Uma Plataforma para Programação Multimídia Baseada no Modelo ISO/PREMO e sua Implementação

Adilson Barboza Lopes
DIMAp-UFRN
Natal-RN/Brasil
e-mail: adilson@dimap.ufrn.br

Lucimara Desiderá
DCA-FEEC-UNICAMP
Campinas-SP/Brasil
email: lucimara@dca.fee.unicamp.br

Maurício Ferreira Magalhães
DCA-FEEC-UNICAMP
Campinas-SP/ Brasil
e-mail:mauricio@dca.fee.unicamp.br

Neil Paiva Tizzo
DCA-FEEC-UNICAMP
Campinas-SP/Brasil
e-mail: tizzo@dca.fee.unicamp.br

Resumo

Este artigo apresenta uma plataforma para programação multimídia distribuída baseada na arquitetura MSS (Multimedia System Services) do Modelo ISO/PREMO. Essa plataforma é constituída em sua base pelos objetos definidos no componente MSS e por um modelo de programação que contempla o ciclo de vida completo dos objetos. A plataforma tem como objetivo suportar o desenvolvimento e a execução de aplicações multimídia em ambientes operacionais heterogêneos. O artigo descreve também uma implementação que que foi realizada utilizando uma plataforma CORBA (Common Object Request Broker Architecture) como infraestrutura de suporte à distribuição. No artigo são tratados os aspectos da implementação da plataforma relacionados com a conexão de dispositivos multimídia e o esquema de sincronização de fluxos de dados em ambientes distribuídos.

Palavras-chave: Multimídia; Sistemas distribuídos; Sincronização; Programação Orientada a Objetos

1. Introdução

A constante evolução da tecnologia na área de microeletrônica com a conseqüente redução no custo de componentes computacionais vem popularizando cada vez mais o uso do computador. Concomitantemente os atuais sistemas de comunicação de dados têm atingido níveis altos de eficiência e velocidade de transmissão, estimulando dessa maneira a construção de sistemas distribuídos. A expectativa é que brevemente usuários geograficamente dispersos e oriundos de plataformas de "hardware" e "software" heterogêneas possam se comunicar, via troca de informações multimídia, no contexto dos mais variados tipos de aplicação.

A pesquisa para computação multimídia distribuída envolve questões em diversas áreas de conhecimento. [1] avalia pontos concernentes à tecnologia de comunicação de sistemas distribuídos em aplicações multimídia, especificamente em redes multi-serviços, interfaces de redes, suporte de comunicação e de sistema operacional, plataformas de sistemas distribuídos e gerência de Qualidade de Serviço. [2] destaca também algumas questões relacionadas com redes de comunicação, sistemas operacionais e sistemas de bancos de dados.

Os modelos padrões e de referência para interconexão de computadores em redes têm atingido níveis de aderência e conformidade, de maneira que atualmente já é possível interconectar subredes das mais variadas tecnologias. Nesse contexto o grande impacto está centrado na tecnologia ATM[15].

No que se relaciona com a computação distribuída a maioria das soluções tem adotado a tecnologia

de orientação a objetos para definição de *frameworks*, ambientes e plataformas[2]. No contexto atual pode-se destacar em especial a plataforma CORBA (Common Object Request Broker Architecture) da OMG (Object Management Group) [3],[4] que é a mais difundida comercialmente, e ODP (Open Distributed Processing) da ISO[5], [6], [7], [8].

Por outro lado a conjunção de orientação a objetos com a multimídia tem motivado a definição de diversos *frameworks* de objetos destinados à representação de características de sistemas multimídia. Dentre eles estão o *Framework* de Simon Gibbs[2] e o MSS(Multimedia System Services) da IMA (Interactive Multimedia Association)[9], [12]. Nesse contexto foi desenvolvido um Projeto Tematico financiado pela FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo)[16] cujo objetivo consiste na construção de uma plataforma distribuída com ênfase em aplicações multimídia. Este trabalho foi realizado no escopo desse projeto.

A plataforma proposta é constituída basicamente pelos componentes fundamentais[11] e MSS do Modelo de Referência PREMO[12], por um ciclo de vida ampliado que foi definino para os objetos da aplicação e por um conjunto de componentes de suporte ao ciclo de vida dos objetos. Entretanto o artigo limita o escopo da apresentação à estrutura de conexão de dispositivos multimídia e ao esquema de sincronização de fluxos de dados. Os demais componentes são abordados superficialmente, objetivando apenas dar uma visão geral da plataforma e do modelo de implementação proposto.

A apresentação do artigo está organizada em 5 seções. A seção 2 introduz o framework MSS e a seção 3 apresenta uma descrição geral da plataforma proposta; a seção 4 discute a implementação e descreve os experimentos realizados. Finalizando, a seção 5 conclui fazendo algumas considerações finais sobre o trabalho e apresentando algumas sugestões como trabalhos futuros.

