AMD - Ambiente Musical Distribuído – Performance Musical Interativa na Internet

Anselmo Guerra de Almeida e Rodrigo Peres Franco Furtado

Laboratórios de Pesquisas Sonoras – LPqS
Universidade Federal de Goiás - CAMPUS II
Caixa Postal 131 - CEP 74001-970 Goiânia / GO

{anselmo@musica.ufg.br, furtado@ih.com.br}

Abstract: Distributed Musical Environment was conceived as an environment that allows musicians to work with synthesized sound elements by way of network computers. They interact each other using graphical interface and a system that provides a communication between them. Regardless of geographic distance that separates them, the participants in this type of performance interact with each other distributed musical process. The esthetic aspect of this type of musical performance is still an open question. Perhaps it is only through the continuing exploration of this creative medium that we are able to measure the musical merit of this environment.

Introdução

O AMD - Ambiente Musical Distribuído - permite que, através de uma rede de computadores, músicos possam interagir com elementos sonoros, independentemente da localização geográfica que os separa. Para esse tipo de performance a idéia é que tenhamos vários computadores clientes em comunicação com o servidor via rede local ou clientes espalhados em outras redes ligados ao servidor via Internet. O servidor pode estar em um ambiente onde o público poderá assistir a esta audição. Ele será o responsável por produzir os requerimentos sonoros que chegarão dos clientes. Desta forma, o servidor tem a ele interligado os aparatos eletrônicos para reproduzir o som. No lado do cliente o concerto também será reproduzido, necessitando-se, ao menos, de uma placa e caixas de som.

AMD se enquadra em uma categoria de programas de performance interativa e, especificamente, na performance via rede (internet ou intranet). Nesse contexto, encontramos projetos similares, como o MidiChat e MidiShare (JAVA MUSIC PROJECT 2000), só para citar alguns. Como característica em comum, os programas desse gênero evitam o uso de arquivos áudio, utilizando o formato MIDI, que contém as mensagens que acionam eventos musicais (notas, intensidades, banco de timbres) no sintetizador interfaceado ao computador cliente. Essa escolha permite que se minimize os problemas de latência inerente à comunicação via rede.

Em sua particularidade, AMD trabalha simultaneamente com diversos formatos de arquivos áudio (aiff, au, wav) e MIDI. Esses arquivos são gerados previamente pelos participantes, e introduzidos na performance. Esse ambiente é voltado para a performance de música eletroacústica, onde a latência está admitida, usada e incorporada ao tempo musical.

1. AMD - O Sistema Cliente-Servidor

Um sistema cliente-servidor (DEERING & HINDEN 1997) nos dá a possibilidade de construirmos programas que tenham uma relação de centralização das ações que o sistema executa (RAGAVAN & TRIPATI 1998) (STEINEMETZ & NAHRSTED1995). No caso do nosso sistema esta centralização se dá com as requisições que os clientes fazem ao servidor para a performance/composição.

1.1 O Servidor

O servidor é um programa Java que recebe requisições dos clientes e as remetem a todos os outros. O servidor, como é característico de programas desta natureza, fica na "escuta" por ações dos clientes. Esta espera é possível porque os clientes e o servidor usam uma mesma porta lógica através de um mecanismo de comunicação que chamamos de *socket*. Java possui um conjunto de classes que implementam comunicações entre clientes e servidor utilizando este tipo de mecanismo. A classe Socket permite que clientes se comuniquem com um servidor com determinado nome e em uma determinada porta de acesso do sistema operacional em questão (Windows, Linux, Unix, MacOS, etc.). Através desta comunicação, por esta porta, o cliente pode se comunicar *via socket* com o servidor e vice-versa.

O servidor além de se comunicar com os clientes precisa atender às requisições dos mesmos. Para cada cliente o servidor cria uma instância (*thread*), a qual irá atender somente aquele cliente. Uma *thread* é um processo que é disparado para atender a uma exigência exclusiva. Um servidor só pode atender a múltiplos clientes se puder disparar *thread*s para cada um deles. Caso contrário ele só poderia atender a um cliente. Assim para cada novo cliente uma nova *thread* é criada e trata exclusivamente do mesmo. Iniciada a comunicação via *socket* e as *threads* correspondentes, a comunicação entre clientes e o servidor está garantida. Abaixo mostramos parte do algoritmo que implementa a parte do servidor:

```
/** Classe Music Server */

public class MusicServer {

  public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = null;
    boolean listening = true;
    ClientList client_list = null;

    try {
      serverSocket = new ServerSocket(5100,30);
      System.out.println("Porta 5100");
    } catch (IOException e) {
      System.err.println("Could not listen on port: 5100.");
      System.exit(-1);
```

