos utilizam protocolos de comunicação que geralmente envolvem muitas trocas de mensagens pela rede. Com agentes móveis, entretanto, ocorre a migração do código para o nó destino e, dessa forma, as interações são realizadas localmente. O tráfego é reduzido às migrações dos agentes, a latência das comunicações pela rede desaparece.
- *Execução autônoma e assíncrona*: ao migrar para uma máquina, um agente móvel pode se tornar independente da aplicação que o criou, podendo executar uma tarefa de forma autônoma e assíncrona. Não é necessário manter conexões abertas entre as duas partes e, por isso, sistemas que dependem de conexões frágeis podem se beneficiar dessa característica.
- *Adaptação dinâmica*: agentes móveis podem receber informações sobre o ambiente onde está sendo executado e reagir a elas.

Além das vantagens técnicas, o paradigma em si já traz um benefício conceitual, pois permite a concepção de sistemas inovadores e abordagens diferentes para problemas conhecidos.

3. Agentes Móveis Musicais

Um agente móvel musical (ou simplesmente *agente*, para simplificar) é um agente móvel que participa de uma atividade de processamento de música. Ele pode fazer isso executando uma ou mais das seguintes ações.

Encapsular um algoritmo de composição: um agente pode implementar um algoritmo de composição e carregar possíveis dados de entrada para este, o que permite que ele gere música autonomamente.

Interagir e trocar informações com outros agentes musicais: numa analogia com músicos tocando num palco real, agentes numa mesma máquina podem interagir e comunicar uns com os outros trocando informações musicais.

Interagir com músicos humanos: um agente pode receber comandos ou sons gravados por um músico humano. Os comandos podem ser, por exemplo, notas tocadas num teclado MIDI (que seriam reproduzidas pelo agente) ou novos parâmetros para o algoritmo de composição executado pelo agente. O som de um instrumento

acústico também pode ser enviado para um agente e processado e/ou reproduzido por ele.

Reagir a sensores: os agentes também podem receber comandos de sensores, permitindo que reajam a certos eventos como, por exemplo, os movimentos de uma bailarina em um palco ou a movimentação do público em um museu.

Migrar: o processo de migração pode ser acionado pelas ações descritas acima. Em outras palavras, um agente pode decidir migrar

- estocasticamente ou deterministicamente, baseado em um algoritmo;
- baseado na interação com outros agentes;
- baseado na interação com músicos humanos;
- em resposta a sinais de sensores.

Como um agente móvel normal, um agente musical pode continuar a sua performance ao chegar ao seu destino.

Utilizando agentes móveis musicais de vários tipos, podemos construir sistemas de composição e performance musical. Vejamos alguns exemplos.

1. *Geração de música estocástica:* Nesse sistema, vários agentes encapsulam algoritmos estocásticos de geração de melodia. Um dos agentes funciona como um metrônomo, fornecendo para todos os outros agentes os tempos do compasso no momento certo. O resultado da execução desses vários agentes numa mesma máquina é a geração de música sincronizada.
2. *Performance distribuída:* Cada músico humano é representado por um ou mais agentes. Cada agente recebe a música gerada pelo músico e a reproduz em tempo real no palco em que se encontra. Dessa forma, um músico pode estar virtualmente presente em mais de um palco. Agentes desse tipo são utilizados para a realização de uma performance distribuída: cada músico numa localidade recebe em seu palco agentes que representam os outros músicos e também envia agentes que o representam para os palcos dos outros músicos.
3. *Sistema musical interativo:* Cada agente recebe informações musicais de outro agente ou de um instrumento musical tocado por um humano. O agente então processa essas informações e gera música baseada no processamento ou repassa a informação processada para outro agente.
4. *Música distribuída:* Considere um museu ou uma sala de exibição onde se encontram vários computadores ligados em rede. Cada computador é equipado com sensores de movimento e hospeda alguns agentes. Os agentes geram uma música distribuída de uma maneira sincronizada. Um dos agentes recebe informações geradas pelos sensores de movimento, de modo que ele possa seguir pessoas que estejam andando pela sala (usando a capacidade de migrar). A sensação do ouvinte é de que parte da música o está seguindo. Da mesma maneira, é possível fazer com que uma outra parte da música sempre se afaste do ouvinte. Sons gerados pelo público (por exemplo, frases) podem ser dinamicamente incorporadas à paisagem musical gerada pelo sistema.

Compositores e pesquisadores envolvidos em música contemporânea no Brasil e no exterior já manifestaram grande interesse em estudar as possibilidades de aplicação do paradigma de agentes móveis à música. A colaboração destes músicos nestas pesquisas será essencial para vislumbramos as reais possibilidades dessa nova tecnologia.

Enquanto isso, os exemplos acima já trazem à tona diversos problemas técnicos a serem superados. Em particular, questões de *tempo real*, *latência* e *qualidade de serviço* são importantes para a geração final do som. O suporte de agentes móveis para essas questões é um assunto ainda não estudado e que pretendemos explorar no futuro.