2. Serviços de Sistemas Multimídia

O *framework* denominado Serviços de Sistemas Multimídia (*Multimedia System Services - MSS*) foi desenvolvido pela IMA *(Interactive Multimedia Association)* na forma de uma Prática Recomendada (*Recommended Practice*-RP) e está sendo incorporado pela proposta de padronização PREMO (Presentation Environment for Multimedia Objects) da ISO/IEC elevando o nível de generalidade do mesmo. A representação das interfaces dos serviços foram definidas primeiramente, na proposta IMA, em IDL, e em PREMO, na linguagem formal Z e Object-Z [10].

O componente MSS provê um conjunto padrão de serviços que pode ser utilizado por desenvolvedores de aplicações multimídia em uma variedade de ambientes computacionais. O MSS constitui um *framework* de *middleware* (camada de componentes de *software* entre o sistema operacional e as aplicações) e não uma plataforma, uma vez que não é direcionada à segurança, *tollkits*, *scripting*, interface com usuário e compartilhamento de aplicações [12], [13]. Como *middleware*, o MSS controla os recursos de baixo nível do sistema no que se relaciona com o suporte ao processamento multimídia.

As referências [10],[11],[12] descrevem o Modelo de Referência PREMO, cujos conceitos são amplamente utilizados na apresentação desse trabalho.

3. Plataforma Proposta

Com relação aos serviços básicos a plataforma proposta adota o *framework* MSS. Embora o MSS facilite a estruturação de aplicações, ele requer do usuário programador o uso de abstrações de nível relativamente baixo para a especificação dos seus requerimentos. Desde que os usuários expressam normalmente os seus requisitos através de abstrações de alto nível, como, por exemplo, "vídeo colorido de boa qualidade e áudio de qualidade CD", estes precisam ser mapeados em informações técnicas significativas para o sistema, tipo "video com taxa de apresentação de 25 quadros por segundo e áudio com amostras de 16 bits numa taxa de amostragem de 8KHz". Dificuldades maiores ocorrem quando se

deseja estabelecer relações temporais entre mídias distintas. Para isso o usuário precisa interagir com serviços de baixo nível dos objetos *TimeSyncronizable*, *TimeSlave* e *Event*, derivados do objeto EnhancedPremo definido no componente fundamental de PREMO[11]. É desejável, portanto, que o usuário seja poupado dessa tarefa.

A "programação" (construção) de uma aplicaçao multimídia na plataforma é realizada dinamicamente através da interação entre o usuário e os serviços do suporte. No esquema adotado os objetos da aplicação deverão ser criados e encapsulados em um grupo MSS. O processo de criação dos objetos, agrupamento e posterior ajuste de configuração, de maneira a atender aos requisitos do cliente, requer um conhecimento das interfaces dos serviços do componente MSS e da seqüência de operações a ser realizada. Por outro lado as operações habilitadas num dado instante dependem do estado do objeto. Com o objetivo de prover uma interface para gerenciamento dessas operações a especificação PREMO propõe alguns serviços que definem um ciclo de vida para os objetos. Entretanto esses serviços referenciam apenas às fases de criação e destruição de objetos. Para melhor caracterizar as demais etapas este trabalho propõe um ciclo de vida ampliado, o qual contempla as operações de gerenciamento relacionadas com a configuração, negociação de QoS e execução. O objetivo desse ciclo de vida é definir um protocolo que assegure a manipulação das propriedades dos objetos de forma consistente. A plataforma visa principalmente suportar os serviços do ciclo de vida ampliado, de maneira a ocultar as operações internas que são irrelevantes ao interesse do usuário..

3.1. Ciclo de Vida dos Objetos

As aplicações multimídia geralmente envolvem níveis de interatividade bastante intenso entre usuários e sistema para criação, configuração, execução e gerenciamento de objetos: por sua vez os objetos internos do sistema também interagem intensivamente entre si. Em razão disso os relacionamentos entre objetos das diferentes mídias podem ser alterados dinamicamente.

Diante desse contexto fica caracterizada a importância dos modelos de interação entre componentes. Com relação aos componentes PREMO a especificação define em suas interfaces as assinaturas das operações envolvidas. Vale a pena lembrar que PREMO utiliza também o conceito de propriedades, cujas interfaces provêem uma semântica de interação mais flexível onde a definição de tipos e valores para atributos de objetos pode ser retardada para as etapas de configuração e execução da aplicação. Visando disciplinar o uso dessas facilidades propõe-se um ciclo de vida que consiste num modelo de gerenciamento de interações entre objetos. O ciclo de vida define basicamente o conjunto de operações válidas para cada estado em que o objeto possa se encontrar: *Objeto recém criado*, *Configurando*, *Adquirindo os recursos*, *Rodando a aplicação* e *Destruindo*.

