UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Framework para implementação de serviços transmissores de vídeos voltados a PCs e dispositivos móveis, perceptivo à mudança contextual de localização.

Fernando de Moraes Jardim

São Carlos
agosto/2004

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

J37fi

Jardim, Fernando de Moraes.
Framework para implementação de serviços transmissores de vídeos voltados a PCS e dispositivos móveis, perceptivo à mudança contextual de localização / Fernando de Moraes Jardim. -- São Carlos : UFSCar, 2004.
94 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2004.

1. Sistemas multimídia. 2. Framework (programa de computador). 3. Computação ubíqua. 4. Sistemas de segurança. 5. Middleware. I. Título.

CDD: 006.7 (21ª)

“ The dificulty lies not in new ideas,
but escaping from old ones”.
(John Maynard Keynes)

Agradecimentos

Agradeço primeiramente aos meus pais Roberto e Lídia. Eles são meus maiores professores.

Aos meus irmãos Daniela e Roberto Filho, que tanto vibram com minhas conquistas, e, que de certa forma são nossas conquistas.

Ao meu orientador, Prof. Dr. Luis Carlos Trevelin, que com toda sua serenidade e boa vontade me incentivou e me instruiu para que chegasse ao final do mestrado. Ele, além de professor, mostrou-me ser meu amigo.

Aos amigos do grupo GSDR da UFSCar que compartilharam conhecimentos técnicos e sociais, a Bruno Amaral, Eugeni Dondov e em especial Érico Mattos. Eles sem dúvida me apoiaram nos momentos mais difíceis enfrentados nas madrugadas no laboratório, e hoje são grandes amigos.

À minha família que adquiri mudando para São Carlos, principalmente Wesley, o qual compartilhei excelentes momentos por morarmos todo esse tempo debaixo do mesmo teto, a Cristina esposa de meu orientador o qual foi minha "mãe" em São Carlos, ao amigo Renato, Lúcio e Vinícius.

Aos amigos de Goiânia, que vieram me visitar, confirmando mais uma vez a lealdade da amizade existente, em especial o Fabiano (cunhado), Marcelo de Paula, Marcelo Veloso e Diogo.

E agradeço a uma grande amiga e companheira, Pamella, que me aturou nestes últimos 13 meses, e que por sinal, tive o prazer de conhecê-la meados do mestrado em São Carlos. Ela me ajudou a ficar acordado nos momentos difíceis enfrentados, com paciência e compreensão.

Resumo

Com o avanço da tecnologia na área de transmissão de dados e o fato dos computadores estarem presentes cada vez mais em diversos equipamentos, como PDAs (*personal digital systems*), celulares, carros, geladeiras, etc, se vê necessário estabelecer um protocolo comum de comunicação entre estes diversos computadores. O trabalho apresentado nessa dissertação tem como objetivo definir uma infra-estrutura necessária à implementação de serviços de transmissão de vídeos para aparelhos móveis (PDAs e celulares) com transparência na mudança de contexto de localização. Essa mudança de contexto representa a troca de ambiente físico por parte de um usuário do sistema. A idéia é que o usuário continue a assistir a um vídeo (em um dispositivo móvel) quando sair do local onde o estava assistindo. Para isso, foi modelado e implementado um *framework* perceptivo à mudança contextual localista destinado a serviços transmissores de vídeos para dispositivos móveis. Alto grau de mobilidade e transparência é característica de computação ubíqua ("Nova era da computação" [12]). Para um maior embasamento teórico foi realizado uma pesquisa sobre os conceitos relacionados com computação ubíqua, transmissões de vídeos e dispositivos móveis. Este trabalho foi desenvolvido no laboratório de Sistemas Distribuídos e Redes (GSDR) da Universidade Federal de São Carlos – UFSCar.

Abstract

With the progress of the technology in data transmission area and the fact of computers be present more and more in several equipments, like PDAs (Personal Digital Systems), mobile telephones, cars, refrigerators, etc, it becomes necessary to establish a standart communication protocol among these several computers. The work presented in this monograph has as objective to define a necessary infrastructure to implementations of videos transmission services for mobile devices (PDAs and mobile telephones) with transparent change of location context. This context change represents the change of physical atmosphere of the proposed system user. The idea is that the user continues to watch a video (in a mobile device) when he leaves the place where he was watching it. A framework was modeled and implemented to detect change of physical atmosphere destined to videos transmission services directed to mobile devices. High degree of mobility and transparency is characteristic of Ubiquitous Computation (Computation new moment - Weiser). For a theoretical basement was carried through a research on the concepts related with Ubiquitous Computation, mobile devices and video transmissions. This work was developed in the Network and Distributed Systems Laboratory of São Carlos Federal University - UFSCar.

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

Neste capítulo será observado o contexto em que o trabalho se enquadra, os fatos que causaram motivação para tal, e o objetivo principal para a realização do mesmo.

1.1. Contexto do Trabalho

Com o avanço da tecnologia na área de transmissão de dados e o fato de avançados recursos computacionais estarem presentes em diversos equipamentos, como PDAs, celulares, carros, geladeiras, etc, se vê necessário estabelecer um protocolo comum de comunicação entre estes diversos dispositivos computacionais [1]. Um exemplo de comunicação é a capacidade de transmissão de vídeos entre um servidor e diferentes tipos de dispositivos, como por exemplo: aparelhos celulares, PDAs, Televisões Digitais, além dos PCs e *notebooks* bastante utilizados no cotidiano brasileiro.

A programação convencional (orientada 'de' e 'para' computadores pessoais) é amplamente conhecida e divulgada nos meios acadêmico e comercial. A modelagem e programação para os equipamentos embarcados (conhecidos como *embedded systems*), necessitam de cuidados adicionais aos da programação convencional, tais como: requisitos de *hardware* e *software* diferenciados, tamanho de memória (ROM e RAM), possibilidade e espaço para armazenamento de informações, entre outros. Exemplos de equipamentos embarcados são: PDAs e celulares.

A tecnologia *Java* atende os quesitos de comunicação, permitindo tráfego de imagens e vídeos entre equipamentos embarcados, o qual seu ambiente para o desenvolvimento de aplicações voltadas a dispositivos embarcados é o J2ME (*Java* 2 *Plataform MicroEdition*).

Banavar [2] cita um exemplo no qual uma pessoa, que estava assistindo presencialmente a uma conferência precisa sair do espaço físico onde a mesma está sendo realizada. O sistema proposto por ele deve ser capaz de perceber de forma implícita que esta pessoa não está mais presente na sala, e começar a transmitir a conferência para o seu PDA. Quando essa pessoa entra em seu carro é detectado que uma tela situada atrás do banco é capaz de fornecer com maior qualidade a conferência e o sistema deixaria de transmitir para o PDA e passaria a transmitir para essa tela. E assim continua, sempre agindo de modo transparente e com total mobilidade.

Diante deste cenário que representa um alto grau de mobilidade destinado a aplicações multimídia, serão apresentadas especificações e definições da tecnologia sem fio, e dos recursos computacionais que dão suporte a esta tecnologia.

1.2. Motivação

A transmissão de mídias (vídeos, imagens, textos, etc.) entre diferentes dispositivos pode trazer várias facilidades à humanidade. Atualmente, no Brasil, já é possível enviar e receber imagens e textos utilizando telefones celulares, até mesmo tirar uma fotografia utilizando estes dispositivos e enviá-la naquele instante para uma outra pessoa que possua um aparelho celular, ou mesmo enviá-la para um endereço de e-mail.

Entretanto, estes recursos geralmente são usados para diversão ou entretenimento, e acaba sendo desprezado sua usabilidade em um ambiente educacional, profissional ou mesmo familiar. Com a tecnologia atual, por exemplo, pode-se verificar através de um celular como a babá esta cuidando do seu bebê em sua casa. Para isso basta que em sua casa possua algumas câmeras espalhadas os quais recebem as imagens e as enviam para um endereço específico (seu telefone celular). Um outro exemplo é utilizar este mesmo recurso para um sistema de segurança de um prédio, no qual, através de um PDA se pode visualizar as imagens capturadas pelas câmeras espalhadas sobre o prédio. Isto proporcionaria ao responsável pela segurança uma maior mobilidade, e ele poderia fazer sua inspeção noturna observando em seu PDA qualquer invasão.

Percebe-se então que a transmissão de vídeos pode estar disponível a várias pessoas em diferentes localidades e em diferentes tipos de equipamentos, desde computadores comuns (PCs) a PDAs e telefones celulares. Com isso seria possível ao usuário:

- Assistir a um vídeo provindo de câmeras diversificadas (*webcam*, câmera digital, câmera VHS, etc.) utilizando dispositivos computacionais comuns (PC) ou embarcados (PDAs e telefones celulares) destinado a diversas situações, sejam elas profissionais, educacionais familiares ou de entretenimento..

1.3. Objetivo

A infra-estrutura necessária para realizar videoconferências [3] e transmissões de vídeo sob demanda [4] já foram definidas e estudadas por vários pesquisadores. O trabalho apresentado nesta dissertação tem como objetivo definir uma infra-estrutura necessária à implementação de serviços de transmissão de mídias para aparelhos móveis (PDAs e

celulares) com transparência na mudança de contexto de localização. Esta mudança de contexto representa a troca de ambiente físico por parte de um usuário do sistema proposto. A idéia é que o usuário continue a assistir a um vídeo (em um dispositivo móvel) quando sair do local onde estava assistindo-o. Foi modelado e implementado um *framework* perceptivo à mudança contextual localista destinado a serviços transmissores de mídias para PCs e a dispositivos móveis.

1.4. Estrutura da dissertação

O texto referente a essa dissertação de mestrado encontra-se dividido da seguinte forma:

- Estado da arte (Capítulo 2): é feita uma abordagem dos assuntos e de tecnologias referentes ao objetivo desse projeto, como transmissão de vídeo para diferentes dispositivos, softwares existentes, computação ubíqua, e tecnologias relacionadas com o projeto (Java, plataforma JAMP, sensores, etc);
- Projeto dos *Framework*s (Capítulo 3): definição da arquitetura proposta para o alcance do objetivo do projeto, integração dessa arquitetura com o plataforma JAMP, e especificação das classes presentes nos *framework*s desenvolvidos;
- Implementação dos *Framework*s (Capítulo 4): detalhamento do desenvolvimento dos *framework*s e de suas classes e códigos de algumas classes principais;
- Estudo de caso (Capítulo 5): desenvolvimento e implementação de um estudo de caso baseado nos *framework*s desenvolvidos;
- Resultados obtidos (Capítulo 6): Exibição dos resultados obtidos com o desenvolvimento dos *framewoks* e estudo de caso. Conclusão de todo o projeto de mestrado realizado.

2. Estado da Arte

Para alcançar o objetivo proposto, foi realizado um levantamento dos dispositivos, pré-requisitos e arquiteturas existentes, referentes ao envio de uma mídia (em especial vídeo) para PCs e dispositivos móveis. Neste capítulo, são mostrados alguns protótipos, dispositivos, e conceitos relacionados à computação ubíqua.

2.1. Transmissão de vídeo para diferentes dispositivos

O envio/recebimento de uma mídia através de um dispositivo computacional para outro se dá respeitando a arquitetura definida na figura 1. Note que se trata de uma arquitetura cliente/servidor de mídia. Uma mídia pode ser transmitida a partir de um arquivo já armazenado no servidor ou em tempo real, provinda de imagens/áudios capturados a partir de câmeras/microfones conectados ao servidor.

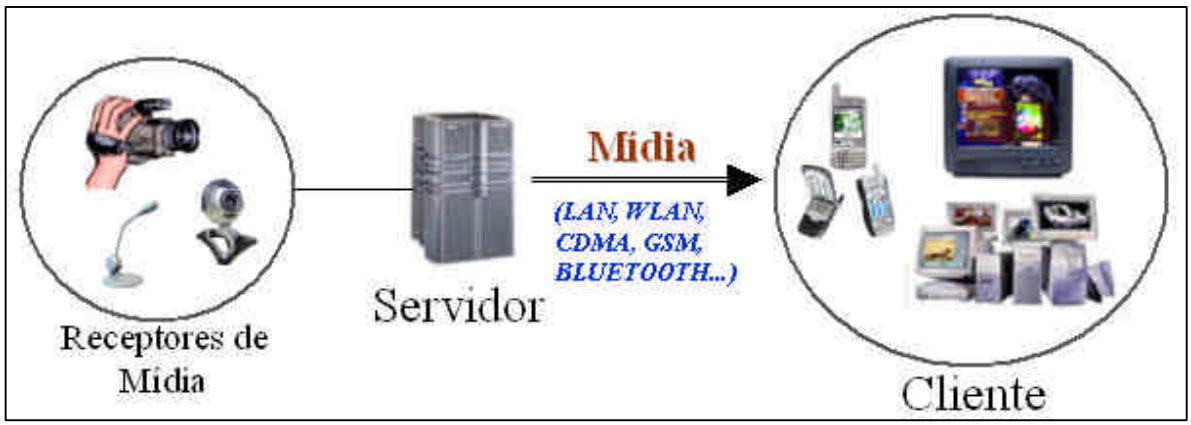

Figura 1: Recursos necessários para a recepção e envio de um vídeo

O envio de um vídeo capturado a partir de uma câmera para um computador se dá respeitando os seguintes passos: 1 – Captura do sinal (câmeras/microfones); 2 – Recepção e compactação digital do sinal; 3 – Transmissão do vídeo digital ao dispositivo alvo.

A captura do sinal pode ser representada por uma câmera filmadora convencional (usadas para gravar vídeos em fitas VHS), por uma *webcam, por uma* câmera de segurança, etc.

O computador que recebe os sinais provindos de uma câmera conectada a ele deve estar preparado para receber estes sinais. Ele deve tratar e, geralmente, compactar a imagem e o som capturado. Para isso ele deve possuir algumas características apropriadas para permitir a função de “receptor” de câmeras, como por exemplo

memória RAM disponível (ex: 256Mb), processador robusto (ex:1Ghz), disco rígido com grande espaço de armazenamento (ex: 20Gb) além de possuir uma porta de entrada para a conexão com a câmera.

Essa porta de entrada pode ser representada por várias interfaces: porta serial/paralela do computador (não permitem taxas de transmissões com alta qualidade); porta USB (atualmente é o padrão comercial mais utilizado para soluções que não exigem alta qualidade devido ao preço e à facilidade de instalação e utilização); porta *firewire* (para conexão de câmeras de padrão digital – DV que trabalham com vídeo de alta qualidade, e permite comunicação bidirecional com a câmera).

É importante ressaltar que as placas de vídeos podem receber sinais analógicos provindos de uma câmera convencional. Ela pode digitalizar o sinal analógico de entrada, tornando-o processável pelo computador. Ela pode também possuir *"codecs"* em *hardware*, que "economizam" processamento principal.

Após a captura do vídeo é gerado um arquivo binário. A resolução e qualidade do vídeo (tamanho, número de cores, qualidade da imagem) dependem do formato utilizado pelo programa de captura ou gravação. Esse programa geralmente é um "*encoder*" e pode trabalhar com vários tipos de "*codecs*" (algoritmos de compactação/descompactação de uma mídia).

Os arquivos gerados podem ser editados e disponibilizados como simples arquivos para *downloads*. Alguns formatos de arquivo permitem o "*progressive-download*", que proporciona a visualização da parte do vídeo que já foi capturado, possibilitando acompanhar o conteúdo sem a necessidade de receber todo o arquivo. Outra opção de transmissão é utilizar formatos de *stream*, que possibilitam a transmissão de áudio e vídeo ao vivo utilizando tecnologias de rede LAN (*Local Area Network*), WLAN (*Wireless* LAN), *Internet* ou mesmo tecnologia destinada à telefonia celular (ex: CDMA, GSM).

Para realizar *stream* de áudio e vídeo pela *Internet*, é preciso controlar o fluxo contínuo de som e imagem. Esta tarefa é realizada por servidores de vídeo que retransmitem aos *players* o sinal já digitalizado e compactado pelo *encoder*, gerenciando as taxas de transmissão para cada usuário. Alguns *softwares* de *encoder* realizam funções de servidor, dispensando o uso de um terceiro programa ou equipamento.

2.1.1. Digitalização e formato de vídeos

A digitalização de vídeo consiste em armazenar as informações (cores, posição, etc.) de cada ponto de uma imagem estática capturada em um *pixel*. O conjunto de *pixels* resulta em um quadro de um vídeo, ou seja, um vídeo digitalizado pode ser visto como um conjunto de quadros (imagens seqüenciais). A digitalização pode ser feita de um vídeo já armazenado (em uma fita VHS, por exemplo) ou um vídeo oriundo de uma câmera analógica, e pode ser realizada tanto via *software* como *hardware*.

Para a digitalização via *software* são utilizados os recursos compartilhados do computador (memória RAM, processador, etc) para o processo, já para a digitalização via *hardware* as funções referentes ao processo são de responsabilidade da placa receptora (placa de vídeo – possui gerenciamento próprio: memória, processador, etc).

Ao digitalizar um vídeo [5] pode-se aplicar técnicas para a compactação dos quadros obtidos. Existem hoje diversos *codecs* com a principal funcionalidade de comprimir dados redundantes. Isso resulta em arquivos de vídeos com um menor tamanho, exigindo assim uma menor largura de banda da rede para a transmissão.

Geralmente as extensões de arquivo destes vídeos digitalizados são: AVI, MOV e MPEG. Quando não é aplicado nenhum algoritmo de compactação na digitalização temos um "AVI puro", senão, um formato compactado. É importante dizer que AVI é um padrão de vídeo para o sistema operacional *Windows*, e que um arquivo no padrão MPEG, MOV ou em algum outro padrão podem estar representado por um AVI.

A qualidade do vídeo digitalizado depende principalmente da quantidade de quadros capturados por segundo e da qualidade de cada quadro, que pode ser exprimida pela quantidade de *pixels* utilizados (dimensão da tela) e da quantidade de informação em cada *pixel* (variação das cores).