4. Computação Musical

Agentes móveis musicais podem ser explorados em três sub-áreas da computação musical: composição algorítmica, sistemas musicais interativos e música na Internet. A seguir, abordamos sucintamente cada uma destas sub-áreas levando-se em consideração a definição de agentes da Seção 3.

4.1. Composição Algorítmica

Há séculos, compositores trabalham com a idéia de que certos aspectos da composição musical podem ser formalizados em procedimentos bem definidos. A composição algorítmica consiste basicamente no uso de processos formais para a criação de música [Roads, 1996, Loy, 1989, Hiller, 1970].

O método de Guido d'Arezzo para compor cantos como acompanhamentos de textos foi desenvolvido por volta do ano de 1026; é o registro mais antigo de uma formalização nesse sentido [Roads, 1996]. Neste método, para criar uma melodia a partir de um texto, cada sílaba do texto é associada a uma nota de acordo com a vogal da sílaba e com uma tabela que relaciona notas e vogais.

Até o aparecimento dos computadores, muitos outros métodos foram desenvolvidos. Talvez o exemplo histórico mais conhecido de algoritmo de composição seja o *Musikalisches Würfelspiel* de Mozart. Trata-se de um jogo onde um minueto é composto juntando-se pequenos fragmentos musicais pré-definidos. Esses fragmentos são escolhidos ao acaso jogando-se dados [Roads, 1996].

No século XX, ainda antes do computador, a tendência era o uso de procedimentos matemáticos e estatísticos na composição, e também de equipamentos eletrônicos. O uso de computadores a partir da década de 50, como ocorreu com diversas áreas, trouxe novas possibilidades à composição algorítmica. É importante notar que a aplicação de computadores na música acompanhou a evolução dos computadores praticamente desde a sua criação. Na verdade, pela própria característica de automação da composição algorítmica, o uso de máquinas anteriores ao computador já era comum.

Lejaren Hiller foi um pioneiro na computação musical e a composição algorítmica era um dos seus interesses principais. No meio da década de 1950, quando ele iniciou seus experimentos, era comum os grandes centros de pesquisa projetarem e construírem as suas próprias máquinas. Entre 1955 e 1956, Hiller e Leonard Isaacson utilizaram o computador Illiac da Universidade de Illinois em Urbana-Champaign para criar a que é normalmente considerada a primeira composição gerada por um computador: a *Illiac Suite* para quarteto de cordas. O papel do computador foi gerar a partitura a partir de processos algorítmicos, que foi então traduzida para notação musical tradicional.

Muitos outros trabalhos pioneiros seguiram os de Hiller. Para mais informações e referências sobre a história da composição algorítmica, veja [Roads, 1996, Loy, 1989, Hiller, 1970].

Na Seção 6.1 descreveremos uma implementação de um sistema estocático de composição algorítmica nos moldes do Exemplo 1 da Seção 3. Cada agente que compõe o sistema carrega um algoritmo que utiliza simulações de ruídos $\frac{1}{f^{\beta}}$ para gerar parâmetros de uma melodia. Veja a descrição do *algoritmo fractal* em [Roads, 1996] para maiores informações sobre esta técnica.

4.2. Sistemas Musicais Interativos

Embora o surgimento do computador tenha causado impacto, grande parte dos trabalhos iniciais em composição algorítmica buscava a semelhança com a música feita na época. Isto é, os sons produzidos tentavam seguir as tendências correntes, sendo o computador uma ferramenta com pouca influência estética sobre o material produzido.

Num cenário mais recente, o computador passa a inovar a forma de se produzir, executar e até de se escutar música. Os sistemas musicais interativos são uma evidência dessa nova tendência. Nesses sistemas, o computador "ouve" e reage à música que é produzida em sua volta.

Segundo Rowe [Rowe, 1993], sistemas musicais interativos são aqueles cujo comportamento muda em resposta ao recebimento de informações musicais. Esta classe de sistemas tem sido muito explorada pela música erudita na última década. Observa-se uma profusão de obras musicais onde o computador aparece com um elemento que gera música baseada na interação com outros músicos, dançarinos, atores e até mesmo com o público [Iazzetta, 2003].

A autonomia dos agentes e a sua capacidade de interagir com o ambiente no qual está inserido e reagir a estímulos dele provenientes faz dos agentes musicais um mecanismo particularmente apropriado para a construção de sistemas musicais interativos.

4.3. Música na Internet

Uma conseqüência natural do crescimento expressivo da Internet e do acesso cada vez mais comum de computadores pessoais a recursos de multimídia foi o surgimento do interesse em sistemas multimídia distribuídos. Mas devido à característica estritamente temporal da música e o grande volume de dados de que sua representação digital necessita, as dificuldades envolvidas em sua implementação são numerosas [Kon and Iazzetta, 1998].