3.2. Proposta de Implementação

A proposta de implementação aqui apresentada considera basicamente dois tipos de componentes: componentes básicos do PREMO e componentes de suporte. Este item descreve alguns aspectos da implementação relacionados com os objetos dispositivos virtuais e conexão virtual do PREMO, e com os mecanismos de suporte propostos para sincronização em ambientes distribuídos.

3.2.1. Dispositivo Virtual

Para modelar a operação de um dispositivo virtual deve-se definir pelo menos dois processos: um para a interface *stream* e outro para o dispositivo que objeto efetivamente abstrai. Deve-se ressaltar que objetos dispositivos virtuais estão relacionados com objetos conexões virtuais e/ou grupos no contexto de uma aplicação.

A especificação PREMO estabelece que os dispositivos devem criar os seus objetos de configuração, inclusive os de porta. Entretanto parece mais razoável que a conexão virtual, ou o grupo, decida sobre qual objeto protocolo usar na configuração, ao invés do dispositivo. Dessa forma, o

dispositivo virtual deverá criar as instâncias de protocolo suportadas no domínio e publicar as informações necessárias à identificação dos mesmos. De posse dessas informações, objetos clientes podem definir o tipo ideal de protocolo a ser utilizado, atribuir valores adequados às propriedades do objeto protocolo, configurar portas, instanciar adaptadores de conexão apropriados, etc. Com a finalidade de suprir tais informações, foi acrescentado à interface VirtualDevice um atributo denominado ConfigProtocols.

Uma operação denominada *process* foi também acrescentada à interface *VirtualDevice* e representa o elemento de processamento. A operação *process* foi definida para que o *stream* do dispositivo pudesse comandar o processamento interno do mesmo.

O objeto porta foi modelado como tendo três operações: getData, para obtenção de dados da porta; putData, para colocar dados na porta; e setTarget para indicar o destinatário e qual operação deverá ser evocada quando a porta estiver no modo ControlDriven (ver conexão virtual a seguir).

3.2.2. ConexãoVirtual

A implementação da conexão virtual compreende basicamente três atividades: o estabelecimento da conexão, a aquisição dos recursos e o monitoramento do transporte[18].

Estabelecimento da conexão

Esta atividade tenta realizar o casamento entre as propriedades dos potenciais parceiros de interação. Se houver mais de uma opção para uma determinada propriedade, o suporte deverá negociar e fechar um acordo que melhor atenda aos requisitos definidos pela aplicação. Nesse processo devem ser considerados, por exemplo: os tipos da conexão (LAN, IPC); os tipos de protocolos (TCPTransport, UDPTtransport,etc); os tipos de formatos (MPEG,RawVideo); as características do formato comum escolhido. Para negociar o acordo, a conexão virtual define se é necessário criar instâncias de adaptadores de conexão, verificando se os dispositivos podem, ou não, trocarem dados diretamente.

A conexão virtual também determina para cada componente envolvido na conexão (portas e adaptadores) os tipos de mecanismos de troca de dados a serem usados - *DataDriven* ou *ControlDriven*. Componentes que forem definidos para operar no modo *DataDriven* fornecerão os dados num *buffer* compartilhado de maneira que os componentes *ControlDriven* parceiros que operam no mesmo fluxo peguem os dados. Nas conexões em rede os adaptadores operam no modo *ControlDriven*, enquanto que as portas dos dispositivos virtuais operam no modo *DataDriven*. Dessa forma o adaptador irá pegar os dados na porta de saída do dispositivo fonte e irá colocá-los na porta de entrada do dispositivo destino.

Aquisição de Recursos

A operação *AcquireResource* em objetos de conexão virtual tenta realizar o casamento entre as restrições de QoS dos recursos virtuais envolvidos. Esse processo consiste em fechar um acordo com o suporte operacional em relação aos níveis de QoS. No entanto, a maior parte do esforço para o estabelecimento desse acordo é desprendido pelo grupo através dos serviços de gerenciamento. Estes serviços mapeam os requisitos de QoS do cliente para intervalos de valores de propriedades de modo que os serviços de gerenciamento tenham margens de negociação para viabilizar um acordo na seleção de níveis de QoS que possa aumentar o benefício da aplicação

Monitoramento do Transporte

O monitoramento do transporte é um processo que observa a largura de banda usada pela conexão. A realização desse processo depende do tipo do transporte empregado, podendo ser realizado no adaptador de conexão virtual, na porta fonte ou na porta destino. Informações de estado podem ser armazenadas como atributos da conexão virtual, ou como valores de propriedades.