A transmissão de vídeos digitais entre dispositivos computacionais pode atender várias necessidades do nosso cotidiano. Como exemplo: assistir através de um *notebook* como a babá está cuidando do seu bebê naquele instante; assistir em seu PC a final do seu time favorito de futebol; visualizar em um PDA, enquanto é feito a ronda noturna, todas as entradas de um prédio, como um maior controle de segurança; realizar uma videoconferência, etc.

É importante ressaltar que existe uma diferença entre transmissão de vídeo e videoconferência, apesar da videoconferência também ser uma transmissão de vídeo, ela permite a interação dos dois pontos que se comunicam.

2.1.2. Dispositivos aptos a visualizar um vídeo digital

Após o processo de digitalização de um vídeo, ele estará pronto para a visualização em um dispositivo computacional. Entretanto, ao se tratar de uma transmissão em tempo real, é preciso verificar se os recursos de transmissão entre servidor/cliente do vídeo suportam a largura de banda exigida pelo formato do vídeo gerado; se o recurso computacional utilizado como cliente possui o conjunto de *hardwares* (memória disponível, processamento, etc) e *softwares* (Sistema Operacional, programa para visualização e *codec* instalado do vídeo recebido) necessários para a visualização em seu *display*.

Existem atualmente vários dispositivos aptos a receberem um vídeo digital. Todavia, serão mostradas a seguir algumas características dos dispositivos computacionais de pequeno porte, como os PDA´s, aparelhos celulares e os *smartphones*.

2.1.2.1. PDAs

O primeiro *handheld* (computador de mão) registrado na história (figura 2) foi o *Newton*, e foi desenvolvido em 1993 por *John Sculey*, da empresa *Apple*. A sigla PDA é a abreviação de *Personal Digital Assistant* (assistente pessoal digital). As características marcantes de um PDA são: pequeno tamanho (cabe em um bolso de camisa), possui funções de computadores comuns (PCs), exemplo: edição de textos, planilhas, capacidade de armazenamento e recuperação de informação. Um PDA possui toda a arquitetura necessária para executar aplicativos específicos, como por exemplo, um aplicativo de conexão com a Internet, um aplicativo que gerencia os dados de uma empresa, ou mesmo um aplicativo capaz de receber vídeos por *stream*.

Figura 2: Newton – 1° PDA desenvolvido por John Sculey, da Apple

O *Palm* é um modelo de PDA, e foi criado por *Jeff Hawkins*, em abril de 1996. Ele foi um sucesso comercial. O nome *Palm* e *PalmTop* virou sinônimo de computador de mão, o que permanece até hoje.

A *Microsoft* foi proibida de usar somente o nome *Palm* [HRef-1]. Ela criou então um outro nome, o *PalmPC* (atual *PocketPC*). No ano de 2003, no Brasil, a versão do *Palm* mais avançada foi o *Tungsten* T3 (figura 3). Ele é um "computador de mão" que possui um processador Intel de 400MHz, 64MB de memória RAM, 16MD de ROM, e uma tela de 320x480 *pixels* com 16 *bits* de cores. Além do *hardware* consideravelmente avançado pelo tamanho do dispositivo os *softwares* que o acompanha permite que seu usuário execute funções como: envio/recebimento de e-mail, reconhecimento de escrita, recepção de um vídeo, planilhas, textos, etc.

Figura 3 - Palm Tugsten T3 – 400MHz, 64MB RAM

Empresas como HP, *Compaq*, *Dell*, *Itautec* e *Toshiba* investiram na tecnologia de PDAs, e lançaram versões que atualmente concorrem com os PDAs da empresa Palm. A plataforma utilizada para estes PDAs recebeu o nome de *PocketPC*. Ou seja, existem duas plataformas definidas e mais utilizadas atualmente para PDAs: *Palm* e *PocketPC*. A plataforma *Palm* suporta somente o sistema operacional definido pela *Palm*, já o *PocketPC* permitem a instalação do sistema operacional *Windows* (*WM2003*, *PocketPC2002/2003*) e até mesmo do *Linux*.

2.1.2.2. Aparelhos celulares e smartPhones

O estudo da tecnologia de aparelhos celulares existe desde 1947, e iniciou-se com a *Bell Laboratories*, no departamento de pesquisa da AT&T (operadora norte-americana na época). Em 1970 usava-se variação de freqüências de ondas de rádio para permitir comunicação entre os sistemas móveis. Já em 1973, *Martin Cooper*, (gerente geral da Motorola), fez a primeira chamada de um "telemóvel" do mundo através do protótipo *Dyna-Tac* (figura 4). Cooper entrou para a história não só como sendo o primeiro

utilizador, mas também por ser considerado como o inventor deste novo meio de comunicação. [HRef-2]

Figura 4: Foto do primeiro celular (1973-Motorola Dyna-Tac)

O serviço celular é caracterizado por uma divisão de uma cidade ou região em pequenas áreas geográficas denominadas células, sendo cada uma delas servida pelo seu próprio conjunto de rádios transmissores e receptores de baixa potência. O tamanho das células pode variar na faixa de metros a dezenas de quilômetros, onde uma célula comunica-se com a vizinha.

Inicialmente os telefones celulares enviavam e recebiam apenas sinais de áudio. Com o avanço das tecnologias de rede e dos aparelhos, atualmente os telefones celulares são capazes de enviar dados também. Estes dados podem representar desde simples mensagens textuais até informações multimídia (imagens, áudios e vídeos).

Existem muitos fabricantes de telefone celular atualmente, sendo que os mais conhecidos são: *LG, Motorola, Nokia, Ericsson, Samsung, Toshiba, Siemens* e *Qualcomm*. Estas empresas vêm desenvolvendo telefones celulares que mais se aproximam de um "computador celular", capazes de enviar/receber e-mails, textos, imagens e até pequenos vídeos.

Os *smartPhones* ou telefones inteligentes são aparelhos celulares que desempenham funções de telefones celulares e PDAs. As empresas vêm investindo pesado nestes dispositivos, já que a praticidade que o mesmo oferece é algo desejado por muitos consumidores. Em março de 2004 já era possível comprar um *smartPhone* (figura 5) por R$650,00 (Motorola A388), que é um preço inferior a muitos telefones celulares desta mesma época.

Figura 5: Foto do smartPhone Motorola A388 (telefone celular com funções de PDA).

2.2. Softwares existentes para Envio/Recepção de vídeo

Existem muitos *softwares* que cumprem a função de envio e recepção de vídeo utilizando recursos computacionais, entretanto, a maioria deles exige uma alta largura de banda da rede, que geralmente se encontra na média de 0,5 Mbps.

Exemplo disso é a aplicação *VideoLAN* [HRef-4], que foi desenvolvida por pesquisadores franceses da *Ecole Centrale Paris*. Essa aplicação foi criada com o intuito de prover uma solução completa de *streaming* vídeo para o envio em uma rede local com uma alta largura de banda. Hoje, a aplicação já pode ser utilizada para visualização de vídeos de alta qualidade (MPEG 1, 2 e 4) provenientes de uma rede local (LAN) ou até mesmo de uma WAN (*Wide Area Network*).

A solução para envio de *streaming* de vídeo para esta aplicação (figura 6) envolve duas aplicações: VLS (*VideoLAN Server*) e VLC (*VideoLAN Client*). A VLS pode trabalhar com *stream* de MPEG 1, 2 e 4, DVDs, sinal digital de satélite, sinal digital terrestre de televisão, e transmissões ao vivo, utilizando recursos *unicast* ou *multicast*. A VLC aceita *stream* MPEG 1, 2 e 4, arquivos do tipo DIVX e transmissões de vídeo ao vivo.

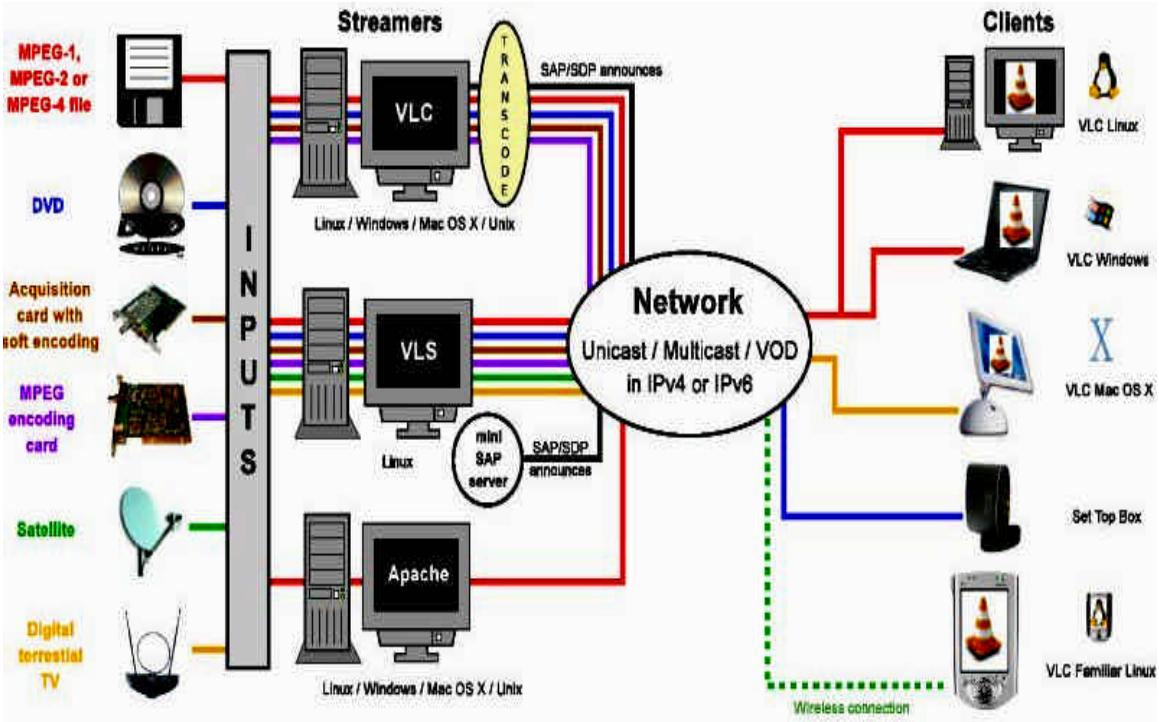

Figura 6: Arquitetura VideoLAN

A largura de banda exigida para transmissão de vídeo utilizando a aplicação *VideoLAN* pode variar conforme o formato do arquivo (tabela 1):

Padrão	Velocidade em Mbps
MPEG-4	*0,5 a 4*
MPEG-2	*3 a 4*
DVD	*6 a 9*

Tabela 1 : Velocidade em Mbps e padrão de vídeo em uma aplicação VideoLAN

Existem, atualmente, várias outras aplicações capazes de transmitir fluxos de áudio e vídeo na Internet:

- ***Windows Media Service da Microsoft*** [HRef-5]: Esta aplicação codifica em seu formato privado, ASF, os seguintes formatos de áudio e vídeo: WAV, WMA, WMV, ASF, AVI, e MPEG-1. Usa os seguintes protocolos para transmissão: UDP, TCP HTTP.
- ***Real System na Real NetWork*** [HRef-6]: Esta aplicação transmite os vídeos nos seguintes formatos: MPEG-1, AVI, WAV, AIF e RM. Usa os seguintes protocolos para transmissão: IPMulticast, TCP, UDP e HTTP.
- ***VideoCharger da IBM*** [HRef-7]: Esta aplicação transmite os vídeos nos seguintes formatos: MPEG-1, MPEG-2, AVI, WAV, LBR e MOV. Usa os seguintes protocolos para transmissão: RTP, TCP, HTTP e IPMulticast.

Além dos servidores de vídeo por *stream*, é necessário que exista a aplicação cliente (receptora) destes "pacotes". Exemplos atuais de aplicações clientes de vídeo por streaming são: *MTV, Windows Media Player, RealAudio e JMF*.

2.3. VideoConferência

Além dos *softwares* responsáveis apenas pelo envio e recepção de vídeos por *stream*, existem *softwares* que utilizam recursos multimídia que são capazes de criar um meio de comunicação entre pessoas e/ou grupos de pessoas. Exemplo disto, são os sistemas de videoconferência, que são utilizados desde a década de 80.

Um sistema de videoconferência [6] é um serviço de teleconferência composto por uma coleção de *softwares* capazes de transmitir imagens estáticas ou dinâmicas (vídeo), arquivos (por FTP, por exemplo), mensagens em forma de textos, através de dispositivos computacionais.

Teleconferência pode ser vista como um conjunto de facilidades de telecomunicações [7] que permite que um ou mais participantes em localidades distintas estabeleçam uma comunicação bidirecional através de dispositivos eletrônicos de comunicação, enquanto compartilham, simultaneamente, seus espaços acústicos e visuais, tendo a impressão de estarem todos em um único ambiente.

Os serviços de teleconferência são classificados como:

- Áudio Conferência - sinais de áudio e controle são transmitidos entre os participantes;
- Conferência Áudio-Documentária – parecido com áudio conferência, mas com o tratamento de documentos textuais;
- Conferência Audiográfica - suporte a transmissão de áudio, sinais de controle, documentos textuais e imagens estáticas;
- *Freeze-Frame* Videoconferência - parecido com conferência audiográfica acrescida do envio periódico de imagens estáticas dos participantes;
- Teleseminário - consiste da distribuição dos eventos ocorridos num local (áudio e vídeo) para todos os demais participantes, sendo o áudio o único sinal de retorno;
- Videoconferência – parecida com audiográfica acrescida do envio, em tempo real, de sinais de vídeo entre os vários participantes.

Uma videoconferência pode ser realizada de forma *Ponto-a-Ponto* ou *Multicast*. Na forma *Ponto-a-Ponto*, os computadores que participam da videoconferência conectam-se diretamente através do número do endereço de rede IP. O tipo *Multicast* permite transmitir pacotes a mais de um endereço ao mesmo tempo. Neste tipo uma mensagem é enviada com o endereço de um grupo e é recebida por todos os membros deste grupo, desde que estes estejam acessíveis pela rede *MBONE* [8].

Em um sistema de videoconferência pode-se ter uma unidade central chamada MCU (*Multipoint Control Unit*). Quando ela estiver presente (figura 7), o processamento do fluxo de áudio/vídeo é realizado de forma centralizada, ou em um conjunto de computadores; caso contrário este processamento é realizado na própria máquina de cada cliente da videoconferência.

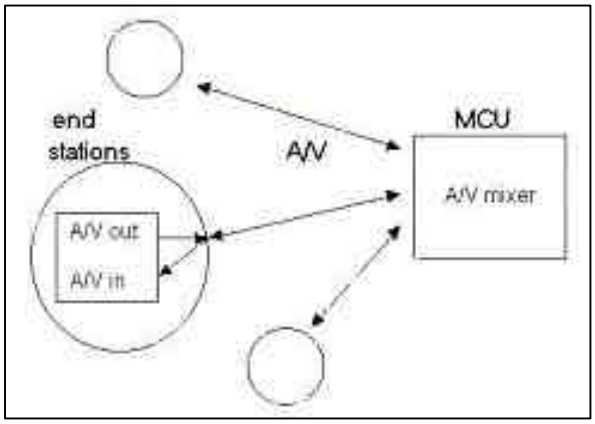

Figura 7: Videoconferência com o uso de MCU ("Multipoint Control Unit")

O MCU limita a escalabilidade do sistema, pois os sistemas de videoconferência são projetados para cenários específicos e não são facilmente adaptáveis para uso em outros cenários. Cada MCU suporta um número específico de participantes, além de gerar tráfego, pois o fato de utilizar conexões unicast faz com que todo o tráfego seja roteado pelo MCU, podendo provocar um congestionamento na rede.

Com a chegada da *Internet*, e distribuição de serviços, pensou-se em usar a comutação de pacotes, o qual os usuários se responsabilizam pelo tratamento dos dados, descentralizando o serviço (extinguindo o uso do MCU). Neste tipo de sistema a tolerância a falhas é maior do que na arquitetura centralizada, uma vez que não dependerá do perfeito funcionamento do servidor (MCU). Os usuários se conectam em uma estrutura *multicast* que os interconectam, permitindo a troca de informações entre eles.

A maior desvantagem dos sistemas descentralizados baseados em redes de comutação de pacotes é que não há nenhuma facilidade para o sincronismo dos "*codecs*", pois não existe mais um "*clock* do sistema" transmitido pela linha. [9]

Outra dificuldade enfrentada em sistemas de comutação de pacotes é a garantia de qualidade de serviços. Ao tentar aproveitar ao máximo a largura de banda, vários usuários compartilham o mesmo meio, competindo por recursos. Em determinados momentos esta competição pode atingir os limites da capacidade do canal, degradando assim a qualidade de serviços oferecida pela rede, e influenciando no sincronismo do áudio e vídeo transmitidos em uma videoconferência, por exemplo.

2.3.1. Largura de banda e QOS

Em uma transmissão de vídeo podemos ter diferentes tipos de computadores interligados em diferentes redes, cada uma com sua largura de banda. Ao se tratar de diferentes computadores, é preciso analisar a capacidade dos recursos oferecidos para cada computador, como uma transmissão de vídeo em que um dos participantes possua um PDA. Neste caso. é bem provável que uma adaptação de conteúdo seja necessária, atendendo as especificações de *software* e largura de banda disponíveis em um PDA. por exemplo.

Os requisitos para largura de banda [8] em aplicações multimídias para obtenção de mídias com qualidades boas (sem interrupções, imagem bem nítida, etc), podem variar de 100 Kbps a 100 Mbps (figura 8). Assumindo que a largura de banda seja limitada, pode-se selecionar uma qualidade de vídeo inferior que trabalhe dentro da largura de banda disponível.