O sistema *NetSound* [Casey and Smaragdis, 1996] tenta resolver o problema da largura de banda utilizando uma representação alternativa do som. Em vez de enviar o som digitalizado pela rede, o *NetSound* envia uma descrição matemática das ondas do som, que é reconstruído com alta qualidade na máquina de destino. Em ambientes onde as exigências musicais não são tão estritas, também são comuns sistemas como o *RealAudio* (`http://www.real.com`), que utilizam áudio comprimido com qualidade reduzida.

O projeto *Global Visual-Music* (`http://visualmusic.org/gvm.htm`) está desenvolvendo uma infra-estrutura para a realização de performances com improvisação através da distribuição de conteúdo multimídia em tempo real para vários pontos.

À medida que os problemas de largura de banda e atraso da rede são equacionados, os sistemas de música na Internet caminham em direção à interação com o usuário e à possibilidade de colaboração entre diversos usuários. O sistema Andante é um reflexo dessa tendência.

5. Andante

Apresentamos a seguir uma descrição de nossa implementação do Andante abordando as tecnologias utilizadas, a arquitetura do sistema e detalhes de implementação.

5.1. Tecnologias Utilizadas

Toda a implementação foi feita na linguagem Java devido aos seguintes motivos.

- *Independência de plataforma*: esperamos que o sistema seja utilizado por programadores, compositores e instrumentistas. Necessitamos então que o sistema possa ser executado em ambientes de hardware e software distintos, como os baseados em Linux (muito utilizado por programadores), MacOS (muito utilizado por compositores) e Windows (muito utilizado por instrumentistas). Java parece ser, no momento, a melhor alternativa para a construção de sistemas que executem facilmente nestas três plataformas.
- *Java Swing*: Java oferece uma sólida biblioteca para construção de interfaces gráficas. O nosso interesse está em criar rapidamente interfaces independentes de plataforma e que ofereçam um alto grau de interatividade com o usuário.
- *Suporte a multimídia*: embora a implementação oficial da *Java Sound API* (`http://java.sun.com/products/java-media/sound`) ainda não esteja no nível desejado, com ela foi possível construir um protótipo da infra-estrutura em pouco tempo.

Apesar de termos apenas usado Java, pretendemos no futuro permitir que partes do sistema sejam implementadas em outras linguagens. O principal motivo para isso é possibilitar o uso de outras tecnologias de geração de som, diferentes das providas pela Java Sound API. Empregamos CORBA [OMG, 2002, OMG, 1998] para atingir esse objetivo.

Inicialmente, utilizamos as classes de MIDI fornecidas pela API do Java Sound para gerar o som. Devido a isso, a implementação atual da infra-estrutura é baseada no protocolo MIDI. Contudo, tomamos o cuidado de não deixar que a arquitetura fosse muito influenciada por esse protocolo, já que ele é considerado muito limitado para aplicações musicais sofisticadas.

Como alternativa para geração de som, iniciamos experimentos com o ambiente MAX/MSP, que incorpora uma linguagem visual para construção de sistemas interativos e oferece suporte para síntese sonora em tempo real. Esse sistema foi criado originalmente pelo IRCAM (Institut de Recherche et Coordination Acoustique/Musique) em Paris e hoje é desenvolvido e comercializado pela Cycling '74 nos EUA (`http://www.cycling74.com/products/maxmsp.html`). Para a integração com o Andante, utilizamos o suporte não oficial do MAX/MSP para o protocolo Open SoundControl [Wright and Freed, 1997] (veja `http://www.cnmat.berkeley.edu/OpenSoundControl`). Esse protocolo permite a comunicação pela rede entre sistemas multimídia. Planejamos também implementar um Dispositivo de Áudio baseado no sistema CSound, uma poderosa e difundida ferramenta para síntese de som [Boulanger, 2000].

Os agentes móveis foram implementados utilizando a plataforma Aglets. Ela é distribuída na forma de um pacote chamado *Aglets Software Development Kit* (ASDK), que é escrito em Java e foi originalmente desenvolvido pela IBM (`http://www.`

`trl.ibm.com/aglets`). Hoje é um projeto de código aberto (`http://aglets.sourceforge.net`). O ASDK oferece uma API e aplicações para a implementação e gerenciamento em tempo de execução de agentes móveis [Lange and Oshima, 1998].

5.2. Descrição da Arquitetura

No Andante, os agentes executam as suas ações numa rede de computadores heterogênea. Para isso, os computadores dessa rede executam um programa preparado para receber os agentes. Esse programa é um componente da arquitetura, funciona como um palco onde os agentes se encontram e "tocam" música. Chamamos esse componente de *Stage* ou Palco.