3.2.3. Suporte para sincronização de fluxos de mídia distribuídos

A integração de diferentes mídias no contexto de sistemas distribiuídos é bastante complexa uma vez que envolve vários componentes, tais como, sistemas de comunicação, sistemas operacionais, bancos de dados, e a própria aplicação em si, a medida que incorpora nos dados informações intrínsecas de sincronização[14].

Nesse sentido foi definida uma arquitetura de suporte para gerenciamento de sincronização envolvendo todas as fases do processamento dos fluxos[17]. A arquitetura está estruturada em três níveis: gerenciamento, sincronização de alto nível e sincronização de baixo nível. No nível inferior tem-se as funções de sincronização de baixo nível, onde se processam as informações necessárias à realização da sincronização: nesse nível pode ocorrer eventuais alterações nos parâmetros de QoS locais ao domínio. Os serviços dessa camada se destinam à sincronização intermídia de granularidade fina. Para a definição dessa camada adotou-se as funcionalidades oferecidas pelo ACME[19]. Acima deste estão os serviços definidos pelo MSS para o controle/monitoramento de fluxos. Como a plataforma assume que os objetos da aplicação são inclusos em um grupo MSS, chama-se essa camada de sincronizador de alto nível, o qual deve controlar as conexões envolvidas pela aplicação e traduzir a QoS especificada pela aplicação para valores significativos para o sistema.

A aplicação interage com o Gerente de Sincronização através de uma ferramenta de interação para estabelecer as conexões entre os dispositivos envolvidos; negociar a QoS entre a aplicação e o sistema; etc. O Sincronizador de Alto Nível controla todos os objetos envolvidos pelo sistema desde a captura até a apresentação da mídia. Ele faz um mapeamento dos requerimentos de QoS que foram negociados com a aplicação para os parâmetros significativos para o sistema [20]: variação do atraso, taxa de transmissão, etc. Nesse nível são também estabelecidas as políticas de sincronização e monitoramento dos fluxos através das Conexões Virtuais.

As interações entre o sincronizador de alto nível e os componentes internos do grupo são realizadas por intermédio dos serviços de propriedades: a definição e/ou obtenção de um valor de propriedade pode expressar, respectivamente, uma restrição de QoS, ou um relatório de comportamento do objeto correspondente. Ao adotar os valores de propriedades para representação das restrições do cliente e dos estados dos recursos, os serviços de sincronização podem se beneficiar dessa flexibilidade para, dinamicamente realizar monitoramento, e, eventualmente implementar políticas de ajustes de sincronização.

4. Implementação e Resultados

Esta seção discute alguns aspectos de projeto, implementação e validação realizados através de um protótipo simplificado para plataforma. O ambiente de implementação para a plataforma constitui-se de: sistema operacional AIX versão 3.25; software para suporte aos objetos distribuídos Orbix[21],[22] versão 1.3; e linguagem de programação C++, com compilador C Set ++ da IBM.

A utilização do Orbix foi motivada pelo fato de ser uma implementação comercial da arquitetura CORBA e a escolha da linguagem C++ deve-se a dois fatores: ela é orientada a objetos, permitindo aplicação dos conceitos da análise com maior facilidade; por restrição desta versão do Orbix, que possui apenas mapeamento IDL-C++.

As versões citadas de sistema operacional e do Orbix possuem algumas limitações, as quais implicaram em restrições para implementação. A maior delas é a ausência de *multi-thread*, um ponto fundamental devido à necessidade de execução de atividades concorrentes pelos objetos. Os impactos causados por estas limitações serão discutidos adiante.

Para testar as interações da conexão virtual com outros objetos, foram criados inicialmente dois dispositivos virtuais simples relativamente aos recursos multimídia (dispositivos *TextFileDevice* para

captura de dados e *XWindowDevice* para apresentação); isso ocorreu porque não havia *hardware* nem *software* especializados nas estações que permitissem a elaboração de dispositivos virtuais sofisticados para processamento de, por exemplo, áudio ou vídeo. Os dispositivos virtuais do exemplo foram concebidos com os objetos de configuração *stream*, formato e protocolo.