1.2 A Tarefa do Servidor

O servidor recebe requisições dos clientes através dos *socket*s. O servidor responde aos clientes "escutando" numa determinada porta de acesso do sistema operacional, como vimos anteriormente. Na realidade os clientes existem independentemente do servidor, porém seu funcionamento só é possível, através do mesmo. Os clientes são *applet*s que são carregados através de um *browser web* ou um

visualizador de *applets*. Esta chamada ao *applet* (cliente) faz com que o mesmo inicie sua comunicação com o servidor através de uma requisição de um *socket*. Depois disto o servidor espera por requisições dos clientes. Estas requisições são eventos de *mouse* e teclado, que acontecem sobre a interface do cliente. Estes eventos geram ações que são remetidas ao servidor e este as executa. Como sabemos, o servidor tem o papel de centralizar as ações exercidas pelos clientes.

Para que os clientes possam funcionar corretamente o servidor deve estar funcionando desde o início da performance. O servidor é o primeiro processo computacional que deve executar antes da performance ter início.

Quando o servidor é iniciado fica na espera pelos clientes. Cada cliente faz diferentes requisições ao servidor e este sabe como processá-las. O servidor recebe, como comandos dos clientes, ordens para enviar determinados arquivos sonoros para os mesmos. Estes arquivos são executados (e ouvidos) em cada um dos clientes. O servidor possui uma lista dos clientes a ele conectados e envia cada requisição sonora a todos. Este tipo de difusão de informação, que o servidor executa é o que chamamos de *broadcast*. Um *broadcast* significa que todas as ações que um cliente solicita a um servidor serão enviadas a todos os demais. Desta forma todas as ações realizadas pelos clientes serão conhecidas pelos demais, e todos as reproduções sonoras serão realizadas em cada cliente.

O servidor então tem o papel de controlar as comunicações entre os clientes. Cada *performer* participará independentemente da obra que está sendo executada, e terá acesso a participação dos outros. O servidor pode também reproduzir todas as ações que ocorrem no Ambiente. Como todas as requisições passam primeiramente pelo servidor, ele possui os meios para a reprodução exata da performance distribuída. No caso de uma performance onde todos os participantes estão fisicamente distantes, utilizando a Internet como meio de comunicação, o servidor pode reproduzir a performance para uma audiência fixa no mesmo espaço físico. Desta forma o servidor tem o papel principal de promover a comunicabilidade entre os participantes da performance e se necessário reproduzir o concerto para uma audiência fixa, num espaço físico determinado.

1.3 O Cliente

O cliente é um *applet* que executa na máquina de quem o requisitou. Ele contará com uma interface que permitirá ao músico interagir com o servidor durante a performance distribuída. No caso do Ambiente os *applets* não se comunicam diretamente um com os outros. As requisições dos clientes passam primeiramente pelo servidor e só depois são distribuídas aos demais. Cada cliente possui uma interface gráfica semelhante, e todos tem a mesma possibilidade de ações. Os *applets* são carregados em páginas html nas máquinas dos *performer*s. Estes *applets* fazem uma requisição ao servidor, por um *socket* exclusivo. Este *socket* se torna a via de comunicação entre este cliente e o servidor, e indiretamente entre os clientes.

Um cliente não precisa ser obrigatoriamente um *applet*. No caso de aplicações em rede o *applet* se torna interessante pois pode ser carregado em qualquer máquina, com um *browser* html, conectada ao servidor via rede. Portanto nossa escolha em construir o cliente como um *applet*, é justamente por termos esta necessidade.

O cliente, assim como o servidor, comunica-se através de um *socket*. Este *socket* é implementado pela classe Java Socket. O cliente sabe através de parâmetros o nome do *host* (máquina onde o servidor está executando) e a porta na qual ele "escuta". Quando o servidor é iniciado, os clientes podem, através do *socket*, fazer requisições ao servidor, enviar e receber dados do mesmo.

O *applet* deve também disparar uma *thread* específica para ficar à espera de dados que chegam do servidor. Diferentemente do servidor, cada *applet* cria uma, e apenas uma, *thread*. Esta *thread* é exclusiva daquele cliente e só é conhecida e manipulada pelo mesmo. Então podemos concluir o papel das *threads* da seguinte forma: as *threads* disparadas pelo servidor servem para atender a múltiplos clientes, de outra forma o servidor só poderia atender a um cliente por vez; a *thread* disparada pelo *applet* (cliente) serve para que o mesmo possa capturar os dados que chegam do servidor. A seguir é mostrada parte do código do cliente:

```
//método start() do applet
    public void start () {
    Thread outputT;
    String serverHost = "200.137.201.21";
    String argServerPort = getParameter ("serverport");
    //trasnformando o valor da string em inteiro
    Integer serverPort = new Integer(argServerPort);



    try {
        clientSocket = new Socket(serverHost, serverPort.intValue());
        out = new PrintWriter(clientSocket.getOutputStream(), true);
    in=newBufferedReader(newInputStreamReader
                        (clientSocket.getInputStream()));
    } catch (UnknownHostException e) {
        System.err.println("Don't know about host: furtado.");
        System.exit(1);
    } catch (IOException e) {
        System.err.println("Couldn't get I/O for the connection to: furtado.");
        System.exit(1);
    }

    BufferedReader stdIn = new BufferedReader
                        (new InputStreamReader(System.in));

    //disparo uma thread para cuidar da comuncacao deste cliente com o servidor
```