O Palco oferece serviços para que os agentes possam realizar suas ações. Para, por exemplo, tocar uma melodia, um agente precisa utilizar o serviço de geração de som do Palco. O Palco, por sua vez, utiliza um outro componente da infra-estrutura para fornecer o serviço de som. Esse componente é o *Audio Device* ou Dispositivo de Áudio.

Acabamos então de definir três elementos fundamentais da arquitetura: os agentes móveis musicais (ou *Mobile Musical Agents*), o Palco (ou *Stage*) e o Dispositivo de Áudio (ou *Audio Device*). A Figura 1 mostra uma visão abstrata da arquitetura.

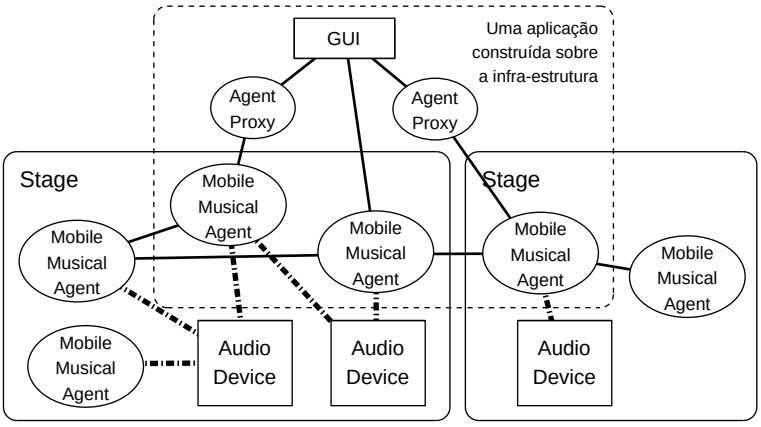

Figura 1: Visão geral da arquitetura

A figura ainda ilustra uma possível aplicação construída sobre a infra-estrutura. A *GUI* não é necessariamente parte da infra-estrutura, mas tem o importante papel de prover uma interface gráfica para interação entre agentes e usuários humanos. A aplicação se utiliza de um quarto componente da arquitetura: o Representante de Agente (ou *Agent Proxy*). Esse componente fornece transparência de localização de um ou mais agentes para a GUI. No caso de uma migração, um agente informa a sua nova localização ao seu representante, que por sua vez é responsável pela comunicação entre a GUI e o agente ou por repassar a nova localização para a GUI. A GUI também pode escolher comunicar-se diretamente com um agente ou ela própria ser representante de um ou mais agentes.

5.3. Implementação

A Figura 2 mostra um diagrama de classes UML da visão abstrata. A classe `MusicalAgent` representa o agente móvel musical, a classe `Stage` é responsável por hospedar agentes numa máquina. Por serem partes que dependem do ASDK, essas duas classes precisam obrigatoriamente ser escritas em Java.

O Palco oferece os seguintes serviços para os agentes.

- `channel`: fornece um canal de comunicação por onde passam mensagens de geração de som, semelhantes às do protocolo MIDI (mas não limitadas a

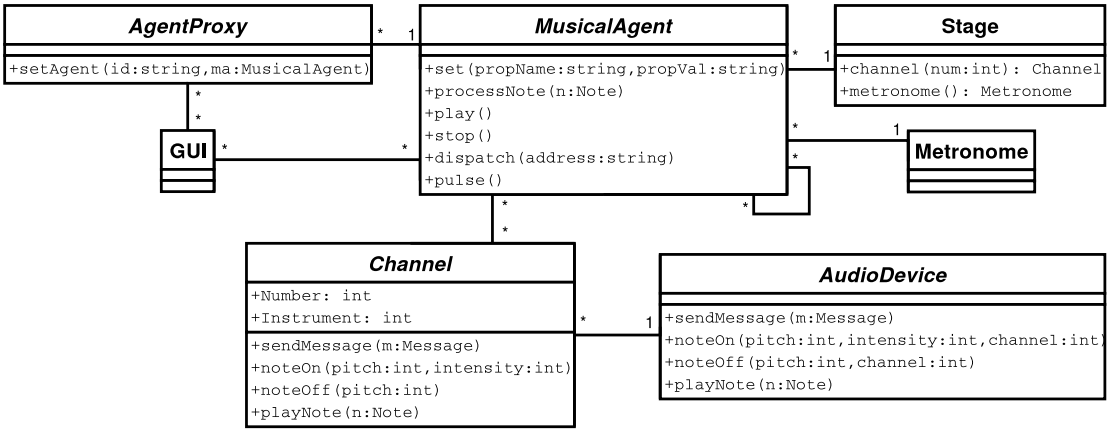

Figura 2: Diagrama de classes da visão geral (Figura 1)

```
// Todo agente móvel musical deve ser subclasse
// da classe abstrata 'MusicalAgent'.
public class RandomMelodyAgent
  extends MusicalAgent {

  boolean play = false;
  short [] cMaj = {60,62,64,65,67,69,71};
  java.util.Random random;
  Channel ch;

  // Método chamado no final do tratamento do
  // evento de chegada deste agente.
  public void init() {
    random = new java.util.Random();
    ch = stage.channel(1);
    play();
  }

  public void play() {
    play = true;
    int pitch, intensity, duration;

    while (play) {
      pitch = cMaj[random.nextInt(cMaj.length)];
      intensity = random.nextInt(128);
      duration = random.nextInt(1000);
      ch.noteOn(pitch, intensity);
      try {
        Thread.sleep(duration);
      }
      catch (InterruptedException ie) {}
      ch.noteOff(pitch);
    }
  }

  public void stop() {
    play = false;
  }

}
```