O primeiro passo para a implementação consiste na geração das interfaces IDL[21], tomando como base as classes do modelo OMT[24] e os esquemas em *Object-Z*. Uma vez definidas as interfaces IDL, o compilador IDL gera uma classe IDL-C++ contendo informações necessárias à invocação dos métodos de cada interface. O código que implementa os serviços da interface, assim como operações internas não acessíveis ao cliente, é escrito em uma classe C++ denominada classe de implementação. A partir dela constrói-se o servidor e instanciam-se objetos Orbix. Portanto, cada classe IDL-C++ será mapeada em alguma classe de implementação. Convencionou-se para este trabalho que a classe de implementação teria o mesmo nome da classe IDL-C++ seguido de "_i". Por exemplo, a interface da conexão virtual descrita na proposta PREMO é transcrita para IDL da seguinte maneira:

```
interface VirtualConnection : VirtualResource {
      exception ConfigurationMismatch { ConfInfo master;
                                        ConfInfo slave; };
      exception PortMismatch { string reason; };
      void connect(in VirtualDevice deviceMaster,in Port portMaster,
            in VirtualDevice deviceSlave, in Port portSlave);
          raises (ConfigurationMismatch, PortMismatch);
      void disconnect();
      EndpointInfoList getEndepointInfoList();
};
```

A classe C++ que implementa esta interface é:

```
class VirtualConnection_i: public virtual VirtualResource_i {
   protected:
      EndpointInfoList *connection_points;
      VirtualDevice *devMaster, *devSlave;
      unsigned short port_master, port_slave; //identif. das portas
      PortConfig config_master, config_slave; //config. das portas
      unsigned char compare(ConfInfo& ci_master,ConfInfo& ci_slave);
      virtual ConnectionType type_of_connection() = 0;
      SequenceString* match_lists (SequenceString listMaster,
                                   SequenceString listSlave);
   public:
      VirtualConnection_i();
      ~VirtualConnection_i();
      virtual void connect(VirtualDevice* deviceMaster,
                     Port portMaster, VirtualDevice* deviceSlave,
               Port portSlave);
      virtual void disconnect ();
      virtual EndpointInfoList getEndpointInfoList ();
};
```

O adaptador de conexão virtual foi elaborado como um subtipo de *ConnectionAdapter[12]*. Ele foi implementado utilizando o próprio ORB como veículo de transporte e não leva em consideração questões de qualidade de serviço, cujo controle no protótipo se limitou à realização de ajustes através do sincronizador de baixo nível. Uma operação *transfer* foi implementada especificamente para buscar os dados na porta de saída do dispositivo de captura e entregá-los na porta de entrada do dispositivo de apresentação, uma vez que para conexão em rede o adaptador opera sempre desta forma. A Figura 1

ilustra a execução da função *transfer* pelo adaptador (disparada através do *stream* da conexão) que utiliza o ORB para o transporte dos dados.

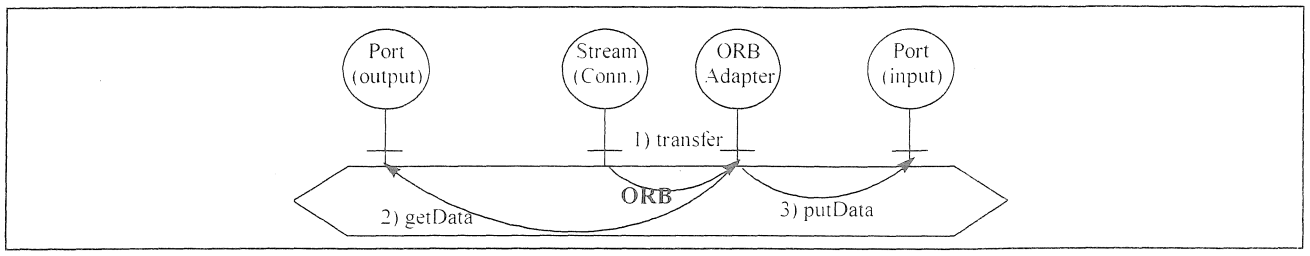

Figura 1: Adaptador realizando a transferência de dados

A abordagem utilizada para o controle das atividades do adaptador (na qual o *stream* da conexão virtual dispara as chamadas *transfer*) não é a mais adequada devido à sobrecarga causada no sistema de comunicação: estes objetos interagem com grande freqüência mas estão em processos diferentes, e até mesmo em máquinas separadas. Outras abordagens foram estudadas para tornar o adaptador de conexão mais independente e são discutidas a seguir: inserir um objeto *stream* (objeto que abstrai as questões de tempo e sincronização) no próprio adaptador; redefinir a interface do adaptador para que ela contenha operações de contole e implementá-lo como uma máquina de estados (equivalentemente ao *Stream*).

As propriedades dos objetos de configuração, assim como da conexão virtual, foram iniciadas estaticamente através de uma função evocada no construtor de cada classe (na realidade pelo construtor da classe base *EnhancedPREMOObject*). Isto tornou-se necessário pois não foi implementado um mecanismo de registro (banco de dados) que permitisse armazenar as propriedades de cada classe.