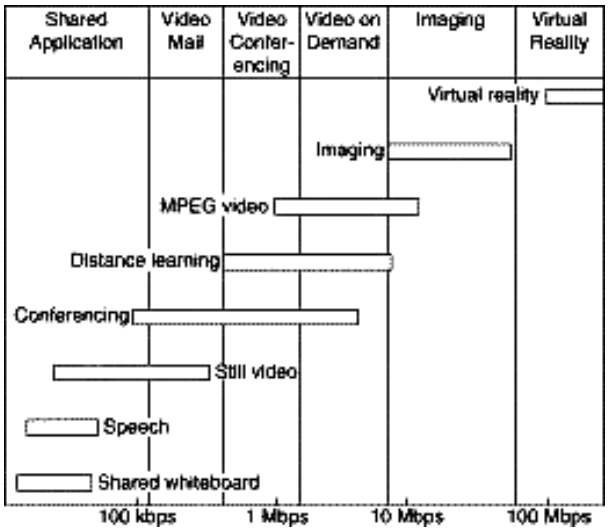

Figura 8: Largura de banda para aplicações multimídia

Diferente de outros tipos de serviços oferecidos (exemplo: FTP, SMTP...), dados multimídia geralmente são úteis somente se entregues dentro de um período de tempo específico. O atraso deles interfere no entendimento da informação a ser transmitida. A latência e o *jitter* (instabilidade) são os fenômenos principais que geram o atraso de dados de áudio e vídeo.

A latência ocorre em aplicações de tempo real (sensíveis ao atraso). A rede contribui para a latência de várias formas [8]:

- atraso de propagação: tempo que a informação leva para viajar a distância da linha;

- atraso de transmissão: tempo gasto para enviar o pacote multimídia;
- atraso no armazenamento para envio: tempo gasto para um dispositivo da rede enviar um pacote recebido (exemplo: um *switch*);
- atraso de processamento: tempo gasto pelo dispositivo da rede para inserir campos adicionais no pacote para garantir o roteamento correto (exemplo: cabeçalho da rede, chaves);

O *jitter* ocorre quando a rede entrega pacotes ou células com uma latência variável. Em transmissões de áudio, o *jitter* pode causar *pop* (interrupções no meio de uma palavra) e *clicks* (palavras sem interrupções entre uma e outra).

Para solucionar problemas de QoS, de sinalização e transmissões multimídia vários protocolos [10] foram definidos. Eles trabalham sobre a camada de transporte (figura 9), dentre eles se destacam:

- SIP: *Session Initiation Protocol*. Possui como objetivo principal, procurar, localizar e convidar usuários a participar de uma conexão. O SIP também é responsável pelo gerenciamento básico de uma chamada
- RTSP: *Real time Streaming Protocol*. Estabelece e controla fluxos contínuos de áudio e vídeo entre servidores e clientes. O servidor fornece os serviços de reprodução e gravação de dados enquanto um cliente solicita dados contínuos do servidor. Fornece as operações de recuperação de dados de um servidor, convite de servidor a uma videoconferência e inclusão de dados em uma apresentação existente.
- RSVP: *Resource Reservation Protocol*. É usado para estabelecer reserva de recursos na rede para uma aplicação que utilize conceitos de tempo real, sendo utilizado nos roteadores para distribuir qual QoS especificada pelos clientes, para todos os nós que fazem parte do caminho que a informação percorre entre a origem e o destino das informações.
- RTP: *Real Time Protocol.* Foi desenvolvido para prover serviços de envio e recebimento de dados com características específicas de tempo real (*unicast* ou *multicast*), como uma videoconferência, por exemplo.
- RTCP: *RTP Control Protocol*. Trabalha em conjunto com o RTP, o qual tem como função fornecer informações de controle sobre a QoS oferecida pelo meio de transporte e colher dados sobre os membros participantes de uma

sessão. É baseada na emissão periódica de pacotes de informação, no mesmo formato utilizado pelo RTP, para os nós da sessão em andamento.

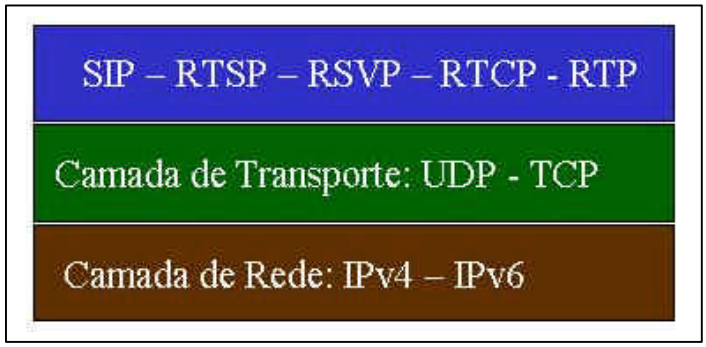

Figura 9: Protocolos de reserva de recursos, transporte, controle e streaming

Um outro fator a ser usado para manter uma QoS é o processo de adaptação de conteúdo . Ao usar este processo em tempo real [11] deve-se respeitar um determinado deadline, já que se espera que o cliente assista a um vídeo em um tempo mínimo de atraso.

2.3.2. Softwares de Videoconferência existentes

Existem atualmente muitos *softwares* de videoconferência no mercado. Eles foram difundidos principalmente depois da popularização da Internet. Nem todos seguem as recomendações do ITU-T [9], porém são utilizados com sucesso.

Muitos *softwares* de videoconferência optam por sistemas centralizados, com um servidor de conferência que recebe os dados dos diversos usuários, os processa e os retransmite para cada um dos outros participantes. Isto ocorre pela facilidade de implementação. Este tipo de utilização não se preocupa com a má utilização da rede, uma vez que diversos pacotes replicados são transmitidos, um para cada participante. Alguns *softwares* de videoconferência existentes são: *CU-SeeMe*, TVS e J-VCR.

2.3.2.1. CU-SeeMe

Desenvolvido por *Tim Dorcey* na Universidade de *Cornell*, EUA, o *CU-SeeME* foi um dos primeiros protótipos de videoconferência disponíveis pela Internet [3]. Trata-se de um sistema baseado na idéia de refletores. Quando um usuário deseja participar de uma conferência, ele deve se conectar a um refletor daquela conferência, informando o identificador da conferência desejada. Caso a conexão seja realizada entre dois participantes apenas, o uso do refletor é opcional. O *CU-SeeMe* possui implementação

para *Macintosh* (sua plataforma nativa) e *Windows*, devendo o refletor ser instalado em estações UNIX.

O *CU-SeeMe* utiliza codificações proprietárias. O vídeo transmitido pode ter a resolução de 320x240 *pixels* ou, mais comumente, 160 x 120 *pixels*. Os *pixels* são codificados em 16 tons de cinza, com quatro bits por amostra.

O algoritmo de codificação de vídeo possui três etapas bem definidas. A primeira é a etapa de redução da imagem, dos 640x480 *pixels* NTSC fornecidos pela câmera para a resolução adequada (metade ou quarta parte do NTSC), sendo a resolução de cores do *pixel* reduzida de 16 bits em cores para 4 bits em tons de cinza. A primeira etapa reduz um quadro na razão de 64:1. A segunda etapa consiste da divisão de um quadro em blocos de 8x8 *pixels* seguida de análise que garante a transmissão somente daqueles que possuírem suficientes modificações para o anterior. Cada bloco é transmitido periodicamente mesmo que não tenha sofrido alterações significativas, para evitar acumulação de distorção.

A terceira e última etapa consiste de um algoritmo de compactação que manipula cada linha de 8 *pixels* com 4 bits cada como uma única palavra de 32 bits, o que melhora a performance em arquiteturas de 32 bits. A compactação proporciona uma redução adicional da ordem de 40% no sinal de vídeo. O áudio é igualmente codificado por algoritmo proprietário, estando disponível apenas em algumas das plataformas.

Na sua versão comercial, o vídeo pode ser codificado em cores através de algoritmo desenvolvido por uma empresa denominada *Crystal Net Corporation*. Este algoritmo, não compatível com o ITU-T H.261, promete uma boa performance, embora não seja um padrão endossado por organismos internacionais.

O *CU-SeeMe* permite que mensagens sejam escritas sobre a janela de vídeo dos participantes e na sua versão mais recente, apresenta uma janela de conversação, onde os diversos usuários podem digitar mensagens textuais. Não há a existência de um coordenador da videoconferência.

Não existe um sistema de segurança. Não há suporte para votação. O *CU-SeeMe* oferece a facilidade de um "*whiteboard*" simplificado para tratamento de documentos, que permite a apresentação de uma imagem estática com resolução de 8 bits, em tons de cinza, como apoio às discussões em andamento.

Para a identificação do interlocutor no *CU-SeeMe* é preciso que o mesmo possua uma câmera de vídeo, pois ela só acontecerá para os demais participantes caso eles visualizem em suas telas a imagem provinda da câmera do interlocutor. Ou seja, se o

interlocutor não tiver uma câmera conectada em seu micro, os demais participantes não serão capazes de identificá-lo. A otimização da banda passante na rede pode ser configurada manualmente no *CU-SeeME*.

A última versão do *CU-SeeME* foi a 6.0 (figura 10). Após algumas adaptações, atualmente ele é conhecido como o *CU-World* (versão 6.0 Beta).

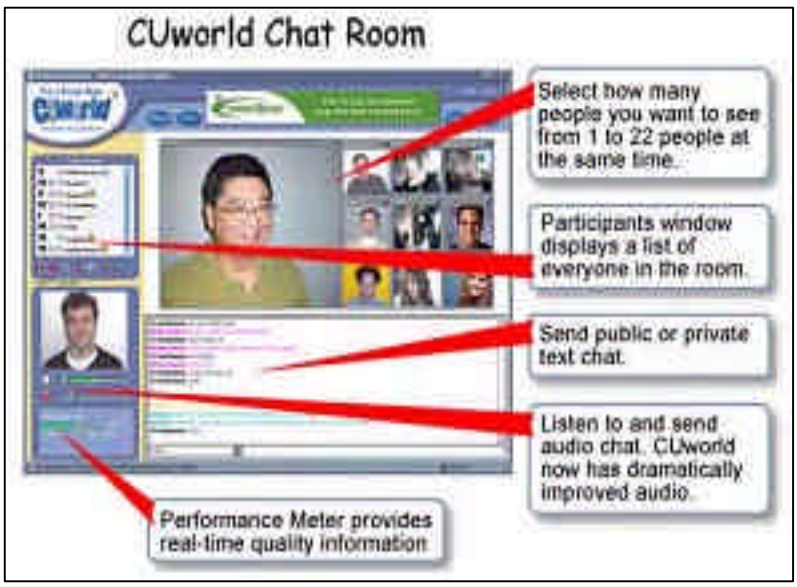

Figura 10: Software videoconferência Cu-SeeMe/CuWorld

2.3.2.2. TVS

O TVS [6] é um sistema do Laboratório TeleMídia, PUC-Rio. Desde sua especificação, o TVS procurou seguir ao máximo os padrões estabelecidos para sistemas de videoconferência, bem como para o intercâmbio de objetos multimídia/hipermídia.

O TVS utiliza a recomendação ITU-T H.261 para codificar os sinais de vídeo, como requisitado pela recomendação ITU-T F.730. O H.261 é um método de codificação de sinal de vídeo desenvolvido para aplicações em tempo real. O H.261 define dois formatos de imagem, o CIF e o QCIF, respectivamente com resolução de 352 *pixels* por linha e 288 linhas por imagem e 176 *pixels* por linha e 144 linhas por imagem.

A codificação de áudio utiliza a recomendação G.711 da ITU-T, como indicado na recomendação F.730. A recomendação G.701 define o PCM como um processo no qual um sinal é amostrado e cujas amostras são quantizadas de modo independente das outras e posteriormente convertidas para um sinal digital através de codificação. O PCM é um método de digitalização de um sinal, ou seja, transformação de um sinal analógico (a voz no nosso caso) em sinal digital.

O TVS provê suporte ao trabalho cooperativo através da função de manipulação cooperativa de documentos multimídia/hipermídia implementados na janela de base compartilhada (figura 11), que funciona como uma área visualizada por todos os participantes da conferência.

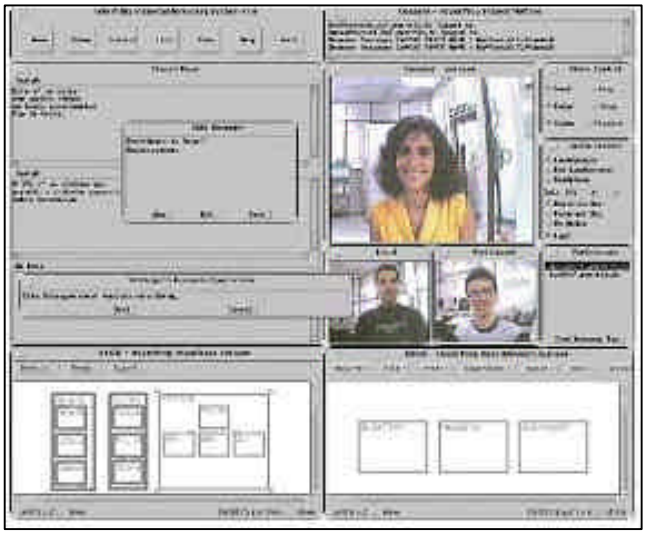

Figura 11: Software videoconferência TVS

O TVS provê suporte a votações. Em qualquer instante um participante pode criar uma votação, indicando o título e as opções da votação, selecionar uma votação previamente configurada e finalmente votar.

Em qualquer instante, um participante pode enviar uma mensagem textual para um outro participante ou grupo de participantes da conferência. Deste modo o sistema garante a troca de informações entre usuários, uma vez que não permite conversas paralelas. O organizador pode desabilitar esta função para um determinado usuário ou grupo de usuários, podendo também o coordenador habilitar/desabilitar esta função para cada usuário durante o desenrolar da conferência.

O TVS prevê uma etapa anterior à conferência propriamente dita, denominada pré-conferência. Nessa etapa, o organizador agenda e configura o ambiente da conferência. Nesse agendamento, se disponibiliza o assunto principal a ser tratado, a data da conferência, e quais participantes poderão fazer parte da videoconferência.

O sincronismo entre as mídias de áudio e vídeo é implementado através do uso de *time-stamps*, que consiste em marcar o sinal de áudio e vídeo com valores que indicam a relação entre as mídias. Estes valores são utilizados pela estação destino para o consumo simultâneo das mídias com a mesma marca.

O mecanismo de controle de acesso ao ambiente TVS é realizado através de detecção de silêncio, em que cada estação executa o algoritmo de detecção de silêncio.

O TVS é o único protótipo a apresentar as características de suporte à votação, controle de acesso e existência de um coordenador.

2.3.2.3. J-VCR – *Java* Vídeo *Conference Recorder*

Desenvolvido por *Shervin Shirmohammadi, Li Ding*, e *Nicolas Georganas* no *Multimedia Communications Research Laboratory* da Faculdade de Informação e Engenharia da Universidade de Ottawa, Ottawa, Canadá [HRef-3].

O J-VCR é conseqüência do sistema JETS (*Java-Enabled Telecollaboration System*) que também foi desenvolvido na Universidade de Ottawa. O JETS com o uso de *Java applets* permite múltiplos usuários em uma sessão de telecolaboração. Existem várias razões para seu uso: não depende de plataforma (usuários com diferentes plataformas, Windows, Sun, IBM, etc podem usá-los); não precisa fazer *download* do *software* e nem instalar na máquina para que funcione. Basta ter um navegador com o *plugin* do java instalado nele.

O J-VCR usa o RTP (*Real Time Protocol*) sobre o MBONE, para transmitir *streams* de áudio e vídeo. Com isso muitos participantes podem receber estas *streams* se estiverem inseridas no grupo *multicast*. Já o JETS comunica através de envio de pacotes por broadcast enviados a partir do servidor JETS.

O J-VCR permite que seja gravada a sessão em um formato de dado independente para ser visualizado por um sistema posteriormente. Esta gravação é armazenada em uma base de dados podendo ser visualizada de modo assíncrono.

A captura do vídeo é feita com funções do JETS e uma associação real-time de *streams* de vídeo e áudio. O J-VCR combina como uma sessão especial do JETS. Diferente de outros participantes, o cliente J-VCR apenas recebe dados, enquanto os outros clientes enviam e recebem dados.

O JVC-R utiliza o *Java* Media *Framework* (JMF) para capturar, manipular e armazenar dados de áudio do vídeo. JMF é uma extensão da tecnologia *Java* capaz de mostrar e capturar dados multimídia através de aplicações *Java* e *applets*.

2.4. Computação Ubíqua

O avanço da tecnologia vem fazendo com que o tamanho e usabilidade dos computadores diversifiquem constantemente. Inicialmente, há 3 décadas aproximadamente, se falava de grandes computadores centrais, poderosíssimos, conhecidos como *mainframes* (“Era dos Mainframes”). Os usuários dos *mainframes* tinham que compartilhar esta única máquina para executar diferentes tarefas. Logo depois (década de 80) se falava dos computadores menores e pessoais, em que cada pessoa possuía seu próprio computador (“Era dos computadores pessoais”).

Com a chegada da computação distribuída e da Internet [12], tem-se o modelo cliente-servidor, o qual além dos recursos oferecidos por um computador pessoal, pode-se usufruir recursos oferecidos por outros computadores interligados (“Era da Internet e Computação Distribuída”). Abriu-se então a possibilidade dos computadores estarem presentes em diversos objetos, como carros, telefones, etc. (“Era da Computação Ubíqua” - *ubicomp*).

2.4.1. Conceitos básicos de Computação Ubíqua

Mark Weiser definiu como característica da computação ubíqua uma intercomunicação entre os computadores presentes em vários objetos (PDAs, PCs, laptops, celulares, geladeiras, etc.).

A propagação dos computadores [13] sugere além da disponibilidade de infra-estrutura computacional, um novo paradigma inspirado pelo acesso constante à informação e às capacidades computacionais.

A computação ubíqua insere um novo conceito de percepção de mundo: dividido em periferia e centro. O termo “periferia” diz respeito àquilo que está à nossa volta, mas que não nos prende a atenção. Já o termo “centro” representa todos os objetos que concentram nosso foco de atenção. Para esta nova abordagem que integra estas novas tecnologias ao nosso cotidiano foi dado o nome de *calm technology* [12]. Por exemplo, ao assistir uma palestra, nossa atenção está concentrada ao palestrante; neste caso o cenário é a “periferia” e o palestrante o “centro”.