1.4 A Tarefa do Cliente

O cliente é uma interface para que o usuário possa participar da performance distribuída. Esta interface permite ao participante escolher um determinado elemento sonoro a ser executado. Depois que a escolha é feita o *applet* envia para o servidor a referência ao arquivo sonoro e este é distribuído para todos os clientes através de um *broadcast*, como vimos. O cliente então envia e recebe dados diretamente do servidor. A cada novo dado recebido o cliente executa um arquivo sonoro. Estes dados são executados na ordem que chegam do servidor.

Ao cliente então cabem três tarefas principais: prover uma interface com os elementos sonoros que serão utilizados durante a performance/composição; enviar estes dados selecionados pelo usuário; receber os dados enviados pelo servidor. Lembramos, finalmente, que todos os dados que chegam ao cliente foram enviados por outro cliente (ou por ele mesmo) ao servidor, e este tratou de redirecioná-los a todos os participantes (aos outros *applet*s da performance).

2. Funcionamento do AMD

Vamos agora resumidamente detalhar como o AMD funciona como um todo. Como já mostramos os detalhes do cliente e do servidor vamos aqui focar o aspecto global de funcionamento do AMD. O AMD inicia-se com a execução do servidor. A primeira tarefa é colocar o servidor em funcionamento. Isto é feito através de uma simples chamada na linha de comando do sistema operacional: ***java MusicServer***

Após esta chamada o servidor está pronto para receber clientes e tratá-los. Os clientes são executados através de uma página html. Esta página é carregada em um browser com o endereço IP (Internet Protocol) da máquina onde o cliente está executando: ***http://123.123.123.123/AMD.html***

O número IP no exemplo acima é fictício. Para cada rede e máquina o IP deve ser ajustado. Após a página html ser carregada no browser os *performer*s podem iniciar a execução dos seus trabalhos. Lembrando que cada *performer* pode iniciar e finalizar sua participação quando desejar. A seguir temos uma ilustração da interface que será utilizada pelo *performer*:

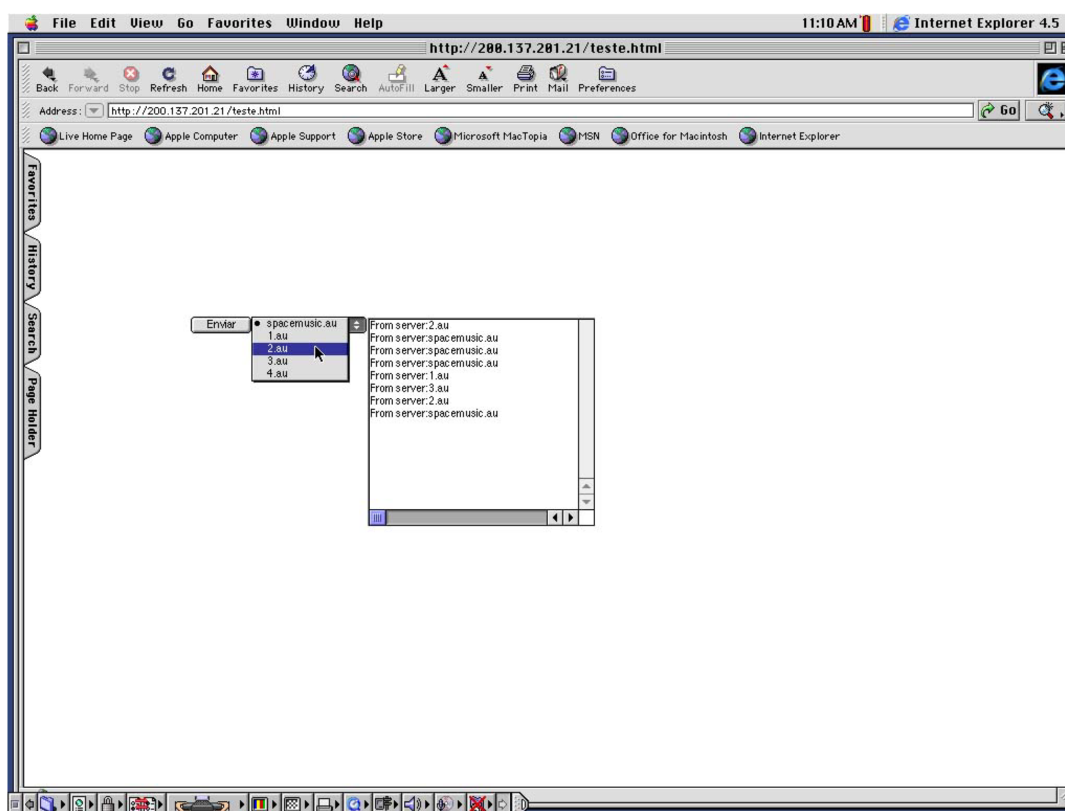

Figura 1- interface do AMD

Desta forma a cada intervenção do usuário, ações são requisitadas ao servidor, que trata de distribuí-las aos demais participantes. Assim a obra vai se desenvolvendo até que o último participante deixe o AMD. Quando isso ocorre, a execução do servidor pode ser encerrada e a performance/composição termina.

3. Performance Musical via AMD

Do ponto de vista estético o resultado sonoro gerado por uma performance no AMD tem um caráter muito particular. Cada participante possui um conjunto de

eventos sonoros, na forma de arquivos aiff, au, wav ou MIDI, que podem ser utilizados durante a performance. O resultado musical se constitui de um complexo sonoro gerado com a intervenção de cada participante. A estrutura deste complexo pode se desenrolar sem tempo previsto e sem regras pré-determinadas ou pode também ser programada previamente pelos participantes.

Além desta flexibilidade do AMD em relação aos elementos sonoros e ao som gerado, vale a pena ressaltar um fator importante. O resultado sonoro possui um caráter de improviso, pois é resultado da criatividade e vontade dos participantes. Além do improviso, está presente também a imprevisibilidade na própria estrutura física e lógica que o AMD utiliza. Sabemos que em transmissões via rede não podemos prever com certeza o tempo de transmissão e o caminho que os dados vão seguir. Apesar de termos bom indicativos de performance computacional em redes, sabemos de que fatores aleatórios podem ocorrer (atrasos de transmissão, queda em redes específicas, perda de dados, etc). Estes fatores característicos da estrutura das redes de computadores podem produzir efeitos inusitados e interessantes.

Sendo assim, o resultado deve ser visto como um tipo especial de obra musical onde performance, composição, apreciação estão intimamente ligados. Além disso, as limitações do meio são incorporadas e usadas como parte do conjunto de particularidades do sistema.

4. Conclusões

Ao longo de todo o trabalho, a preocupação principal foi a viabilização da tecnologia de redes para o uso musical. As possibilidades de uma nova concepção de performance/composição/apreciação musical foram os alicerces deste projeto. É a música que tenta explorar estas novas potencialidades da tecnologia. Como não poderia ser diferente, o AMD foi construído para músicos. Sendo assim a nossa visão para o futuro do AMD será baseada nas experiências e no uso que os músicos farão deste ambiente. É através do *feedback* dos mesmos, que o AMD poderá se desenvolver ainda mais.

O AMD não constitui um sistema computacional simplesmente. Antes de tudo ele é um laboratório, para permitir novas experiências e concepções na música contemporânea. A sua construção foi baseada muito mais nas possibilidades que ele poderia trazer à música, do que propriamente se ele poderia ser implementado com as tecnologias disponíveis. Assim todas os seu possíveis melhoramentos e mudanças futuras serão baseados nas opiniões dos usuários que dele se utilizarem.

Referências Bibliográficas

BOOCH, G (1994) Object Oriented Design with Applications. Benjamin/Cummings.

DEERING, S.; HINDEN, R. (1997) Internet Protocol Version 6 (IPv6) Specification. Londres.

DEITEL, H.M.; DEITEL, P.J. (1998) Java: how to program. Prentice Hall:New Jersey.

GOGINS, M. (1999) AXCsound/JCsound 3.0. New York. Disponível em: <http://www.pipeline.com/~gogins>.

MEYER, B.(1988) Object-Oriented Software Construction. New York: Prentice Hall.

RAGAVAN, S.V; TRIPATI, S.K. (1998) Netwoked Multimedia Systems Concepts, Architeture and Design – New Jersey; Prantice Hall.

STEINEMETZ, R.; NAHRSTED, K. (1995) Multimedia: Computing Communications and Applications. New Jersey: Prentice Hall.

SUN (2001). The Java Tutorial. Silicon Valey. Disponível em: <http://www.javasoft.com>.

VERCOE, B. (1997) Csound Manual. Massachusettes: Media Lab, MIT.