Um fator limitante para validar a arquitetura de sincronização foi o ambiente não suportar o processamento de informações multimídia; dessa forma, visando obter resultados mais significativos (envolvendo mídia contínua), decidiu-se migrar o esquema de suporte proposto para computadores IBM PS/2 (CPU 486, 33MHz, 16Mb de memória RAM) rodando o sistema operacional IBM OS/2 v2.1 e dotados dos seguintes recursos: uma placa *M-Audio* capaz de capturar/reproduzir áudio no formato .WAV[25]; um *kit M-Box* com alto-falantes e microfone que possibilita controlar volume, nível de gravação, nível de agudos e balanço dos canais de som; e uma câmera de vídeo e uma placa *ActionMedia II*, que gera arquivos de vídeo no formato .AVS

Esses computadores foram ligados em rede, com possibilidade de escolha entre a FDDI ou Ethernet, tendo sido feito testes com as duas. Em ambos os casos o protocolo de comunicação utilizado foi o TCP/IP. A idéia seria utilizar o MMPM/2[26] para controlar as placas *M-Audio* e *ActionMedia II*. A implementação da captura, transmissão e reprodução do áudio foi realizada sem grandes dificuldades. Entretanto verificou-se que o *software* não seria capaz de trazer a imagem capturada pela placa para o espaço de endereçamento da aplicação. A solução foi a sua substituição por texto.

Assim sendo, o sistema ficou da seguinte forma: do lado da captura/transmissão, um microfone gera o sinal de áudio para a placa *M-Audio* a partir de uma voz que lê um texto que é apresentado em uma janela no monitor deste nó. Texto este que se encontra armazenado em um arquivo em disco. O texto é recuperado com uma certa freqüência, ou seja, a cada 93ms apresenta-se um caracter no monitor. Esta freqüência foi determinada de modo que possibilitasse uma leitura normal do texto à medida em que estivesse sendo apresentado. Estas duas mídias, caracteres e voz, são transmitidas através de datagramas para um outro computador semelhante. No lado da recepção/reprodução ocorre o inverso: o som é recebido e enviado à placa *M-Audio* que o reproduz através da *M-Box*; e o texto é exibido no monitor. Duas redes diferentes foram empregadas para testar o sistema: uma FDDI e uma Ethernet.

A implementação do sincronizador de baixo nível foi realizada na linguagem C; adotou-se um

esquema de sincronização dirigido-a-conexão (os dados são exibidos à medida que chegam no nó destino), podendo haver a eleição de um fluxo mestre ou não. O fluxo mestre não deve sofrer influência da sincronização, ele é exibido à medida que seu fluxo chega ao receptor. A sincronização se realiza apenas nos fluxos escravos que seguem estritamente a reprodução do mestre.

As ações corretivas têm a mesma funcionalidade do ACME e a abstração de sincronização é baseada na definição, pelo cliente, de dois parâmetros:

- *desvio em números de unidades (Skew_E):* duas unidades de mídias diferentes, que chegam juntas ao destino, não podem ser exibidas caso a diferença ordinal entre elas ultrapasse o valor Skew_E. Em outras palavras, mede a defasagem em relação ao número de unidades; e
- *desvio temporal (Skew_D):* duas unidades de mídias diferentes que possuem o mesmo carimbo de tempo não podem ser exibidas caso a defasagem temporal entre elas seja maior do que o valor Skew_D, ou seja, mede a defasagem temporal entre duas unidades.

A taxa de amostragem do áudio foi de 11 025 vezes por segundo. O tamanho da unidade, 370ms, foi o menor tamanho conseguido utilizando-se os recursos disponíveis. Aumentando-se este valor, o atraso entre a captura e reprodução também era aumentado na mesma quantidade, criando-se um eco entre os dois sistemas.

Considerando as variações nos parâmetros de controle e carga na redes foram realizados 288 testes. Os dois gráficos apresentados a seguir (Figuras 2 e 3) ilustram a operação do sincronizador mostrando resultados sem nenhum controle (Figura 2) em um experimento que foi realizado sem fluxo mestre, usando Skew_E=1, e Skew_D=700ms.