A computação ubíqua veio para ajudar a organizar e interferir nas interações sociais em qualquer lugar e em qualquer tempo que essas situações possam ocorrer. Durante os próximos 5 a 10 anos a computação ubíqua [14] pregará um desafio para o desenvolvimento de serviços relacionados, que mudará o conceito básico que existe

sobre infra-estrutura de integrações de computadores além de mostrar a evolução que as aplicações sofrerão para se adaptar a esta nova era.

A computação ubíqua pode ser vista [14] como uma junção da computação móvel e da computação permeável (*Pervasive*). São definidos quatro tipos de computação: Computação Tradicional, Computação *Pervasive*, Computação Móvel e Computação Ubíqua. A computação tradicional possui pouco nível de transparência e mobilidade. Já a computação *Pervasive* possui um alto nível de transparência, sendo capaz de adaptar funcionalidades dinamicamente, porém com pouca mobilidade, não se adaptando dinamicamente com as variações do contexto da aplicação (mudanças de ambiente - fisicamente). Na computação móvel a capacidade de mobilidade é alta, porém o nível de transparência é baixo. (Figura 12)

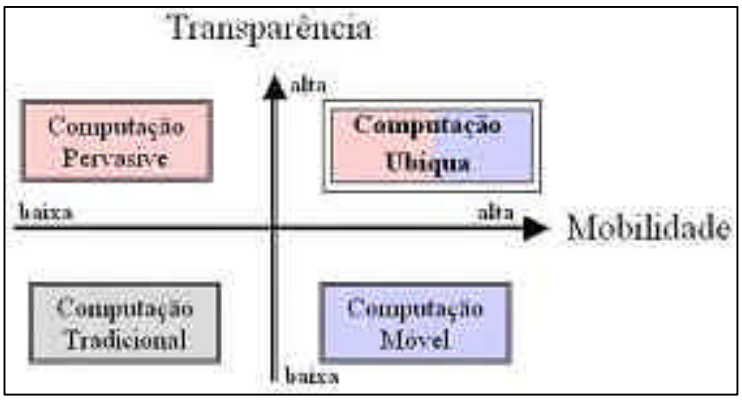

Figura 12: Grau de transparência e mobilidade na Computação Tradicional - Pervasive – Móvel e Ubíqua

Exemplo de aplicações que se relacionam com a computação *Pervasive*, são programas desenvolvidos sobre a tecnologia JINI (tópico 2.5.1.4 deste trabalho) e *middlewares* que promovem uma interligação entre diferentes equipamentos.

2.4.2. Aplicações de Computação Ubíqua

As aplicações resultantes das pesquisas na área de computação ubíqua [13] abordam três desafios: “Interfaces Naturais”, “Captura Automatizada” e “Consciência de Contexto”.

2.4.1.1 Interfaces Naturais

O objetivo das interfaces naturais é suportar formas comuns de expressão humana. Esforços anteriores se focaram em interfaces de reconhecimento de voz e escrita com uma caneta eletrônica, mas estas interfaces ainda não lidam com eficácia com os erros

que ocorrem naturalmente com estes sistemas. Além disso, estas interfaces são muito difíceis de serem implementadas.

Ao se tratar de interfaces naturais, a computação ubíqua inspira um certo deslocamento de metáfora do computador comum (teclado, *mouse*, monitor) para uma forma de interação mais parecida com o modo que os humanos interagem com o mundo físico. Humanos falam, gesticulam e escrevem para se comunicar, e isso poderia ser usado implícita ou explicitamente como entradas para sistemas de computação ubíqua.

2.4.1.2. Captura Automatizada

Na captura automatizada, o conteúdo referente às nossas experiências do dia a dia pode ser capturado de modo a suportar a geração automática de documentos hipermídia que podem ser armazenados, recuperados e apresentados ao longo do tempo. Aplicações que automatizam este processo de captura e acesso nos permite concentrar a atenção na experiência em si, sem que precisemos nos preocupar com a tarefa de registrar informação.

Uma grande parte da nossa vida é gasta escutando e gravando mais ou menos precisamente os eventos que estão a nossa volta e depois tentando lembrar partes importantes das informações adquiridas nesses eventos. Há um claro valor e um perigo em potencial [15] em usar recursos computacionais para ajudar na "falta de memória" dos seres humanos, especialmente quando há múltiplas seqüências de informação relacionada que são virtualmente impossíveis de serem absorvidas como um todo. Ferramentas que suportem gravação e acesso automático a experiências reais podem remover o fardo de fazer alguma coisa que os humanos não são bons de modo que possamos focar a atenção em atividades que nós somos bons.

2.4.1.3. Consciência De Contexto

A definição de contexto [16] é o de uma informação qualquer que possa ser usada para representar a situação de uma entidade (um lugar, pessoa ou objeto considerado importante para interação entre um usuário e uma aplicação).

Uma aplicação que mude dinamicamente suas funções de acordo com o contexto que ela se enquadra é um exemplo de contexto. Exemplo disso é uma aplicação *Java* fornecida na Internet que detecte que seu cliente não possua a máquina virtual *Java* instalada; diante disto, ela deixaria de usar seus recursos *Java*, e utilizaria outros recursos HTML para suprir as necessidades do cliente.

Este projeto de mestrado trabalhou com percepção de mudança de contexto localista por parte do usuário do sistema, permitindo mobilidade do usuário baseado no conhecimento da localização atual. Para a detecção de mudança de contexto localista, no estudo de caso realizado, foram utilizados sensores [17] representados por *webcams* convencionais.

A função de detecção de mudança de contexto de localização pode ocorrer de diversificadas maneiras. Por exemplo, pode ser usado recursos infravermelhos ou a tecnologia GPS para detectar a localização de determinada entidade. No caso da arquitetura definida nesta monografia, a detecção de mudança de contexto localista agirá de forma com que o usuário continue assistindo ao vídeo que estava assistindo em um outro local (sala, quarto, etc).

Uma outra definição para contexto [13] é o entendimento dos "cinco W's" (*Who, What, Where, When* e *Why*):

- ***Who* (Quem?):** A aplicação sempre leva em consideração a pessoa que esta a utilizando, não importando as características de outras pessoas.
- ***What* (O que?):** O sistema deve interpretar "o que" o usuário está fazendo naquele momento. Interpretar ações humanas é uma tarefa complicada, porém um dispositivo direcionado ao contexto provavelmente precisará incorporar interpretação de atividades humanas para fornecer informações úteis.
- ***Where* (Onde?):** Fornece um histórico dos movimentos do mundo físico capaz de se adaptar, mostrando informações baseadas no caminho visível de interesse do usuário.
- ***When* (Quando?):** Fornece entendimento relativo às mudanças com o passar do tempo. Por exemplo, em uma casa com consciência de contexto este componente pode identificar uma alteração na rotina de um morador da casa em um determinado horário, comparando com as ações realizadas neste mesmo horário em dias anteriores.
- ***Why* (Porque?):** Deve fornecer o por quê do usuário está realizando determinada ação. É preciso detectar outras formas de informações de contexto, como por exemplo, batimento cardíaco, temperatura do corpo, para determinar realmente o por quê daquele contexto. Para saber este "por que" deve saber o que (*what*) o usuário estava fazendo e onde (where) o usuário se localiza.

2.4.3. Protótipos de *hardware*

Segundo *Weiser*, os *hardwares* relacionados com computação ubíqua possuem diretrizes opostas dos *hardwares* voltados à realidade virtual. A realidade virtual coloca a pessoa dentro do mundo da computação, enquanto a computação ubíqua força o computador a viver no mundo das pessoas.

Os primeiros equipamentos relacionados com computação ubíqua surgiram [HRef-8] na forma de “tabs”, “pads”, and “boards” construídos pela empresa XEROX PARC.

2.4.2.1. Tab

São dispositivos pequenos e portáteis (figura 13) com uma entrada de informação. Possui tela sensível ao toque, e conectividade constante. Utiliza o infravermelho como tecnologia de comunicação sem fio, com velocidades entre 9600 e 19200 bps.

Figura 13: Exemplo de TAB

2.4.2.2. Board

Telão sensível ao toque (figura 14), grava os dados que são escritos através de uma caneta eletrônica. São conhecidos como *whiteboards*. Podem ser representados por um *software* o qual o usuário desenha com o próprio mouse em uma janela aberta em seu computador.

Figura 14: Exemplo de Board (sala de aula)

2.4.2.3. Pc-Prancheta

É um dispositivo criado pela Intel (figura 15) capaz de controlar todos os eletrodomésticos de uma casa compatíveis com tal dispositivo (ex: geladeira, forno, microondas, *home theater*, etc).

Figura 15: Exemplo de PC-prancheta da Intel que controla todos os eletrodomésticos

2.4.2.4. *Wearable Computing* – "Olhos Alheios"

Tecnologia da Motorola que é formado por um colar com microcâmera e brincos-gravadores (figura 16) através dos quais o usuário pode estabelecer uma conexão sem fio com familiares, colegas ou amigos, transmitindo em tempo real mensagens de voz, dados e imagens.

Figura 16: Exemplo de wearable computing – "olhos alheios"

2.4.2.5. *Wearable Computing* – "Mãos Que Escutam"

Tecnologia da Motorola (figura 17) que armazena vídeos e sons para editar, assistir ou ouvir quando se voltar para casa ou para o escritório. Caso o usuário prefira, poderá enviar as imagens em tempo real para seus interlocutores e visualizá-los em um mesmo instante no *display* do aparelho que pode ser dividido em várias telas.

Figura 17: Exemplo de wearable computing – "mãos que escutam"

2.4.2.6. *Active Badge*

Active Badge (*figura 18*) é o dispositivo que transmite e recebe os sinais infravermelhos que permitem ao portador mover-se dentro da área controlada pelo *Active Badge System* (com uso de sensores – recebem informações). O *Active Badge* notifica o sistema da sua posição a cada 10 segundos e recebe sinais do sistema. Tem ainda dois botões que enviam sinais que podem ser programados para algum tipo de funcionalidade adicional.

Figura 18: Active badge

2.4.4. Exemplo de aplicação

Foi visto que o conteúdo referente às nossas experiências do dia a dia pode ser capturado de modo a suportar a geração automática de documentos hipermídia que podem ser armazenados, recuperados e apresentados ao longo do tempo. Neste mesmo contexto, várias aplicações podem ser citadas, como o projeto da *Aware Home*, *E-Class*, navegação/alarme em um automóvel com o uso de GPS, alarmes com envio de SMS, etc.

2.4.3.1. Aware Home

A AHRI ("*Aware Home Research Initiative*") é uma pesquisa [HRef-9] interdisciplinar com o esforço da *Georgia Tech*. Fruto desta pesquisa foi a construção de uma casa (*Aware Home*), que com o uso das ferramentas desenvolvidas para a mesma, pode ser utilizada para ajudar as pessoas no seu dia a dia, ou mesmo contribuir com o avanço da medicina. Esta casa foi construída para servir de laboratório para aplicações de computação ubíqua.

A *Aware Home* possuía no ano de 2002, dois quartos, dois banheiros, um escritório, uma cozinha, uma sala de jantar, uma sala de estar, uma lavanderia, além de uma sala específica onde estarão centralizados os computadores. Várias câmeras foram instaladas, além de usar um sistema de radiofreqüência capaz de informar com precisão, para onde as pessoas estão se movendo.

A *Aware Home* usa tecnologias capazes de interferir/auxiliar muito o dia a dia do ser humano. Como exemplo, um porta-retrato (figura 19) que mostra imagens dinamicamente referentes a fotos tiradas no dia a dia na casa, um pingente capaz de capturar o tremor das mãos de um integrante da casa (detectando por exemplo uma doença) ou mesmo capaz de enviar uma mensagem para ligar/desligar as luzes da casa, um piso ciente de contexto responsável por capturar os passos de uma pessoa dentro da casa (detectando a forma de andar de cada pessoa).

Figura 19: Porta retrato com quadro digital que muda diariamente

Foi introduzido também um sistema de consciência de contexto que captura as informações que possam ser úteis no futuro. Esta funcionalidade é utilizada caso uma pessoa esqueça de alguma tarefa que tenha realizado, obtendo assim, uma ajuda para lembrar do ato realizado.

2.4.3.2. E-Class Ou Classroom2000

O *E-Class* ou *ClassRomm*2000 [13] é um projeto que vem sendo desenvolvido desde 1995 na *Georgia Tech*, em Atlanta/EUA, e no ICMC/USP desde 1998. Ele é um projeto de uma sala de aula que provê a captura das interações que ocorrem em uma aula típica dentro de uma universidade, influenciando nas diretrizes do Ensino a Distância (EAD).

É um ambiente pouco intrusivo (*figura 20*) e conta com projetores, *whiteboards*, microfones, câmeras de vídeo e computadores ligados em rede, visando uma captura detalhada das várias mídias existentes em uma apresentação.

As anotações realizadas no *whiteboard* são capturadas e armazenadas por uma *applet* escrita em *Java*, o *ZenPad*. As anotações, vídeos e áudios capturados podem ser visualizados posteriormente através da Internet.

Neste projeto ocorre a captura da informação na lousa eletrônica, de todo o ambiente com uso de câmeras de vídeo, do áudio através de microfones instalados no teto, além de gerar materiais a serem disponibilizados na Internet. Com isso o aluno poderá visualizar o conteúdo de uma aula gravada retrocedendo ou avançando o vídeo gerado.

Figura 20: ClassRoom2000

2.4.3.3. Projeto Gaia

O projeto Gaia [HRef-16] é um projeto que vem sendo desenvolvido por um grupo de pesquisadores da Universidade de Illinois, nos Estados Unidos, e busca trazer um ambiente que relaciona objetos virtuais com físicos. Destinado à construção de aplicações genéricas que não se relacionam diretamente a um *hardware* específico.

A arquitetura definida pelo projeto Gaia possui um *framework* de aplicação. Ele adapta o modelo de aplicações tradicional voltada para dispositivos computacionais que trabalham com *display*, mouse e teclado. Para esse modelo de aplicações tradicional dá-se o nome *Model-View-Controller* (MVC), ou seja, Modelo-Visão-Controlador. Ele possui um componente "Visão" destinado a visualização dos dados da aplicação em um *display*. A adaptação resulta em um modelo de computação definido por "Espaços Ativos". São construídas aplicações de um modo genérico, usando descrições abstratas dos componentes exigidos que compõem a aplicação.

O *framework* de aplicação está baseado em um modelo chamado *Model-Presentation-Controller-Coordinator* (MPCC). Neste modelo tem-se um controlador e coordenador. Ele é semelhante conceitualmente com o modelo MVC, embora renomeie

o componente de “Visão” a “Apresentação”, possuindo um componente adicional chamado “Coordenador”.

O modelo implementa a lógica de aplicação, a apresentação exporta os dados do modelo, o controlador introduz mudanças no modelo (o modelo notifica todas as apresentações automaticamente), e o coordenador é responsável por administrar a composição dos três componentes prévios (figura 21).

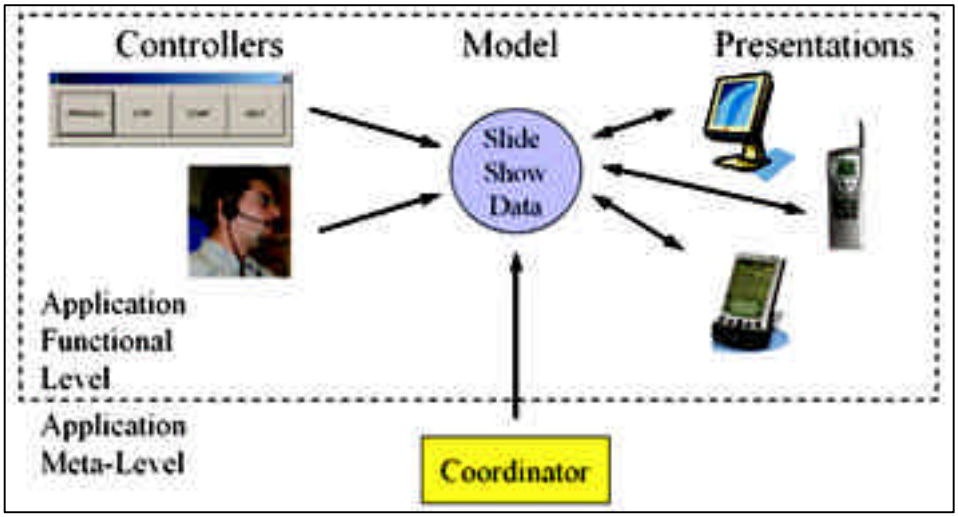

Figura 21: Model-Presentation-Controller-Coordinator Application

2.4.3.4. Outras Aplicações – Sensíveis ao Contexto

A navegação e alarme em um automóvel utilizando a tecnologia GPS são exemplos de aplicações sensíveis ao contexto. GPS(*Global Positioning System*) é uma tecnologia desenvolvida na década de 90 que oferece um sistema que consiste em uma unidade de localização emissora e receptora de sinais de radiofreqüência, e utiliza recursos de satélites.

Equipamentos com recursos GPS (figura 22) atualmente não são mais novidades em automóveis. Eles indicam precisamente em que localidade o automóvel se encontra. Eles podem possuir um detalhado banco de dados de mapas para navegação e são capazes de instruir o motorista para qual direção ele deve ir para encontrar um determinado endereço/local.

Em caso de roubo, o motorista poderá telefonar para uma central, que emite um sinal de radiofreqüência para localizar o carro. Este, por sua vez, devolve o aviso para as antenas da central indicando sua posição que pode chegar a uma precisão de 15 metros e em tempo real. Com isso a central pode tomar decisões implícitas ao usuário, como

mandar viaturas da polícia para o local, ou mesmo um helicóptero para uma melhor visualização e posicionamento.

Figura 22: Exemplo de equipamento que usa GPS para se localizar

Um outro exemplo são os alarmes residenciais ou mesmo de automóveis, em caso de arrombamento ou uma invasão, que pode ser detectada por infravermelho, pode enviar implicitamente uma mensagem via SMS para o proprietário da casa/automóvel. Com isso, independente do local onde o proprietário estiver, ele será avisado que a sua casa/automóvel foi arrombado, ou mesmo que o alarme foi acionado, podendo ter conhecimento, por exemplo, qual porta/janela foi arrombada. Lembrando que a maioria dos alarmes residenciais, atualmente, já comunicam imediatamente a polícia quando o mesmo é acionado através de recursos da telefonia comum.