Figura 3: Exemplo de um agente Andante

este). Em geral, cada palco possui vários canais, cada um permitindo diferentes configurações em seus parâmetros (por exemplo, timbre utilizado para tocar as notas).

- `metronome`: fornece um objeto que funciona como um metrônomo. Esse objeto mede o tempo de acordo com a fórmula de compasso que armazena. Ele recebe pedidos de registros de agentes e envia a mensagem `pulse` em cada tempo do compasso a todos os agentes registrados.

A classe `MusicalAgent` é abstrata, todos os novos agentes devem estender `MusicalAgent` e implementar os métodos que faltam. Em particular, o método `play`, que representa um pedido para que o agente inicie a sua performance, precisa ser implementado. A Figura 3 mostra um exemplo de um agente móvel musical implementado sobre a infra-estrutura do Andante. Esse agente reproduz uma melodia gerada aleatoriamente em tempo real.

Descrevemos a seguir uma situação que ilustra um possível uso do agente `RandomMelodyAgent` na infra-estrutura descrita até agora.

- *Implementação de agentes*: já temos um agente implementado, o `RandomMelodyAgent`. Por simplicidade, usaremos apenas uma instância

desse agente neste exemplo. Vamos chamar essa instância de *rmAgent*.

- *Envio de agentes*: depois de criado, *rmAgent* deve ser enviado a uma instância da classe `Palco`.
- *Procedimentos de chegada*: ao chegar no Palco, o campo herdado `stage` (veja a Figura 3) recebe uma referência para o Palco. Além disso, é iniciado o tratamento do evento de chegada. Esse tratamento é feito pelo código da superclasse `MusicalAgent`. Ao final do tratamento, o método `init` do próprio agente é chamado. No caso de *rmAgent*, esse método obtém um canal (campo `ch` na Figura 3) e chama o método `play`.
- *Performance do agente*: com a chamada ao método `play`, *rmAgent* inicia sua execução. Ele usa as operações de `ch` para tocar notas musicais aleatórias na escala diatônica de dó.
- *Controle do agente*: é possível implementar uma interface gráfica que envia comandos para *rmAgent*. Nesse caso, isso poderia ser útil para chamar o método `stop` desse agente ou transferi-lo para outro palco.

Camada CORBA

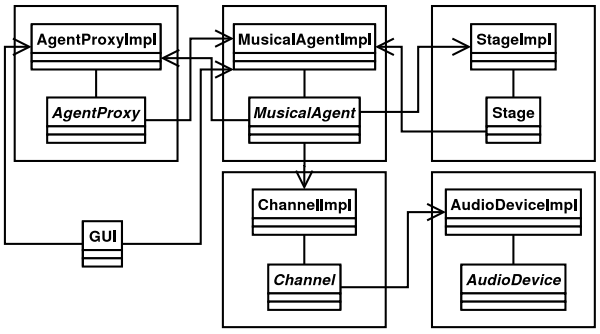

Figura 4: Diagrama de classes com a camada CORBA

Como já mencionamos, queremos permitir que certos elementos, mais precisamente `AudioDevice` e `GUI`, sejam implementados em outras linguagens e, eventualmente, utilizem sistemas legados. Para tornar isso possível, existe uma camada CORBA para a comunicação entre os elementos. Essa camada, que foi omitida da descrição até agora, é ilustrada na Figura 4 (por simplicidade, os métodos e metrônomo não estão na figura). As classes com o sufixo *Impl* no nome representam serventes CORBA.

6. Aplicações de Exemplo

Para demonstrar a viabilidade da arquitetura descrita na Seção 5, construímos duas aplicações simples utilizando a infra-estrutura do Andante.