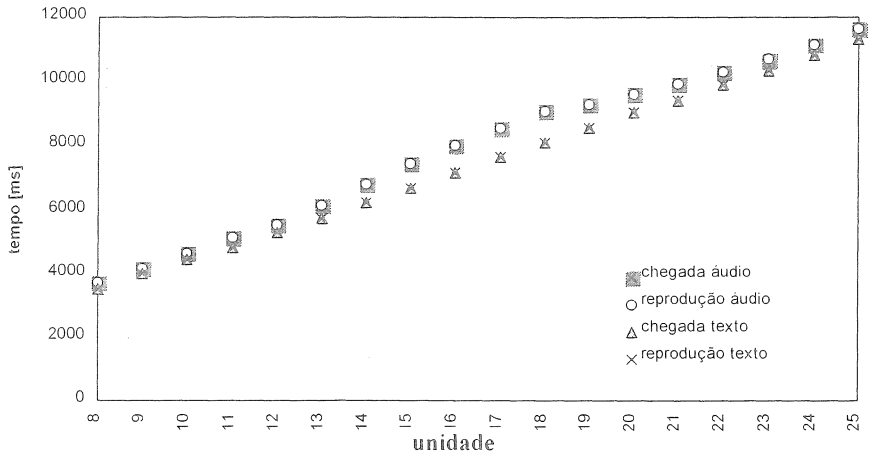

Figura 2: FDDI com pouco tráfego, sem fluxo mestre e sem sincronismo

No Gráfico da Figura 2 é possível notar que no intervalo de 5500ms a 9200ms houve uma defasagem temporal entre as unidades de texto e áudio; logo após a rede conseguiu enviar as unidades atrasadas, e o sistema as reproduziu sem que ocorresse nenhuma perturbação nos seus fluxos.

Aplicando-se a sincronização (Figura 3) é possível verificar a atuação do sincronizador em relação ao fluxo do gráfico anterior.

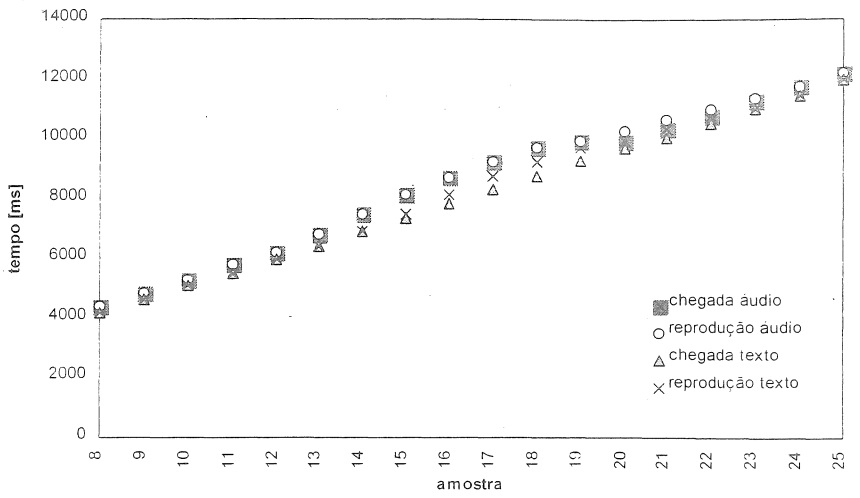

Figura 3: FDDI com pouco tráfego, sem fluxo mestre, Skew_E=1 e Skew_D=700ms

A Figura 3 mostra que no período em que ocorreu a desfasagem (5 500ms a 9 200ms) a exibição das unidades de texto tentou acompanhar a chegada das unidades de áudio. Foi utilizado nesse teste os parâmetros Skew_E= 1 e Skew_D= 700ms. Isto significa que a defasagem máxima permitida em relação ao número de unidades é igual a um e que a defasagem máxima em relação ao tempo é igual a 700ms. Quando a unidade 15 de texto chega, a 14 já foi exibida mas a unidade 14 de áudio ainda não chegou. Devido a isto a unidade 15 de texto deve esperar a chegada e, conseqüentemente, a reprodução da unidade 14 de áudio. Este comportamento continua até a unidade de número 19, quando ocorre a chegada das unidades atrasadas, que são reproduzidas o mais rápido possível na tentativa do sistema voltar à sua taxa normal de operação.

5. Conclusão

A decisão de escolher a arquitetura de serviços MSS foi correta no sentido de que esse *framework* é um modelo poderoso e flexível. Contudo, devido ao fato do MSS ter um nível de generalidade excessiva, provoca uma certa dificuldade de interpretação e uso das abstrações definidas. Nesse sentido a disponibilidade de uma plataforma com um conjunto de ferramentas e bibliotecas de objetos auxilia o processo de desenvolvimento e criação de aplicações.

Ademais, seus conceitos estão sendo utilizados como fundamentos para outras definições, como é o caso do recente documento *Control and Management of A/V Streams* [27]. O referido documento é a submissão da Lucent Technologies Inc (Bell Labs) ao RFP (*Request for Proposal*) da OMG para criação de facilidades de controle e gerência de *streams* de mídia, no contexto da arquitetura CORBA/OMA. Nele são empregadas amplamente abstrações como dispositivo virtual, conexão virtual, porta e formato.

Com relação ao ambiente de desenvolvimento empregado, pode-se dizer que ele é inadequado para a aplicação em questão. Aplicações multimídia exigem muito dos recursos do sistema, que a configuração utilizada não conseguiu suprir. Em função disso foi tentado validar a arquitetura de sincronização proposta em um ambiente que suportasse mídias contínuas; apesar das limitações observadas na validação, pode-se, ainda assim,concluir que o trabalho cumpriu os seus objetivos permitindo o contato com a tecnologia de integração dos futuros sistemas distribuídos. Como continuidade do trabalho a implementação está sendo migrada para uma plataforma RMI/Java[23].