A televisão interativa também é um exemplo de aplicação sensível ao contexto. Você está assistindo ao seu programa de TV favorito, quando num determinado momento da programação surge um sinal que você pode interagir com a informação que está sendo veiculada naquele exato instante. Um símbolo "I" é um exemplo de link interativo, posicionado no canto superior direito da TV. Trata-se de um sinal trazido pelo sinal convencional da emissora, através de uma codificação especial.

A *LabOne* (uma empresa de mídia digital, especializada no desenvolvimento de tecnologia, produção, gerenciamento e distribuição de conteúdo interativo), fornece às emissoras de TV e produtores independentes o serviço de *links* interativos, tornando sua programação mais inteligente e potencializando ações de comércio eletrônico e interação, como se fosse na Internet convencional.

2.5. Tecnologias relacionadas com o projeto

Os dispositivos ubíquos como por exemplo mini-laptops, PDAs, telefonia sobre IP, telefones celulares e aplicações para casa, nos proporcionam uma boa ajuda nas várias

tarefas de nosso dia a dia. A integração destes dispositivos na rede para sistemas heterogêneos e móveis vem causando um certo problema, pois estes dispositivos podem possuir *softwares* e arquiteturas diferenciadas uns dos outros. Eles podem possuir uma limitação de espaço em memória ou mesmo de outros recursos. [1].

Para a transmissão de imagens e vídeos entre equipamentos embarcados pode-se usar a tecnologia *Java* J2ME (*Java* 2 *Plataform MicroEdition*). O J2ME é um ambiente para desenvolvimentos de aplicações voltados a dispositivos móveis. Além desta tecnologia é importante conhecer quais tecnologias de rede sem fio poderiam ser utilizadas para esta transmissão destes vídeos/imagens, bem como quais outras tecnologias já existentes podem auxiliar para o alcance do objetivo desta monografia.

Será apresentado neste tópico então, um estudo sobre sensores, tecnologia *Java*, implementações relacionadas com o projeto existente e a JAMP, e tecnologias relacionadas com dispositivos móveis (padrões de rede e serviços oferecidos), além de um estudo dos tipos mais usados de redes sem fio (*Wireless)*.

2.5.1. Sensores

São dispositivos que mudam seu comportamento sob a ação de uma grandeza física (calórico, luminoso, sonoro, pressional, magnético, motor), podendo fornecer diretamente ou indiretamente um sinal que indica esta grandeza, transmitindo um impulso (mensurável ou operante) correspondente.

Existem atualmente sensores destinados para as mais diversificadas aplicações, como por exemplo *sensor de temperatura*, que podem ser empregados em caldeiras industriais; *sensor de velocidade*, presentes nos controles e medidores de velocidade de motores dentro de máquinas industriais, eletrodomésticos como videocassete e CD, unidades de disquetes e Winchesters de computadores; *sensores de vazão*, que medem o fluxo de líquidos em tubulações; *sensor de posição*, usados em aplicações em que se necessita monitorar a posição de uma peça, como tornos automáticos industriais, ou contagem de produtos, ou verificar a posição de um braço de um robô; *sensor de luz*, bastante utilizados em iluminação pública e alarmes residenciais; e *sensor rádio-freqüência(RF)*, utilizados principalmente para detecção de presença.

Os sensores de luz (sensores fotoelétricos) são usados em grande escala em aplicações para detecção de movimento. Estes sensores operam percebendo uma

mudança na quantidade de luz que é refletida ou bloqueada por um objeto (alvo) a ser detectado. Eles podem ser usados para detectar alvos de 5mm à 250m. [HRef-17]

Os dispositivos fotoelétricos para detecção de movimento utilizam um emissor (gerador) e um receptor (fotossensor) de luz. Eles podem ou não estar em um mesmo objeto. O princípio de seu funcionamento é a capacidade do fotossensor, que é um componente sólido, gerar uma mudança na corrente conduzida conforme a quantidade de luz detectada.

Pode-se usar também sensores eletromagnéticos (radiofreqüências) para detectar movimento (presença). A detecção de movimentos por radiofreqüência é uma técnica baseada no princípio de funcionamento do radar. Uma onda eletromagnética é enviada no espaço e ao encontrar um alvo é refletida e captada por um sensor. A reflexão dessa onda faz surgir um fenômeno conhecido por *Efeito Doppler* que permite ser interpretado e processado. [HRef-18]

2.5.2. Tecnologia *Java*: Introdução, J2ME e Midlets

A tecnologia *Java* reúne uma nova linguagem de programação e uma plataforma de desenvolvimento e execução de programas, resultados de um consistente processo de pesquisa. Foi desenvolvida para proporcionar uma fácil integração com outras tecnologias. O interpretador da linguagem é chamado máquina virtual *Java*.

Antes de um programa em *Java* ser interpretado, ele deve ser compilado para uma linguagem intermediária, chamada *Java Bytecodes*. Uma vez gerado, um arquivo *Bytecodes* pode ser executado sobre a plataforma *Java* instalada na maioria dos ambientes (sistema operacional), como *Windows, Solaris* ou *Linux*. Uma aplicação desenvolvida em *Java* pode estar presente na maioria dos *hardwares* existentes, por exemplo, em *smartcards*, celulares, PDAs, etc.

A plataforma *Java* inclui uma máquina virtual *Java* e a *Java* API. A *Java* API é um conjunto de bibliotecas de classes e interfaces padrão da linguagem. Os programas escritos em *Java* são executados sobre este ambiente.

A Plataforma *Java* 2 *Enterprise Edition* (J2EE) pode ser visto como um conjunto de especificações coordenadas e um guia de práticas que juntos permitem o desenvolvimento, instalação, execução e gerenciamento de aplicações n-camadas no servidor (figura 23). EJB (*Enterprise Java Beans*) é uma das muitas especificações contidas no J2EE. EJBs são a forma de se desenvolver a lógica de negócios e a camada de persistência de uma aplicação J2EE.

A Plataforma *Java* 2 *Standart Edition* (J2SE) define uma tecnologia de aplicações destinadas a computadores comuns (aplicações para PCs, notebooks).

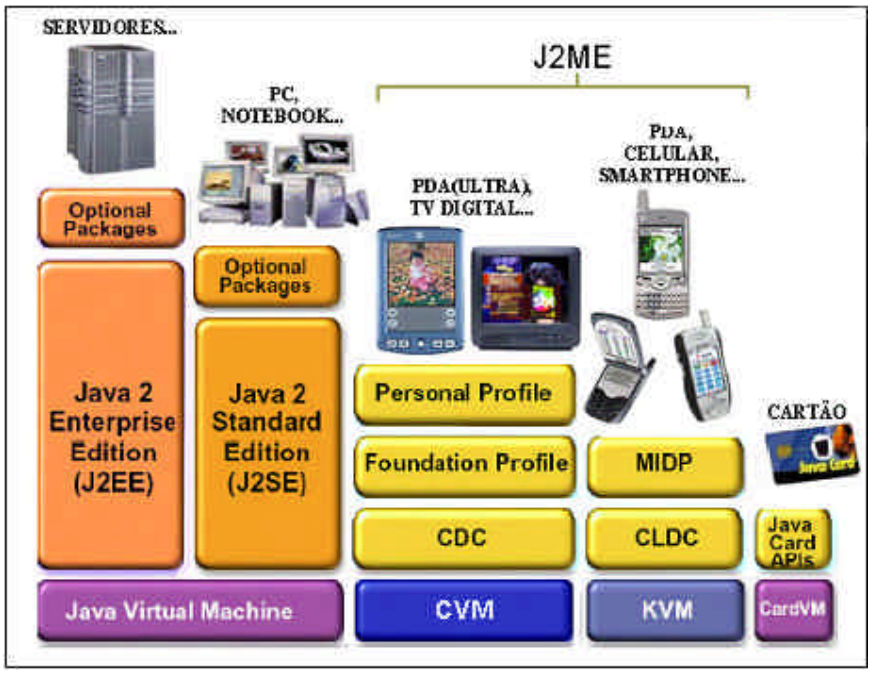

Figura 23: Edições Java e Plataforma J2ME – Micro Edition

2.5.2.1. J2ME - *Java* 2 *Micro Edition*

O J2ME (*Java* 2 *Micro Edition*) é uma tecnologia *Java* direcionada ao mercado de pequenos dispositivos. É um conjunto de especificações que tem por objetivo disponibilizar uma máquina virtual *Java*, API e ferramentas para equipamentos como: telefones celulares, *pagers*, PDAs, videogames, sistemas embutidos, cartões inteligentes, etc.

A tecnologia J2ME trabalha com conceito de *Configuração* e *Perfil*. *Configuração* especifica as características e um conjunto mínimo de bibliotecas *Java* para uma máquina virtual em particular. Ela define um conjunto mínimo de bibliotecas que a máquina virtual *Java* projetada especialmente para dispositivos com telas, memórias e processamento limitados deve saber interpretar.

Duas *Configurações* são definidas: CDC [HRef-10] (*Connected Device Configuration*) e CLDC [HRef-11] (*Connected Limited Device Configuration*). A configuração CDC destina-se principalmente a dispositivos com uma capacidade de memória superior à 512Kb. Já a CLDC é uma *Configuração* voltada para dispositivos com capacidade de processamento inferior, e memória entre 160Kb e 512Kb.

A máquina virtual definida pela *Configuração* CDC é conhecida como [18] CVM. Para a instalação completa da CVM em um dispositivo é necessário um espaço mínimo de memória de 2Mb. Os dispositivos alvos para esta plataforma são PDAs avançados, sistemas integrados nos automóveis, etc.

A *Configuração* CLDC possui como máquina virtual a KVM, que pode ser instalada em grande parte dos dispositivos embarcados, por exigir um pequeno espaço de memória para sua instalação. Como exemplo a estes dispositivos temos os telefones celulares, *smartphones* (telefones celulares acoplado com funcionalidades de PDA) e PDAs.

Os requisitos de *hardware* [19] necessários para que aparelhos celulares trabalhem com o CLDC são: mínimo de 160KB de memória para *Java*, um processador de no mínimo 16 bits com baixo consumo (adequado a baterias típicas de um celular) e conexão de rede (neste caso wireless – 9.6Kbps, 144Kbps ou 2Mbps). Já os requisitos de *software* são: *software* que inclua suporte a um subconjunto da linguagem *Java* e a um subconjunto da máquina virtual *Java* que definam um núcleo de funções que permitam o desenvolvimento de aplicações móveis.

Devido à grande quantidade de dispositivos existentes, variando desde placas com Sistema Operacional embutido até PDAs com grande poder de processamento, e funcionalidades diferentes, com conexões de redes velozes ou não, foi necessário definir um mínimo de funcionalidades (API e JVM) para uma faixa de equipamentos que tivessem características semelhantes (memória, processamento, conectividade). Logo, uma mesma *Configuração* atende vários dispositivos diferentes, não possuindo detalhes específicos do dispositivo.

Dependendo do dispositivo, uma especialização da JVM e API para determinados dispositivos se torna necessária. Define-se então um *Perfil* (figura 24) para este dispositivo. Alguns dos *Perfis* existentes são: MIDP, RMI *Profile*, *Foundation Profile*, PDA *Profile*, *PersonalJava*, etc.

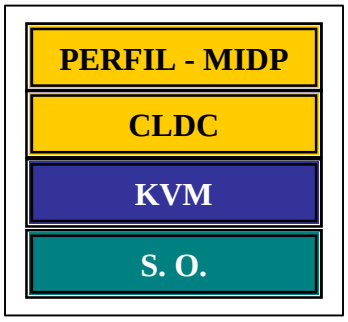

Figura 24: Exemplo de aplicação em uma arquitetura J2ME

O CLDC define em sua API [19] os pacotes *java.io*, *java.lang*, *java.util* e *javax.microedition.io* (conexão e interfaces). Não há suporte a ponto flutuante, suporte completo à classe "*Error*" do J2SE, a referências fracas, verificação de arquivos de classes (há no lugar uma ferramenta de "pré-verificação"), finalização - *Object.finalize*(), JNI (*Java Native Interface*), *reflection*, *thread groups/ daemons* e *user-defined class loaders*. O MIDP define tudo o que há no CLDC com a adição aos pacotes *javax.microedition.lcdui* (interface com o usuário), *javax.microedition.rms* (sistema de gerência de registros para persistência de informações), *javax.microedition.midlet* (suporte para aplicações MIDP, os chamados midlet s).

Algumas das ferramentas necessárias para se trabalhar com o J2ME são J2ME *Wireless Toolkit* e *PalmOS Emulator*. O J2ME *Wireless Toolkit*, vem com um emulador e vários *skins* que permite a escolha de qual o dispositivo (PDA/celular) a ser emulado, além de acompanhar vários exemplos com código fonte.

2.5.2.2. MIDlet

Um *MIDlet* é uma aplicação *Java* desenvolvida para ser executada em um dispositivo móvel, baseado na tecnologia J2ME, que estende a classe *MIDlet*. A classe *MIDlet* provê três métodos abstratos que são usados pela aplicação. Estes métodos são chamados a partir do gerenciador de aplicações do dispositivo. Ele é usado para comunicar com os aplicativos que estão em execução, e é responsável por instalar, executar e, se necessário, excluir uma aplicação de um dispositivo móvel. O método "*startApp*" é chamado (figura 25) imediatamente depois do construtor e cada vez que um aplicativo é ativado.

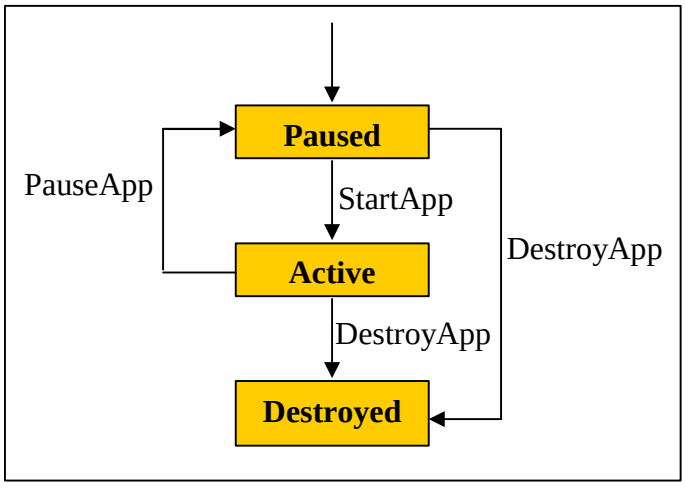

Figura 25: Métodos de uma aplicação midlet

O método "*destroyApp*" é chamado pelo gerenciador de aplicativos para indicar que uma aplicação está preste a terminar. Diferente do método "*startApp*", ele será chamado uma única vez durante o tempo de execução de um aplicativo. Como o MIDP não inclui em sua funcionalidade a capacidade de finalizar seus objetos, é aconselhado que o programador o faça neste método. Ao mesmo tempo, entretanto, um dispositivo móvel típico é bem menos estável que uma plataforma típica de desenvolvimento e será constantemente desligado e/ou reinicializado pelo usuário. Portanto, você não pode contar com a execução de "destroyApp" todas às vezes. [19]

O método "*pauseApp*" é utilizado para notificar uma “pausa” ao aplicativo que está executando. Isto pode acontecer quando o usuário iniciar um outro aplicativo ou utilizar uma função do dispositivo que inviabilizará a execução do seu aplicativo. Pelo fato da maioria dos dispositivos móveis não serem "*multitasking*"(executando várias funções ao mesmo tempo), este método provavelmente será chamado com uma certa freqüência. Neste método se pode liberar os recursos sendo utilizados pelo seu aplicativo, e, quando a aplicação volta a rodar, o método "startApp" será chamado pelo gerenciador de aplicativos, reiniciando a aplicação.

2.5.2.3. Ferramentas para desenvolvimento de aplicações J2ME

Para a programação de um *MIDlet* é preciso escolher primeiramente o ambiente de desenvolvimento. Atualmente existem vários ambientes (*toolkits*), dentre eles: J2ME Wireless Toolkit (versões: 1.0/2.0/2.1) – disponível na URL http://java.sun.com/products/j2mewtoolkit/ e Nokia Developer´s Suíte 2.0 for J2ME – disponível na URL http://www.forum.nokia.com/nds_for_j2me.html.

Após a codificação de uma aplicação J2ME, o *toolkit* é responsável por sua compilação, pré-verificação e empacotamento, para depois emular o resultado e depurar o código.

A aplicação compilada em um *toolkit* gera um arquivo com extensão .JAD contendo informações referentes ao *MIDlet*, e outro .JAR, que é arquivo compactado no formato *PkZip* com todas as classes e arquivos necessários (imagens/áudios/vídeos da aplicação).

Aplicativos para *Palmtops* exigem que o MIDlet (.JAR) seja convertido para um arquivo suportado pelos *Palmtops* - *Palm Resource File* (PRC). Isto pode ser feito

facilmente com a execução de aplicações já existentes disponibilizadas pela própria SUN.

2.5.2.4. JINI

A tecnologia JINI – baseada em *Java* – permite que vários tipos de dispositivos digitais sejam fáceis e rapidamente conectados em redes. Os dispositivos conectados compartilham serviços na rede. O objetivo da SUN era tornar a utilização de redes de dispositivos tão fácil como utilizar seu telefone. O JINI será utilizado com dispositivos eletrônicos de consumidor como computadores palmtop, TVs, câmeras digitais e telefones celulares [20].

A tecnologia utiliza recursos tais como serialização [HRef-15] de objetos, soquetes e RMI (*Remote Method Invocation*). Esta tecnologia traz para um nível ainda maior o conceito de "plugabilidade intercomponentes", sejam estes componentes de *hardware* ou de *software*.

Utilizando a arquitetura JINI, impressoras, câmaras de vídeo, PDA´s poderão se conectar diretamente na rede. A grande vantagem é que os equipamentos já existentes na rede saberão que o novo dispositivo foi adicionado e encontra-se disponível.