6.1. NoiseWeaver

NoiseWeaver é uma a aplicação para geração de música estocástica distribuída. Somente um tipo de agente é utilizado: o *NoiseAgent*. Ele utiliza ruídos $\frac{1}{f^{\beta}}$ para gerar uma melodia simples [Roads, 1996] que é reproduzida pelo agente em tempo real. Nela, seqüências de números que simulam tipos de ruídos selecionados pelo usuário determinam os parâmetros de altura, intensidade e duração de cada nota. Por exemplo, um agente do tipo NoiseAgent pode gerar uma melodia cujas alturas das notas são determinadas por uma seqüência de números que simula o ruído rosa. O mesmo agente pode ter as durações das notas determinadas por um ruído browniano e as intensidades determinadas por um ruído branco. Simplificando, *ruído* no contexto desta aplicação significa uma seqüência

de números inteiros gerada através de um algoritmo estocástico e que é mapeada para parâmetros musicais de uma melodia.

Os agentes usam o serviço de metrônomo do Palco para sincronizar as melodias. Todos os agentes se registram no serviço e o metrônomo passa então a enviar a mensagem `pulse` a todos eles em intervalos de tempo regulares. Ao receber a mensagem `pulse`, um NoiseAgent gera uma nota da melodia (ou não faz nada, se a última nota ainda estiver soando). Cada agente possui diversas propriedades que determinam o modo como a sua melodia deve ser gerada e reproduzida. Essas propriedades serão descritas mais adiante.

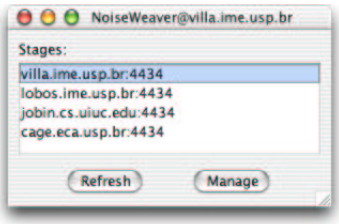

Figura 5: Escolha de Palcos

A aplicação NoiseWeaver fornece uma interface gráfica para controlar vários NoiseAgents hospedados em vários Palcos. Tanto os Palcos como os NoiseAgents são localizados através do Serviço de Nomes CORBA [OMG, 1998]. A Figura 5 mostra a janela com os Palcos encontrados. Para controlar os NoiseAgents de um determinado Palco, o usuário deve selecionar o Palco e pressionar o botão "*Manage*". Com isso, abre-se uma janela semelhante à exibida na Figura 6. Essa janela permite que um usuário altere as propriedades do metrônomo do Palco e dos agentes, mesmo enquanto as melodias estão sendo produzidas.

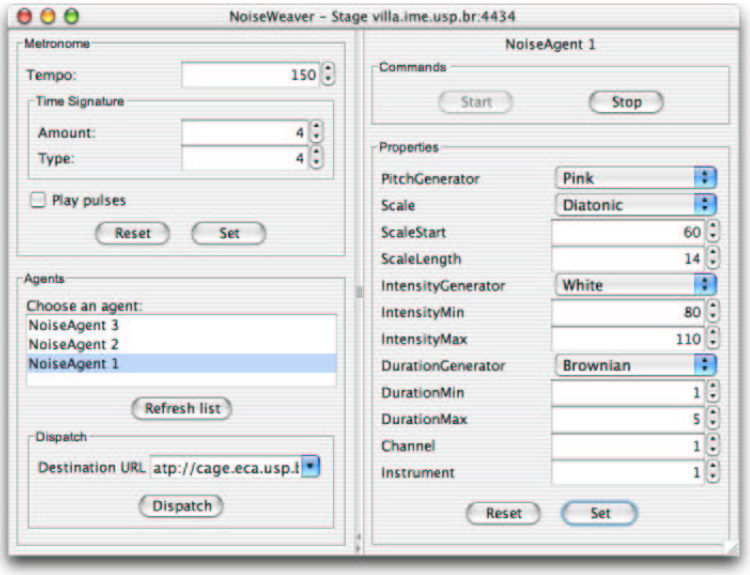

Figura 6: Controle de um Palco

O painel *Metronome* permite definir os parâmetros do metrônomo. *Tempo* é o andamento da música em pulsos por minuto; *Amount* e *Type* são a fórmula do compasso; e *Play pulses* determina se o metrônomo emite ou não um som a cada tempo do compasso. O painel *Agents* lista os vários agentes hospedados no palco. O painel *Dispatch* se refere ao agente selecionado na lista e permite enviar uma ordem de migração para outro palco.

Quando um agente é selecionado, o painel da metade direita da janela é ativado. Esse painel permite controlar as propriedades do agente. No painel *Commands*, temos:

- *Start*: faz o agente começar (ou continuar) a tocar;
- *Stop*: faz o agente parar de tocar.

No painel *Properties*, temos:

- *Altura*: *PitchGenerator* define o tipo de ruído (ou seja, o algoritmo) que será utilizado para gerar as alturas (freqüências) das notas. Os números gerados pela simulação do ruído são mapeados para notas da escala definida por *Scale*. *ScaleStart* e *ScaleLength* determinam quais notas dessa escala serão utilizadas. *Sca-*

leStart é a primeira nota da escala, sendo que 60 é o dó central. *ScaleLength* é o número de notas da escala que serão utilizadas, contando a partir de *ScaleStart*.