6. Referências

[1] BLAIR, G. S., COULSON, G., DAVIES, N. **System Support For Multimedia Applications: An Assessment Of The State Of The ART.** ftp://ftp.comp.lancs.ac.uk/pub/mpg/MPG-93-29.ps.Z

[2] GIBBS, S. J., TSICHRITZIS, D. C. **Multimedia Programming - Objects, Environments and Frameworks.** Addison-Wesley, 1994.

[3] OMG **The Commom Object Request Broker: Architecture and Specification.** Revision 1.2, December, 1993.

[4] BEN-NATAN, R. **CORBA: A Guide to the Common Object Request Broker Architecture.** McGraw-Hill, 1995.

[5] ISO/IEC 10746: **ODP Reference Model Part 1, Overview.** June 1995.

[6] ISO/IEC 10746: **ODP Reference Model Part 2, Foundations.** June 1995

[7] ISO/IEC 10746: **ODP Reference Model Part 3, Architecture.** June 1995

[8] ISO/IEC 10746: **ODP Reference Model Part 4, Architectural Semantics.** June 1995

[9] INTERACTIVE MULTIMEDIA ASSOCIATION. **Multimedia Systems Services** version 1.2, 1994

[10] ISO/IEC 14478-1: **Presentation Environment for Multimedia Presentation (PREMO) - Part 1: Fundamentals of PREMO.** 1996.

[11] ISO/IEC 14478-2: **Presentation Environment for Multimedia Presentation (PREMO) - Part 2: Foundation Component.** 1996.

[12] ISO/IEC 14478-3: **Presentation Environment for Multimedia Presentation (PREMO) - Part 3: Multimedia System Service Component.** 1996.

[13] BLUM, C., MOLVA, R. **A Software Plataform for Distributed Multimedia Applications.** In: PROCEEDINGS OF INTERNATIONAL WORKSHOP ON MULTIMEDIA SOFTWARE DEVELOPMENT, March 1996, Berlin.

[14] STEINMETZ R.: **Analyzing the Multimedia Operating System.** In: IEEE Multimedia, Spring 1995.

[15] PARTRIDGE C. Gigabit Networking , Reading MA: Addson-Wesley, 1994.

[16] MADEIRA, E. R. M. **Software Architecture for Multimedia Communication and Management.** Relatório Interno. Instituto de Computação,UNICAMP, Brasil, 1996

[17] TIZZO, N.P.: **Sincronização de Fluxos de Dados em Aplicações Multimídia.**Dissertação de **Mestrado**. Faculdade de Engenharia Elétrica e de Computação. Universidade Estadual de Campinas (UNICAMP), março de 1997.

[18] DEESIDERÁ L.: **Arquitetura de Conexão de Dispositivos Multimídia em Ambientes Distribuídos**. Dissertação de Mestrado. Faculdade de Engenharia Elétrica e de Computação. Universidade Estadual de Campinas (UNICAMP), maio de 1997.

[19] ANDERSON, D. P., HOMSY G.: **A Continous Media I/O Server and its Synchronization Mechanism.** In: IEEE Computer Magazin, Vol 24, N. 10, 1991.

[20] SHERPHERD D. at all. **Protocol Support for Distributed Multimedia Applications. In:** Computer Communications. Vol 15, No. 6 jul/ago 1992.

[21] IONA Technologies Ltd. **Orbix distributed object technology, Programmer's Guide.** Release 1.3.1, February 1995.

[22] IONA Technologies Ltd. **Orbix distributed object technology, Advanced Programmer's Guide.** Release 1.3.1, February 1995.

[23] ORFALI, R., HARKEY, D. **Client/Server Programming with Java and CORBA.**. John Willey & Sons. !997.

[24] RUMBAUGH J., BLAHA M., PREMERLANI W., EDDY F., LORENSEN W. **Object-Oriented Modeling and Design.** New York: Prentice Hall International, 1991. 500 p

[25] MICROSOFT. **New Multimedia Data Types and Data Techniques, Microsoft Multimedia, Standards Update.** Revision 1.0.97, march 1995.

[26] IBM. **Multimedia Presentation Manager Programming Reference and Programming Guide 1.0, IBM.** Form: S41G-2919-00 and S41G-2920-00, march 1992.

[27] LUCENT TECHNOLOGIES INC. **Control and Management of A/V Streams.** Version 1.0, feb. 1997. http://www.omg.org/docs/telecom/97-02-03.ps