O JINI trabalha com três componentes principais: "Serviços", "Clientes" e "Localizador de Serviços". Exemplo de "Serviços" são as impressoras, câmaras de vídeo, disco rígido. Já os "Clientes" utilizam os serviços JINI disponíveis, e o "Localizador de Serviços" atua como um *broker*, localizador entre serviços e clientes. [HRef-15]

2.5.3. Plataforma JAMP

As aplicações em *Java* são geralmente direcionadas a aplicações para WEB. Porém a construção de novas aplicações desenvolvidas sobre objetos distribuídos tem requerido o desenvolvimento de novas tecnologias, pois necessitam um gerenciamento eficiente, recuperação e disseminação do conjunto heterogêneo de componentes multimídia e documentos. Com a finalidade de suprir a escassez de ferramentas distribuídas para WEB e de minimizar a complexidade desenvolveu-se a plataforma JAMP (*Java Architecture for Media Processing*). [21]

Um sistema distribuído gerencia recursos [22] espalhados por uma rede de computadores de forma transparente para os seus usuários. O importante nesses

sistemas é que os recursos sejam usados através de uma interface uniforme e independente de sua localização. Em comparação aos sistemas centralizados (mainframes), a alternativa distribuída apresenta vantagens de menor custo, maior flexibilidade e confiabilidade.

No mundo real, muitas vezes existem vários objetos do mesmo tipo. Na terminologia da orientação a objetos, diz-se que a classe bicicleta, por exemplo, instancia vários objetos do tipo bicicleta. Todos eles possuem os mesmos atributos e métodos. Entretanto, o estado de cada bicicleta, representado pelos valores armazenados em cada atributo, é independente e pode ser diferente de uma bicicleta para outra.

Desta forma, pode-se tirar vantagem de que os objetos do mesmo tipo são similares e criar uma "fôrma" para estes objetos. Tais fôrmas de *software* são chamadas de classes. Resumidamente, uma classe é uma estrutura de dados que armazena os atributos e define os métodos de uma certa categoria de objeto.

As classes abstratas são assim denominadas por não permitirem a criação de objetos do seu tipo. Seu uso é dirigido para a construção de classes modelo, ou seja, de especificações básicas de classes através do mecanismo de herança.

A Arquitetura *Java* para Processamento de Mídia (JAMP) [23] é mostrada na figura 26. Sua organização segue o modelo proposto pela OMG, dividida em Objetos da Aplicação, Mecanismo de Distribuição de Objetos, Objetos de Serviço e *Frameworks* de Domínio.

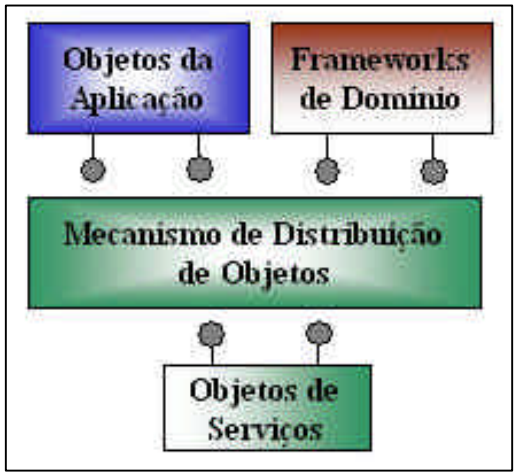

Figura 26: Métodos de uma aplicação midlet

As estruturas citadas na figura 24 são assim representadas: ***Objetos de Aplicação*** refere-se às aplicações existentes que utilizam a JAMP como uma plataforma de distribuição; ***Mecanismo de Distribuição de Objetos*** utiliza o modelo *Java* RMI como

principal forma de distribuição e controle de objetos; ***Frameworks de Domínio*** representa os domínios de aplicação (Áudio, Vídeo, Rope, Midi, Trabalho Colaborativo, Comunicação de Grupo e Comércio Eletrônico); ***Objetos de Serviço*** representa a estrutura imprescindível para o funcionamento da JAMP, o JBroker, que atua na arquitetura realizando o processo de *trading*. [23]

O JBroker é um *objeto de serviço* que oferece importantes funcionalidades para a plataforma. Contém uma base de dados dos servidores disponíveis no momento e permite que os clientes encontrem os servidores distribuídos (objetos remotos) na rede.

As aplicações multimídia distribuídas JAMP [24] podem usar todos os *frameworks de domínio* disponíveis na plataforma. Os serviços dos *frameworks* são embutidos nas aplicações através de sua compilação. As aplicações podem ser: Aplicações *Java*, que são executadas pela máquina virtual *Java* (JVM); ou *Applets*, que são executadas pelos *Browsers* ou *Applet viewers* que executam a JVM.

Os *frameworks de domínio*, dispõem de APIs que são usadas pelas aplicações para incorporar as necessidades de processamento de mídias, trabalho cooperativo e permitir a comunicação distribuída. Nesta camada nota-se a flexibilidade que a plataforma JAMP oferece às aplicações para WEB, pois novos *frameworks* podem ser criados conforme a necessidade, ou seja, a plataforma não está restrita apenas aos *frameworks* citados [25].

2.5.4. Tecnologias de Aparelhos Móveis

Pode-se entender como comunicação móvel aquela o qual existe a possibilidade de movimento relativo entre partes ou as partes sistêmicas envolvidas. Como exemplo, a comunicação de uma aeronave e uma base terrena, ou mesmo a comunicação entre telefones celulares. Já em uma comunicação fixa, não há movimento entre as partes envolvidas. Temos como exemplo um link de microondas entre uma estação rádio base e uma central de comutação e controle de um sistema de telefonia celular.

Até início da década de 90 [HRef-12] a transmissão de dados sobre redes de comunicação móvel era através da conexão de *modems* de baixa velocidade a terminais telefônicos móveis analógicos. Está época foi conhecida como a primeira geração de celular.

O serviço celular funcionava através da divisão de uma cidade ou região em pequenas áreas geográficas denominadas células, sendo cada uma delas servida pelo seu próprio conjunto de rádios transmissores e receptores de baixa potência. O tamanho das

células situa-se na faixa de 500 metros a 10 quilômetros. Quando a chamada de um celular alcança uma torre de transmissão e recepção, a mesma é transferida para o sistema de telefonia fixa regular [HRef-12]. Cada célula possui diversos canais com o objetivo de prover serviços para muitos usuários simultaneamente. À medida que um usuário se movimenta na cidade, o sinal do seu telefone celular passa automaticamente de uma célula para outra, sem sofrer interrupção (fenômeno conhecido como "*handoff*" ou "*handover*").

Essa primeira geração de sistemas celulares caracterizava-se basicamente por ser analógica, utilizando modulação em freqüência para voz e modulação digital FSK (*Frequency Shift Keying*) para sinalização. O acesso à canalização é obtido através do FDMA (*Frequency Division Multiple Access*) [HRef-12]. Nessa época, havia vulnerabilidade às interferências e facilidade de interceptação (escuta) das conversações.

Na metade da década de 90 os celulares digitais surgiram como uma opção. Estes celulares permitem conexões mais confiáveis e com menor ruído. Esta tecnologia, que veio depois da tecnologia pregada pela primeira geração (1G) de telefones móveis, foi batizada de 2G

A tecnologia 2G [HRef-13] permite transmissões com taxas de até 19 kbps (*kilobits* por segundo) e é oferecida pelas operadoras de telefonia em várias opções de pacotes. Fazem parte desse segmento os padrões CDMA (*Code Division Multiple Access*), TDMA (*Time Division Multiple Access*) e GSM (*General System Mobile Communication*).

Atualmente o Brasil já está trabalhando com a tecnologia 2,5G. Através dela o usuário pode conectar seu *laptop* ou seu PDA à internet de qualquer lugar com velocidade de até 144kbps através dos celulares 2,5G ou ainda da placa PCMCIA. A tecnologia 2,5G disponibiliza serviços como envio e recebimento de mensagens multimídia, download de vídeo, monitoramento à distância. (*figura 27*)

A expectativa é de que, daqui a 2 anos, [26] a massificação possibilitada pela queda de preços faça com que quase metade dos usuários de telefonia celular usufruam desses serviços. Esse crescimento estará refletido também no tráfego gerado por eles, de modo que o número de mensagens SMS e os minutos WAP devam triplicar em 2006.

Figura 27: Conexão de aparelhos móveis conectados na Internet

Mesmo não estando ainda os sistemas de segunda geração totalmente amadurecidos e firmemente estabelecidos, já se trabalha intensamente no desenvolvimento da terceira geração. Este trabalho está sendo liderado mais uma vez pela Europa e patrocinado pelo ITUR (*International Telecommunications Union - Radiocommunications* sector) e ETSI (*European Telecommunications Standard Institute*). O objetivo é criar um sistema móvel de terceira geração que será denominado UMTS (Universal Mobile Telecommunications System).

Atualmente, milhões de assinantes [27] (exemplo: assinantes das operadoras na Coréia do Sul) utilizam o padrão CDMA2000 da tecnologia 3G. Estes assinantes usuários dos serviços de dados HSPD - *High Speed Packet Data* (dados/pacotes em alta velocidade), têm ao seu dispor, uma gama de aplicações que somente uma tecnologia com velocidade de transmissão de dados superior a 2 Mbps poderia oferecer, tais como: transmissão *on-line* e *real-time* de eventos, shows e programas de TV (ex: jogos de futebol da Copa FIFA 2002), jogos multiusuários de alta-definição, vídeo-conferência, acesso à Internet em alta velocidade, dentre outros.

As chamadas de dados do tipo HSPD, podem propiciar transmissões de dados de até 153,6 kbps, tanto no *download* quanto no *upload*, simultaneamente. A disponibilidade destas transmissões para uma chamada depende da disponibilidade dos recursos naquele momento. [27]

2.5.4.1. MMS – Serviço de Mensagens Multimídia

O MMS [28] é um serviço de mensagens multimídia para dispositivos móveis. Ele é uma evolução do SMS, no qual era permitido somente o envio de mensagens de texto. O MMS promete oferecer um ambiente completo para aplicações multimídia, envolvendo imagens, texto, áudio, vídeo etc, podendo funcionar nas redes já instaladas. Com o MMS as mensagens podem vir acompanhadas de informações sobre a sincronização e apresentação das diversas mídias enviadas.

Multimídia consiste de um ou mais elementos de mídia (texto, voz, imagem e vídeo); e a combinação destes elementos de forma ordenada e sincronizada cria uma apresentação multimídia. O MMS vai possibilitar um intercâmbio de um ou mais elemento de mídia entre usuários, sem a necessidade de o serviço ser prestado em tempo real. Deseja-se que o MMS aproveite os avanços já realizados na área de multimídia e dos serviços de mensagem atuais, adicionando requisitos específicos para mobilidade, buscando sempre manter a compatibilidade e reutilizar os padrões já estabelecidos. [28]

O MMS permite exibição de imagens em formato JPEG, WBMP, GIF87a e GIF89 que podem ser mostradas em múltiplos quadros e sincronizadas com texto e voz. Para a sincronização entre os conteúdos usa-se a *Synchronized Multimedia Integration Language* (SMIL).

O MMS (figura 28) deve englobar redes diferentes sendo que a conexão e a garantia de compatibilidade entre elas se dá pelo uso do IP e dos protocolos de mensagem da Internet. [28]

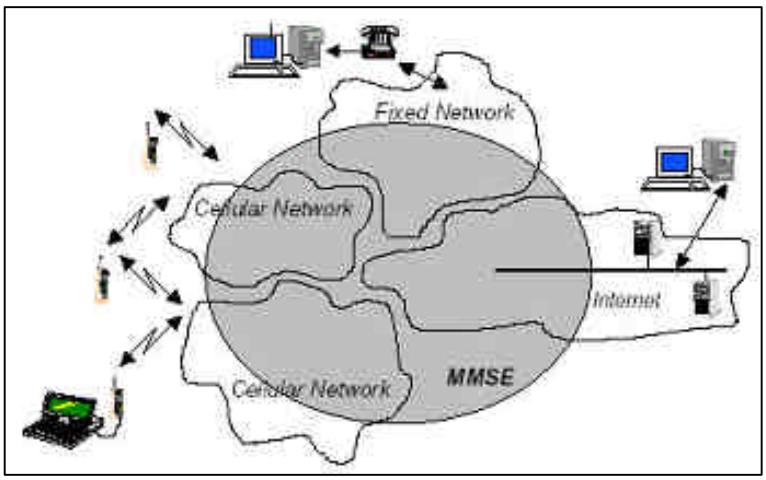

Figura 28: Integração de redes de tipos diferentes

2.5.5. Redes Wireless

Uma rede sem fio (*Wireless*) é tipicamente uma extensão de uma rede local convencional com fio, criando-se o conceito de rede local sem fio (*Wireless Local Area Network* - WLAN). Uma WLAN converte pacotes de dados em onda de rádio ou infravermelho e os envia para outros dispositivos sem fio ou para um ponto de acesso que serve como uma conexão para uma LAN com fio.

Grande parte das redes sem fio é baseada nos padrões IEEE 802.11 e 802.11b, para comunicação sem fio entre um dispositivo móvel e uma rede LAN. Esses padrões permitem transmissão de dados de 1 a 2Mbps, para o padrão IEEE 802.11, e de 5 a 11Mbps, para o padrão IEEE 802.11b, e especificam uma arquitetura comum, métodos de transmissão, e outros aspectos de transferência de dados sem fio, permitindo a interoperabilidade entre os produtos. [29]

A área coberta por uma rede sem fio padrão IEEE 802.11 é baseada na divisão por células. Essas células são denominadas de BSA (*Basic Service Area*), em que o tamanho da célula depende das características do ambiente e da potência dos transmissores/receptores usados nas estações.

Existem várias versões de rede sem fio padrão IEEE 802.11, entretanto, atualmente [HRef-14] são reconhecidas as seguintes: IEEE 802.11a-1999; IEEE 802.11b-1999/2001; IEEE 802.11d-2001; IEEE 802.11F-2003; IEEE 802.11g-2003.

As redes padrão 802.11 abrangem uma área restrita, ou seja, possuem um comportamento de uma rede local. O IEEE reconheceu em 2001 um novo padrão de rede sem fio que abrange áreas metropolitanas [30]. Este padrão recebeu o nome de 802.16, e possui uma área muito maior de cobertura, chegando a 50 quilômetros de alcance, criando verdadeiras redes metropolitanas sem fio. Como as tecnologias 802.11 e 802.16 são complementares, não há necessidade de troca de tecnologia no computador. Com isso, se conectarmos o transmissor 802.16 em uma rede tipo ADSL, por exemplo, ele enviará o sinal para todos os equipamentos compatíveis com o padrão 802.11 a uma velocidade de transmissão de dados a até 70 Mbps.

Outro método de comunicação entre aparelhos móveis e computadores é a utilização da tecnologia *Bluetooth* [31], com base em ondas de radio. A tecnologia foi concebida pela Ericsson, mas implementada pela Nokia, Ericsson, IBM, Intel e Toshiba. Como é uma ligação com base em ondas de radio, a *Bluetooth* não requer uma ligação com linha de mira para estabelecer comunicação. Pode cobrir uma pequena área de

aproximadamente 10m (por exemplo: uma sala, uma casa pequena) a uma taxa de 720 Kbps.

2.6. Considerações finais

Em uma transmissão de vídeo podemos ter diferentes tipos de computadores interligados em diferentes redes, cada uma com sua largura de banda. Ao se tratar de diferentes computadores, é preciso analisar a capacidade dos recursos oferecidos para cada computador, principalmente se tratando de uma transmissão de vídeo para um PDA. Para que este dispositivo visualize um vídeo geralmente é preciso realizar uma adaptação de conteúdo. Isto, devido às limitações impostas pelo *hardware*/*software* utilizado em um PDA. Tornar-se-á necessária a presença de um computador que execute a tarefa de realizar a adaptação de conteúdo.

Está sendo desenvolvido pelo fórum ICAP um protocolo que proporcionará a adaptação de conteúdos armazenados em servidores. O fórum ICAP [32] é uma coligação de negócios de Internet, que inclui vários tipos de provedores, tais como, *hardware* e *software*, *host*, distribuição de conteúdo, serviços de aplicação, publicidade e banda larga.

Nos sistemas móveis a necessidade de adaptação de conteúdo varia de acordo com a condição da rede de transmissão e dos recursos que cada dispositivo suporta. Definir um padrão de qualidade [32] é uma difícil tarefa, uma vez que podemos estar trabalhando com ruídos (interferências) na rede, principalmente nos sistemas móveis.

Junto ao conceito da nova era da computação (Ubíqua), observa-se um significante aumento do uso dos dispositivos embarcados, como os PDAs e smartphones. Como exemplos de outras aplicações não relacionadas ainda com a computação ubíqua e que utilizam o PDA como dispositivo cliente, se destacam:

- *CyberGuide*: aplicação de serviço de guia turístico que trabalha com contexto de localização e serviços de comunicação [33];
- *Campus Aware*: guia turístico para o campus de uma faculdade baseado na localização espacial [34];
- *Dumbo – Dynamic Móbile Meeting Board*: suporta reuniões informais sem hora e data marcada para disponibilização futura na Internet [35].

O modelo do *framework* definido neste projeto pode facilmente ser inserido como um dos serviços oferecidos na plataforma JAMP. Com isso, a plataforma passará a oferecer serviços de transmissões de mídias para diversos dispositivos, além da detecção de mudança de localização por parte do usuário deste.

Os serviços multimídia oferecidos pelo sistema serão cadastrados na JAMP. O JBroker é um *objeto de serviço* da JAMP que oferece uma base de dados dos servidores disponíveis no momento e permite que os clientes encontrem os servidores distribuídos (objetos remotos) na rede.

Os serviços de mensagens multimídias – MMS vem se tornando comum atualmente. As maiores operadoras de telefonia móvel, Vivo, Claro, Tim, já trabalham com tal tecnologia atualmente no Brasil. Outro fato considerável a favor deste projeto é a chegada da tecnologia “3G” da telefonia móvel, fornecendo uma infra-estrutura de rede necessária para transmissão de dados multimídia a uma velocidade de até 2Mbps em operações interativas.