- *Intensidade*: *IntensityGenerator* define o tipo de ruído que gerará a intensidade das notas. A intensidade é um inteiro entre 0 e 127. *IntensityMin* e *IntensityMax* determinam o intervalo de valores possíveis.
- *Duração*: *DurationGenerator*, *DurationMin* e *DurationMax* são as propriedades de duração das notas, funcionam de forma análoga às propriedades de intensidade.
- *Channel*: o canal que o agente usa para enviar as notas.
- *Instrument*: o instrumento usado para tocar a melodia.

Os possíveis valores para os geradores de números baseados em ruídos são: *Constant*, *White*, *Pink* e *Brownian*. As possíveis escalas são: *Diatonic*, *WholeTone*, *Chromatic*, *HarmonicMinor*, *HarmonicNatural*, *Pentatonic*, *Blues*, *PentaBlues* e *Hirajoshi*.

A aplicação NoiseWeaver permite então gerar diversas melodias estocásticas simultâneas através dos NoiseAgents e controlá-las usando a interface gráfica.

6.2. Maestro

A aplicação *Maestro* é uma extensão da aplicação *NoiseWeaver*. Em vez de serem controlados por uma interface gráfica, uma coleção distribuída de NoiseAgents é coordenada por um Maestro, que por sua vez é controlado por um script. O elemento principal do script é a partitura, onde são determinadas mudanças nas propriedades dos agentes ao longo do tempo. Essas mudanças são feitas através da interface `MusicalAgent`, isto é, cada mudança é um envio de mensagem para um agente. Vejamos então uma descrição informal do script de entrada para o Maestro.

Formato do script:
<*Declarações*>
--
<*Inicializações*>
--
<*Partitura*>
--

<*Declarações*> := <*Classe*> <*Id*>; |
<*Classe*> <*Id*>; <*Declarações*>

<*Inicializações*> := <*Envio de mensagem*>; |
<*Envio de mensagem*>; <*Inicializações*>

<*Partitura*> := <*Tempo*> <*Envio de mensagem*>; |
<*Tempo*> <*Envio de mensagem*>; <*Partitura*>

<*Envio de mensagem*> := <*Id*> <*Mensagem*> <*Lista de parâmetros*>

<*Lista de parâmetros*> := <*Parâmetro*> |
<*Parâmetro*> <*Lista de parâmetros*>

<*Parâmetro*> é um nome qualquer.
<*Mensagem*> é o nome de um método da interface `MusicalAgent`.
<*Tempo*> é um instante de tempo (um número inteiro).
<*Classe*> é o nome de uma classe Java que define um agente.
<*Id*> é um nome qualquer.

E, a seguir, um exemplo.

```
NoiseAgent a1;
NoiseAgent a2;
--
a1 set DurationMin 1;
a1 set DurationMax 1;
a1 set Scale Diatonic;
a1 set ScaleStart 36;
a1 set ScaleLength 12;
a1 set Channel 1;
a1 set Instrument 1;
```

```
a2 set PitchGenerator Pink;
a2 set Scale Diatonic;
a2 set ScaleStart 72;
a2 set ScaleLength 12;
a2 set Channel 2;
a2 set Instrument 36;
a1 dispatch villa:4434;
a2 dispatch lobos:4434;
--
1  a1 play;
```

```
5  a2 play;
13 a1 set IntensityMin 80;
13 a1 set Scale Chromatic;
17 a2 set DurationMax 2;
17 a2 set ScaleStart 60;
25 a1 dispatch lobos:4434;
25 a2 dispatch villa:4434;
37 a1 stop;
37 a2 stop;
--
```

No trecho de declarações são criados dois agentes do tipo `NoiseAgent`. Em seguida, são inicializadas algumas propriedades de cada um dos agentes (veja Seção 6.1) e os agentes

são enviados para diferentes palcos. Na seção com rótulos de tempo, são definidos os momentos de início e término da performance de cada agente e, nesse meio tempo, as mudanças nas propriedades e a migração.

Apesar de inicialmente ser uma extensão do NoiseWeaver, o Maestro permite controlar qualquer tipo de agente, não apenas NoiseAgents. Agentes implementados no futuro também poderão ser usados sem alterações no código do Maestro. Essa flexibilidade foi obtida através da API de reflexão de Java, as classes de agentes podem ser definidas em tempo de execução. O mesmo é feito com as mensagens passadas para os agentes. Qualquer mensagem nova incluída na interface `MusicalAgent` que receba parâmetros do tipo `String` pode ser chamada sem alterações no Maestro.

Nos próximos meses, faremos experimentos com músicos especialistas em composição que utilizarão esta infra-estrutura para escrever partituras de novas obras musicais baseadas no *Maestro* e em dezenas de NoiseAgents percorrendo várias máquinas em diferentes localizações.

7. Conclusões e Trabalhos Futuros

O projeto Andante pretende ser mais que um sistema computacional. Esperamos criar uma comunidade onde artistas e cientistas colaboram contribuindo com novas idéias musicais, com novos agentes móveis e auxiliando no melhoramento da infra-estrutura. Para tanto, mantemos um sítio na Internet: `http://gsd.ime.usp.br/andante`.