3. Projeto do Framework

A transmissão de vídeos utilizando redes *wired* (com fio) entre computadores convencionais (PCs/laptops) não é novidade na literatura computacional. Estes vídeos podem se originar de uma câmera “ao vivo” (em tempo real) ou de uma base de dados (em forma de arquivos armazenados) e são transmitidos a partir de um servidor (geralmente PC). Exemplos de aplicações são reuniões eletrônicas e videoconferências. Contudo, a transmissão de mídias (vídeos, imagens, textos, etc.) entre diferentes dispositivos pode trazer várias facilidades à humanidade.

Um protocolo comum de comunicação entre diversificados dispositivos computacionais não deve ser um limitante para que determinadas aplicações sejam desenvolvidas e utilizadas.

Um exemplo, o qual foi uma das motivações do estudo realizado para essa dissertação, é uma aplicação destinada ao esquema de segurança de um prédio. Em um sistema de segurança que trabalhe com câmeras, é esperado que a pessoa responsável pelas imagens capturadas visualize, em uma tela, qualquer imagem suspeita. Com isso, é preciso garantir uma atenção nas imagens capturadas mesmo quando a pessoa responsável for ao banheiro ou tomar um café. Uma solução que otimize essa garantia é proporcionar o envio dessas imagens a um dispositivo móvel, como por exemplo um PDA que possua rede *wireless*, cujo responsável poderá levá-lo para onde for nas proximidades do prédio.

Outro exemplo que envolve envio de imagens para dispositivos móveis capturadas por uma câmera é uma aplicação que permita que os pais vigiem, em tempo real, o filho pequeno que ficou em casa, podendo esse estar sobre os cuidados de uma babá. Para esta visualização, eles poderiam utilizar um telefone celular ou mesmo um PDA e usariam recursos da tecnologia de celular (GSM, CDMA, etc). Sendo assim, a qualquer instante, os pais podem verificar se o filho está recebendo bons cuidados da babá ou mesmo se ele não está fazendo ações indevidas, como por exemplo brincar com objetos que lhe possa causar algum risco.

Entretanto, para a recepção de qualquer mídia em um dispositivo computacional, seja ele fixo (PC) ou móvel (PDA, celular), não se pode desprender da arquitetura cliente/servidor. É preciso definir qual mídia será disponibilizada pelo servidor e como os clientes irão visualizá-las.

Com base nessas necessidades, foi definido e implementado um *framework* que trata de funcionalidades para transmissão de mídias (vídeos, textos, seqüência de imagens, etc.) para PCs e aparelhos móveis (PDAs e celulares). Foram também implementados mecanismos para detectar a ausência e presença de um usuário ao seu dispositivo cliente, através de sensores que podem estar representados por *webcams* convencionais.

A parte ubíqua do *framework* desenvolvido neste trabalho, é representada pela capacidade do sistema em detectar a ausência ou presença de um usuário em um determinado dispositivo através de sensores, e a possibilidade de se enviar uma mídia para dispositivos embarcados. Quando detectada uma mudança de contexto (ausência/presença do usuário diante de um dispositivo), o sistema poderá realizar uma determinada tarefa que faça com que o usuário de forma transparente e, de certa forma, com mobilidade, continue a receber informações que estava recebendo em outro dispositivo, como por exemplo, começar a receber esta mídia em um dispositivo móvel.

Optou-se pelo desenvolvimento de um *framework* para que esse fosse uma solução genérica para várias aplicações relacionadas à transmissão de mídias para dispositivos convencionais e embarcados. Ele é composto por um conjunto de classes que implementam o comportamento genérico de um servidor de mídia, de um receptor de mídia, de um intermediador (que gerencia os servidores disponíveis), e de um detector de presença.

No desenvolvimento do *framework* percebeu-se a necessidade de dividi-lo em dois módulos, nos quais cada módulo assumiu características de um novo *framework*. Isto porque a plataforma *Java* utilizada para os dispositivos comuns (PCs e PDAs avançados) difere-se da plataforma utilizada para grande parte dos dispositivos embarcados (celulares e PDAs convencionais), e a partir deste instante, estes dois módulos são referidos como dois *frameworks* distintos. Foram modelados e desenvolvidos dois *frameworks*, um destinado à tecnologia *Java* J2SE (para PCs e PDAs avançados) e outro J2ME (celulares e PDAs convencionais). Essa divisão em módulos e suas classes individuais podem ser vistas na figura 29.

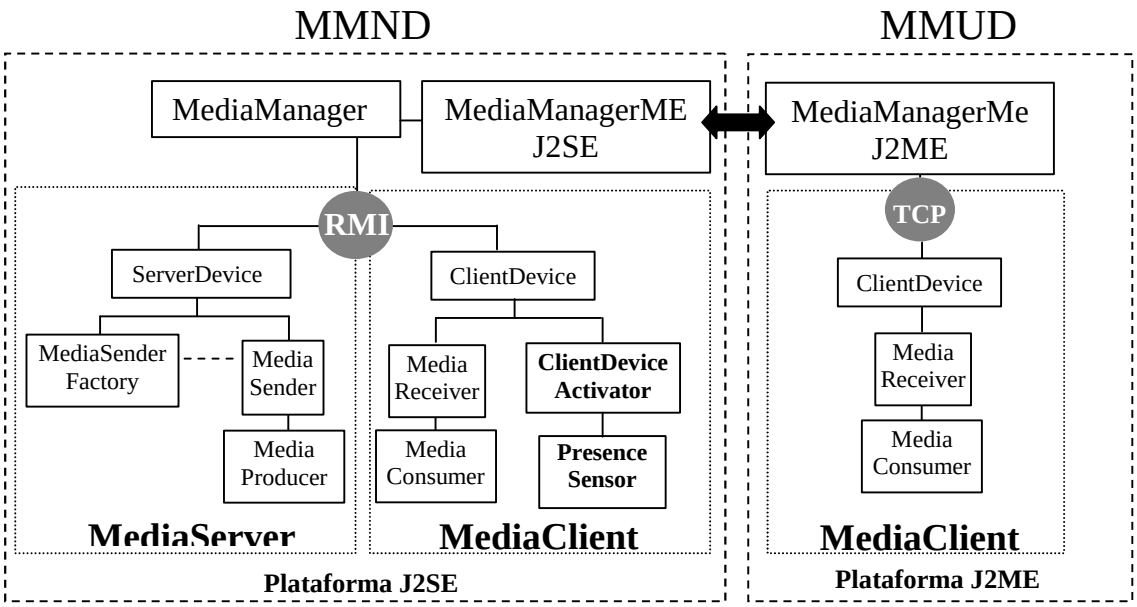

Figura 29: Classes dos frameworks desenvolvidos

A comunicação e utilização dos dois *frameworks* ocorrerão somente quando o dispositivo cliente da mídia estiver trabalhando com a plataforma J2ME com configuração CLDC, ou seja, quando se tratar de um telefone celular ou um PDA convencional. Caso contrário, será utilizado somente o *framework* que está implementado sobre a plataforma J2SE.

Já a comunicação entre cliente/servidor da mídia pode ser visualizada em 3 camadas, como pode ser observado na figura 30. A terceira camada representa a comunicação entre as aplicações presentes nos dispositivos clientes e servidores da mídia. Nesta camada um cliente requisita de um servidor a mídia. A segunda camada representa o controle do envio e recepção da mídia, tratando-se de especificações de camada de rede. A primeira camada trata da produção/captura da mídia (*MediaProducer*) no servidor, que pode ser representado pela captura de imagens através de uma câmera conectada a um PC, e seu posterior consumo (*MediaConsumer*) no cliente.

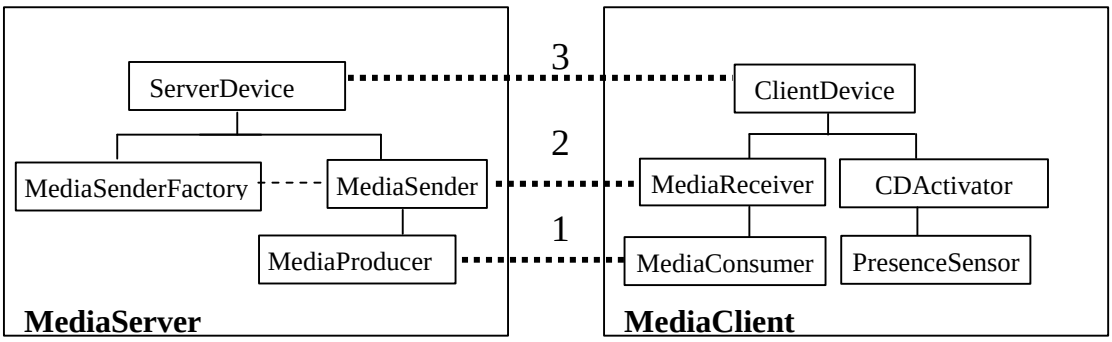

Figura 30: As 3 camadas definidas nos frameworks MMND e MMUD

3.1. Arquitetura definida

Foi definida uma arquitetura dividida [36] em três objetos: Servidor de Mídia (*MediaServer*), Cliente de Mídia (*MediaClient*), e Gerenciador de Servidores e Cliente de Mídia (*MediaManager*).

A arquitetura inicial definida possui 2 camadas (figura 31): Aplicação e Suporte. A camada de aplicação é composta pelos seguintes objetos:

- *MediaServer*: Presente nos servidores de mídia. Cadastra a Mídia no gerenciador e envia dados (streaming de vídeo/áudio/...) para os dispositivos clientes.
- *MediaClient*: Presente nos dispositivos dos clientes. Recebe dados (streaming de vídeo/áudio/...) de algum servidor e os apresentam através do dispositivo a ele associado, e detecta a ausência/presença de movimento.

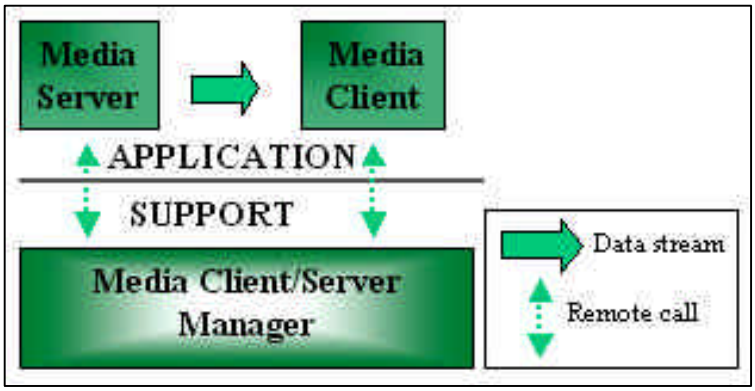

Figura 31: Arquitetura inicial definida para serviços multimídia

A camada de suporte é definida pelo objeto *Media Client/Server Manager*, que será chamado apenas de *MediaManager*. Ele gerencia o controle de mídias disponíveis no sistema, controla as requisições dos clientes e inicia o processo de transmissão de mídia entre *MediaServer* e *MediaClient*.

Para a comunicação entre os três objetos é utilizado o recurso de localização de objetos remotos de um *Broker* [21], que viabiliza a passagem e recepção de parâmetros (*marshling*/*unmarshling*). A função de *marshling* realizada pelo *Broker* consiste na serialização dos parâmetros para a transmissão via rede em um determinado formato, já o *unmarshling* recebe esta transmissão e transforma o formato dos parâmetros recebidos para um formato tipo local.

O *Broker* deve receber uma requisição de um objeto ou serviço do cliente, localizar o objeto que implementa esta chamada, retornando ao cliente uma referência a

este objeto. Com isso o cliente não precisa preocupar-se com a localização do objeto que está sendo invocado, com a linguagem de programação que o mesmo foi programado ou, mesmo, com o sistema operacional que está sendo utilizado.

O *middleware* é uma entidade que possui a função de estabelecer os relacionamentos cliente-servidor entre os objetos. Através do *middleware*, um cliente pode invocar transparentemente um método de um objeto no servidor, que pode estar na mesma máquina ou em qualquer ponto da rede. Um exemplo de *middleware* bastante utilizado é o serviço de ODBC (*Open Database Connectivity*) disponível nos sistemas operacionais, que fornece informações sobre como se conectar a um banco de dados local.

A integração da arquitetura proposta com um middleware, é representada da seguinte maneira: *MediaClient* e *MediaServer* se comunicam com o *MediaManager* que associado a um *Broker* oferece serviços de um middleware (figura 32). Dentre as tecnologias existentes que poderiam ser utilizadas para prover o serviço de Broker se destacam: *Java*-RMI, CORBA, DCOM.

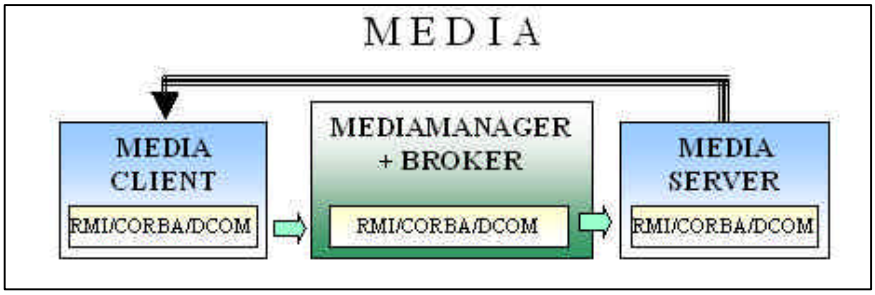

Figura 32: Serviço de middleware e arquitetura definida

Java-RMI é uma tecnologia *Java*, e a interface utilizada pelas aplicações é o RMI (*Remote Metod Invocation*). No caso de CORBA (tecnologia do grupo OMG - *Object Management Grou*p), a interface utilizada é o ORB (*Object Request Broker*). Já o DCOM (*Microsoft*) utiliza a interface COM (*Component Object Model*).

Para esta arquitetura, a interface em que a aplicação do cliente estará trabalhando é o que importa para que ele possa invocar métodos remotos de outros objetos; o middleware oferece uma solução para que aplicações em máquinas diferentes se comuniquem em ambientes distribuídos e heterogêneos, ao mesmo tempo em que interconectam múltiplos sistemas baseados em objetos.

3.2. Arquitetura definida e a plataforma Jamp

A plataforma JAMP conforme já mostrada no tópico 2.5.2 deste trabalho, possui um objeto de serviço, o JBROKER, no qual sua principal funcionalidade é gerenciar uma base de dados dos servidores disponíveis no momento, permitindo que os clientes encontrem os servidores distribuídos (objetos remotos) na rede.

O objeto *MediaManager* da arquitetura definida se cadastra na plataforma JAMP (figura 33) através do JBROKER [24]. Os objetos *MediaServer* e *MediaClient* invocam remotamente os métodos definidos no *MediaManager*, que é um *framework de domínio* cadastrado na plataforma JAMP. Posteriormente, utilizam implicitamente, os métodos definidos no *MediaManager* como se fossem métodos locais, sem se preocupar onde estes objetos se localizam, qual seu n° IP, porta, etc.

A JAMP, por sua vez, já possui vários *frameworks de domínio*, dentre eles se destacam os domínios de aplicação Áudio, Vídeo, Rope, Midi, Trabalho Colaborativo, Comunicação de Grupo e Comércio Eletrônico. Este trabalho insere uma nova funcionalidade à JAMP, que é de gerenciar o envio da mídia e recepção desta através de um PC ou mesmo de um celular ou PDA.

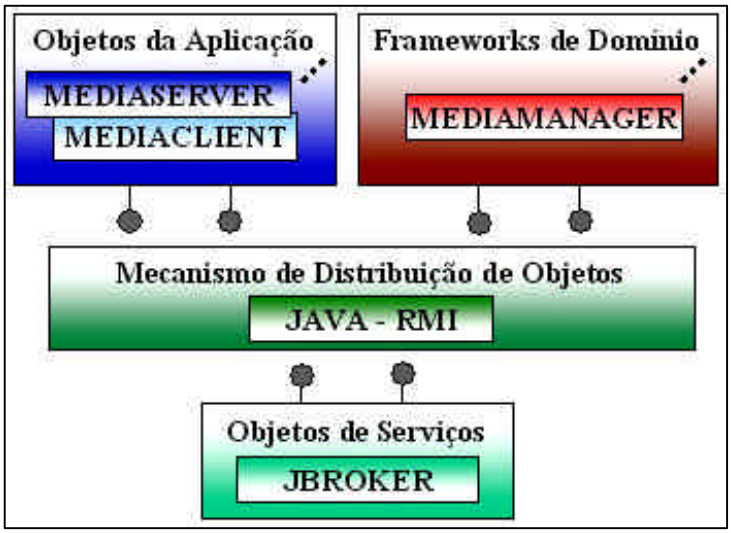

Figura 33: arquitetura definida e a JAMP

O diagrama de seqüência (figura 34) é utilizado para mostrar a seqüência dos eventos realizados, envolvendo o *MediaServer*(*Server*), *MediaClient*(*Client*), *Media Client/Server Manager*(*Manager*) e a arquitetura JAMP.

Inicialmente o *Manager* exporta sua referência para a JAMP (‘*Registration*’). Esta referência condiz o seu tipo (tipo gerenciador de mídia). O *Server* requisita da JAMP o endereço (localização) do *Manager* (‘*Lookup1*’) e se cadastra no *Manager* por

chamada remota de procedimento (‘RPC1’). Ele informa ao *Manager* o seu tipo (tipo servidor de mídia), e os dados da mídia (Midia1 – Tamanho:160 x 120 – Mpeg1 – 10kbps - Colorido) que ele está disponibilizando naquele momento.