Como já temos um protótipo inicial implementado, desejamos agora mover o foco principal do projeto na direção da criação musical. Para isso, os trabalhos seguintes serão acompanhados de uma forte interação com compositores.

Pretendemos continuar o desenvolvimento da infra-estrutura e das aplicações já construídas, implementando novas funcionalidades e refinando o projeto e a implementação da arquitetura. Além disso, idealizaremos e construiremos novas aplicações que explorem a interação homem-agente e agente-agente e a mobilidade dos agentes. A mobilidade pode existir entre computadores no mesmo ambiente e entre computadores presentes em espaços físicos geograficamente distantes. Planejamos criar peças musicais que explorem estas possibilidades.

Para garantir a performance das peças musicais em tempo real, é necessário que a infra-estrutura do Andante forneça garantias às aplicações de que poderão executar dentro dos devidos prazos. A implementação desse suporte será necessária para atrair o interesse de mais usuários. A garantia de qualidade de serviço (*Quality of Service – QoS*) está relacionada a essa questão. A idéia é permitir que as aplicações definam seus requisitos de recursos de sistema (como uso do processador, memória, largura de banda da rede, etc.) para poderem executar com uma qualidade satisfatória. Esperamos que o suporte a tempo real em Java (RTSJ) cuja especificação (veja `http://www.rtj.org`) foi concluída recentemente e cujas primeiras implementações (`http://www.timesys.com/prodserv/java`) estão surgindo, venha a ser o veículo ideal para aplicações musicais com seus fortes requisitos de qualidade de serviço.

Finalmente, seria interessante possibilitar a programação de agentes por usuários não programadores. A construção de um ambiente de programação baseado em modelos visuais pode ser um interessante projeto a ser integrado ao Andante no futuro.

Referências

Boulanger, R., editor (2000). *The CSound Book*. The MIT Press.

Casey, M. and Smaragdis, P. (1996). Netsound. In *Proceedings of the 1996 International Computer Music Conference*, Hong Kong.

Gray, R. S., Kotz, D., Cybenko, G., and Rus, D. (1998). D'Agents: Security in a Multiple-Language, Mobile-Agent System. In Vigna, G., editor, *Mobile Agents and Security*, volume LNCS 1419, pages 154–187. Springer-Verlag.

Hiller, L. (1970). Music composed with computers: A historical survey. In Lincoln, H., editor, *The Computer and Music*, pages 42–96. Ithaca, NY: Cornell University Press.

Iazzetta, F. (2003). A performance interativa em Pele. In *Proceedings of the 9th Brazilian Symposium on Computer Music*, pages 249–255, Campinas, Brazil.

Johansen, D. et al. (1994). Operating system support for mobile agents. Technical Report TR94-1468, Department of Computer Science, Cornell University, USA.

Johansen, D. et al. (1995). An introduction to the TACOMA distributed system version 1.0. Technical Report 95-23, University of Tromsø, Norway.

Johansen, D. et al. (2002). A TACOMA retrospective. *Software - Practice and Experience*, 32(6):605–619.

Kon, F. and Iazzetta, F. (1998). Internet music: Dream or (virtual) reality? In *Proceedings of the 5th Brazilian Symposium on Computer Music*, Belo Horizonte, Brazil.

Kotz, D. and Gray, R. S. (1999). Mobile Agents and the Future of the Internet. *ACM Operating Systems Review*, 33(3):7–13.

Lange, D. and Oshima, M. (1998). *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley.

Lange, D. B. and Oshima, M. (1999). Seven Good Reasons for Mobile Agents. *Communications of the ACM*, 42(3):88–89.

Loy, D. G. (1989). Composing with computers — a survey of some compositional formalisms and music programming languages. In Mathews, M. V. and Pierce, J. R., editors, *Current Directions in Computer Music Research*, pages 292–396. The MIT Press.

Miranda, E. R. (2001). *Composing Music with Computers*. Oxford (UK): Focal Press.

OMG (1998). *CORBAservices: Common Object Services Specification*. Object Management Group, Framingham, MA. OMG Document 98-12-09.

OMG (2002). *CORBA v3.0 Specification*. Object Management Group, Needham, MA. OMG Document 02-06-33.

Roads, C. (1996). *The Computer Music Tutorial*. The MIT Press.

Rowe, R. (1993). *Interactive Music Systems*. The MIT Press.

Ueda, L. K. and Kon, F. (2003). Andante: A mobile musical agents infrastructure. In *Proceedings of the 9th Brazilian Symposium on Computer Music*, Campinas, Brazil.

Wright, M. and Freed, A. (1997). Open SoundControl: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 International Computer Music Conference*, Thessaloniki, Greece.