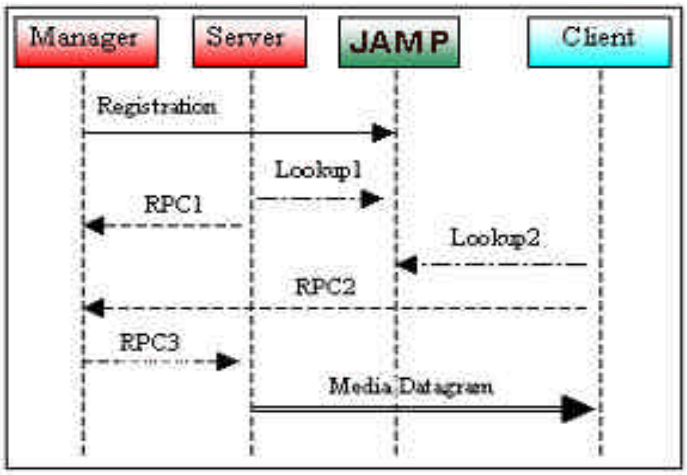

Figura 34: Diagrama de Seqüência – Manager, Server, Client X JAMP

Assim como o *Server*, o *Client* requisita da JAMP o endereço do *Manager* (‘*Lookup2*’). Após o ‘*Lookup2*’ o dispositivo cliente estará apto a visualizar os métodos definidos no *Manager*. Ele solicita então, uma mídia de um servidor cadastrado naquele instante no *Manager* (‘RPC2’). Implicitamente o *Manager* requisita (‘*RPC3*’) do *Server* o início da transmissão por datagramas para o *Client* em questão.

3.3. Os Frameworks desenvolvidos

Foram desenvolvidos dois [36] *framework*s, um destinado aos dispositivos que utilizam a tecnologia J2SE (PCs e laptops) e outro destinado aos dispositivos que utilizam a tecnologia J2ME (PDAs, celulares...). Para o primeiro *framework* (figura 35) foi dado o nome de MMND (*Media Manager Normal Device*) e para o segundo MMUD (*Media Manager Ubiquitous Device*).

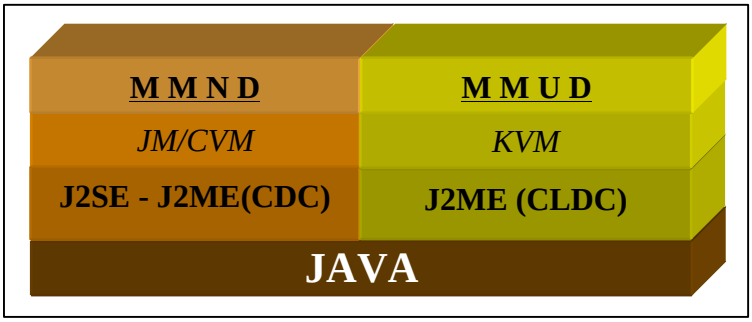

Figura 35: Framework MMND e MMUD

4. Implementação dos Frameworks

A implementação dos *framework*s será apresentada separadamente. Serão mostradas inicialmente as classes e principais linhas de códigos referentes ao *framework* MMND (4.1), e posteriormente ao *framework* MMUD (4.2).

4.1. Framework MMND

O MMND (*Media Manager Normal Device*) é constituído por um conjunto de classes que dão todo o suporte para a implementação dos serviços oferecidos na arquitetura definida nos tópicos anteriores (figura 36). Para cada objeto definido na arquitetura foram criadas classes que implementam sua funcionalidade.

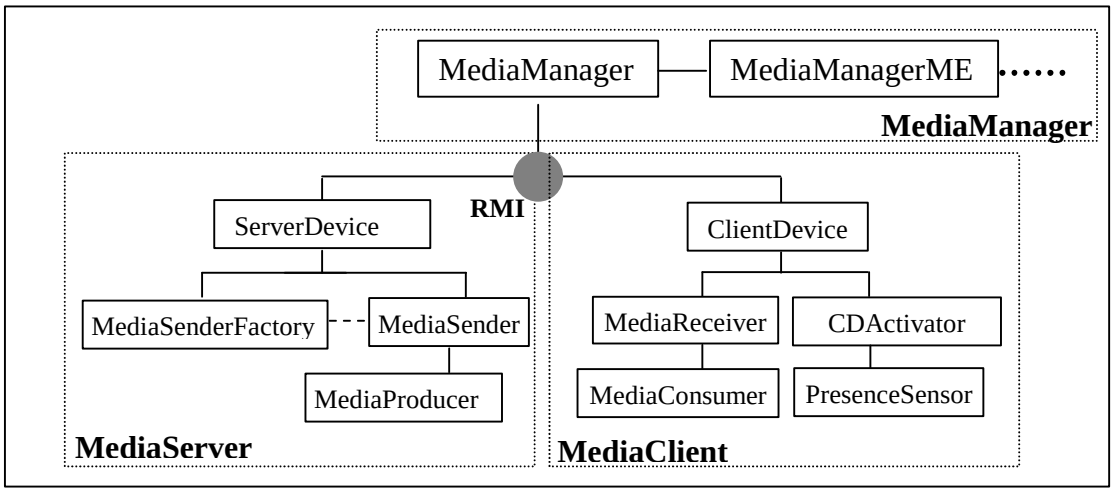

Figura 36: Classes do framework MMND

4.1.1. MediaManager

O *MediaManager* encontra-se representando por quatro classes (figura 37): *MediaManager.java*, *MediaManagerImpl.java*; *MediaManagerME.java* e *MediaManagerME_Skel.java*.

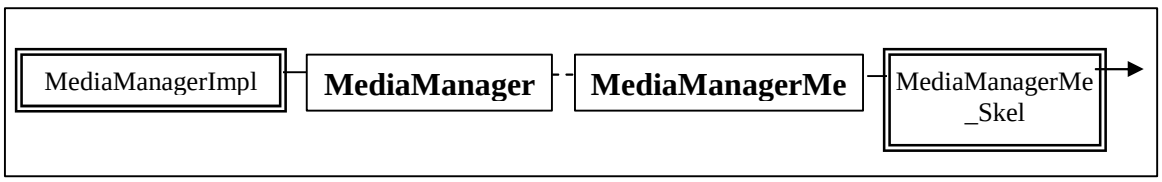

Figura 37: Classes relacionadas ao MediaManager (framework MMND)

O *MediaManager.java* define a interface que é implementada pela classe *MediaManagerImpl.java.* Esta interface define os métodos (figura 38) para adição, remoção e listagem do *MediaServer* no *MediaManager*, bem como os métodos que inicializam e finalizam a transmissão da mídia para os dispositivos clientes (*MediaClients*).

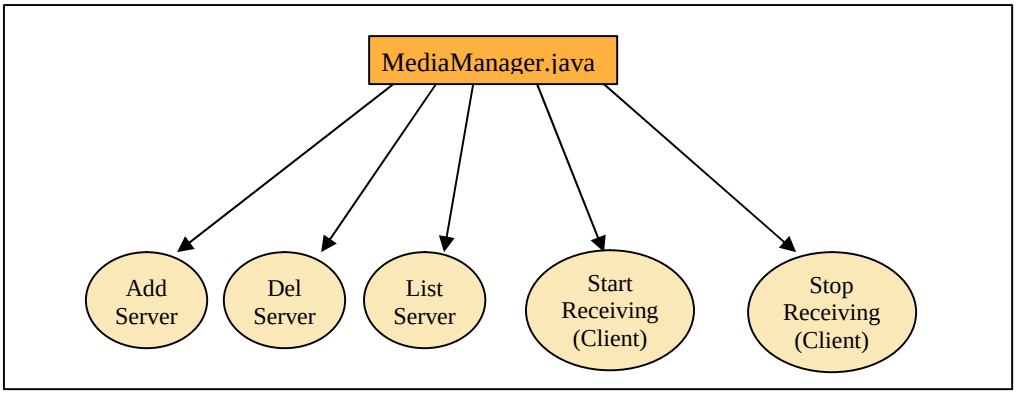

Figura 38: Interfaces definidas no MediaManager (framework MMND)

O *MediaManagerImpl.java* é responsável também pelo cadastro dos serviços oferecidos pelo *MediaManager* na plataforma JAMP (‘*this.session.broker.export ("MediaManager", this*)’).

Já o *MediaManagerME.java* define a interface que é implementada pela classe *MediaManagerME_Skel.java.* Esta interface define os métodos que inicializam e finalizam a transmissão da mídia destinados aos dispositivos que utilizam as classes definidas no *framework* para a plataforma J2ME.

É observado na figura 39 que os dispositivos que possuem recursos computacionais mais potentes (maior quantidade de memória e processamento) utilizam a JAMP para “encontrar” (‘*lookup*’) o objeto *MediaManager*. Já os dispositivos computacionais com escassez de recursos conectam diretamente no *MediaManagerME* via TCP. Isto devido ao fato da configuração (CLDC) da plataforma J2ME que é a utilizada nestes dispositivos não implementar o RMI, e por sua vez, a base da comunicação com a JAMP é o RMI.

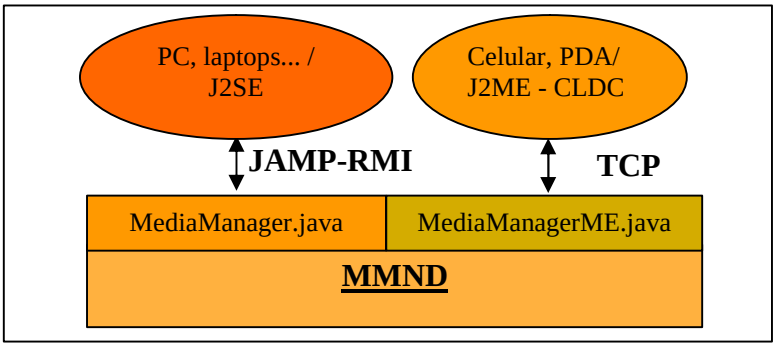

Figura 39: Media Manager e MediaManagerME (framework MMND)

4.1.2. MediaServer

O *MediaServer* é responsável pelo envio da mídia a um *MediaClient* via UDP. Ele é constituído por oito classes (figura 40): quatro definindo interfaces e quatro implementando as interfaces definidas nas primeiras. As interfaces definidas são: ***ServerDevice.java***, ***MediaSenderFactory.java***, ***MediaSender.java*** e ***MediaProducer.java***.

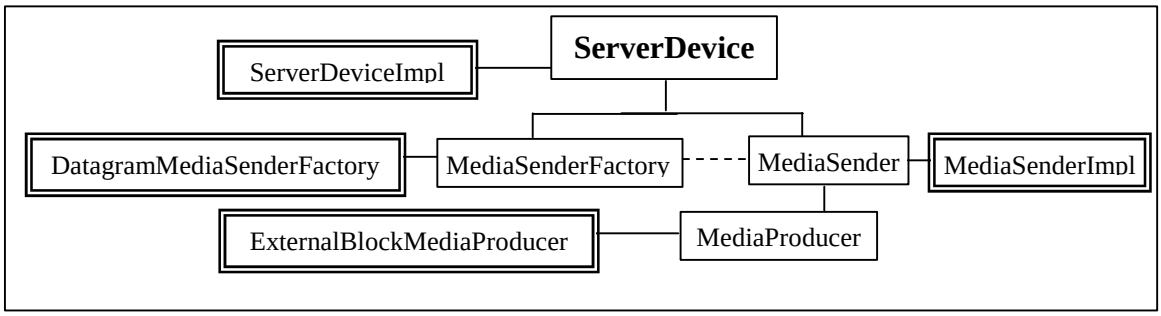

Figura 40: Classes relacionadas ao MediaServer (framework MMND)

ServerDevice.java: define a interface dos métodos (figura 41) que são implementados pela classe *ServerDeviceImpl.java*. São eles: 1)iniciar uma transmissão de uma mídia; 2) interromper uma transmissão; 3)verificar se o servidor está capturando uma mídia; 4) adicionar atributos da mídia disponível no servidor (tipo de mídia enviada, formato, qualidade, etc); 5) remover atributos e 6)para alterar os atributos.

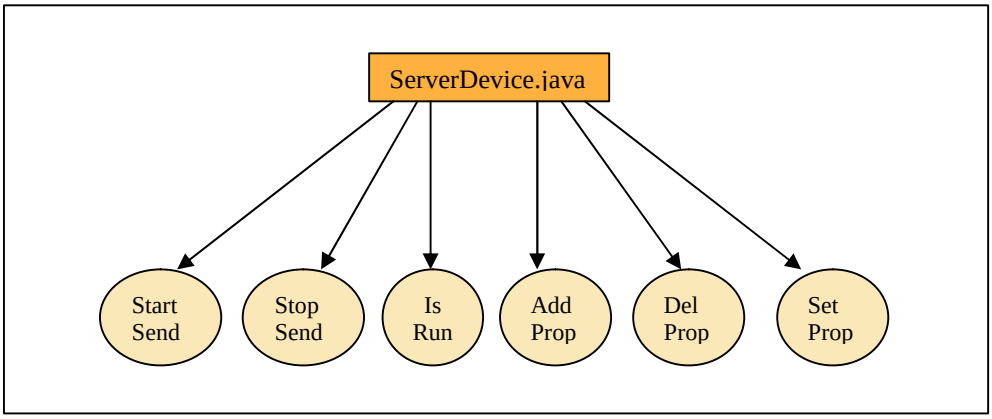

Figura 41: Interfaces definidas no MediaServer (framework MMND)

O objeto *MediaServer* se cadastra no *MediaManager* utilizando o recurso de broker da JAMP:

```
MediaManager mm = (MediaManager)session.broker.invoke("MediaManager");
mm.addServer(name, this);
```

MediaSenderFactory.java: define a interface dos métodos que são implementados pela classe *DatagramMediaSenderFactory.java.* Ela cria um objeto do tipo *MediaSender* para cada requisição de cliente.

MediaSender.java: define a interface implementada pela classe *DatagramMediaSender.java* responsável por receber a mídia capturada pelo objeto definido em *MediaProducer.java* e empacotá-la em um *buffer* para o envio.

MediaProducer.java: define a interface implementada pela classe *ExternalBlockMediaProducer.java.* Ela implementa uma *thread* que recebe dados provindos de uma aplicação externa e armazena em um `buffer (private byte[] resource = new byte[65000])`

Para que uma aplicação externa utilize o *framework* definido para inserir um *MediaServer*, ela deve criar um objeto do tipo *MediaSenderFactory* e outro do tipo *MediaProducer* . Exemplo:

```
    MediaSenderFactory mediaSenderFactory = new DatagramMediaSenderFactory();
    MediaProducer mediaProducer = new ExternalBlockMediaProducer ("cam --type
jpg");
    new  ServerDeviceImpl(name, javabrokerHost, mediaSenderFactory,
                                                    mediaProducer);
```

4.1.3. MediaClient

O *MediaClient* é responsável pela recepção de uma mídia via UDP oriunda de um *MediaServer*, e pela detecção da presença ou ausência da pessoa que assiste à mídia. Ele é constituído por oito classes (figura 42): quatro que definem interface, três que implementam as interfaces e uma que implementa uma *thread* responsável pela detecção da presença/ausência(***ClientDeviceActivator.java***). As interfaces definidas são: ***ClientDevice.java, MediaReceiver.java, MediaConsumer e PresenceSensor.java.***

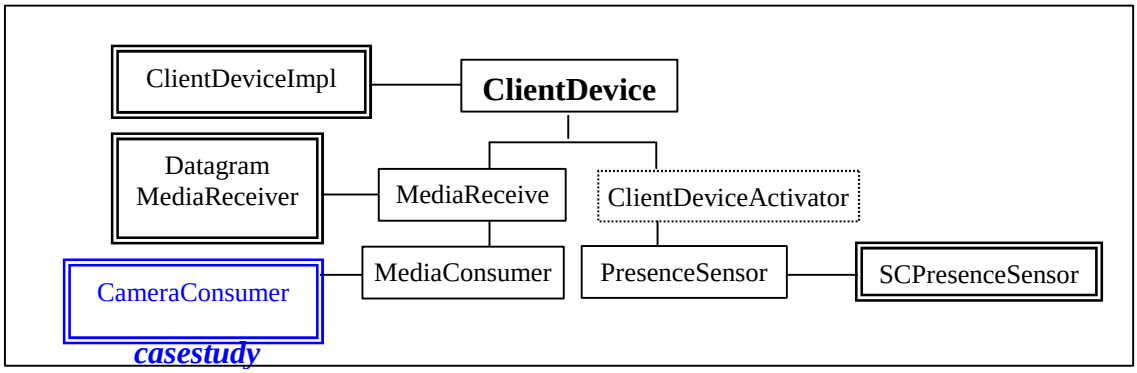

Figura 42: Classes relacionadas ao MediaClient (framework MMND)

ClientDevice.java*:* define a interface dos métodos que são implementados (figura 43) na classe *ClientDeviceImpl.java*. São eles: 1)iniciar recepção de mídia; 2)interromper recepção de mídia; 3)adicionar atributos das características do cliente; 4) remover atributos e 5) alterar atributos; 6)pausar recepções de mídias; 7) continuar recepções de mídias; 8) interromper todas recepções de mídias.

A implementação dos três últimos métodos recebem parâmetros provindos da aplicação responsável pela detecção de mudança contextual

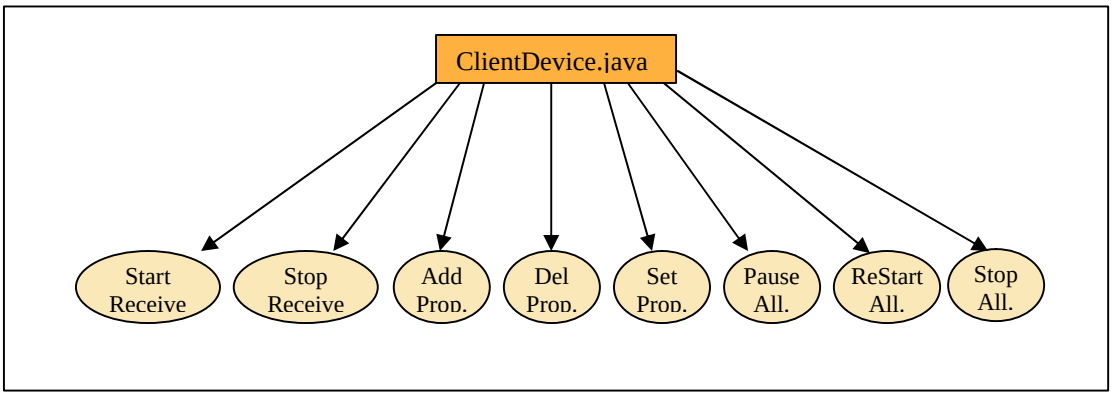

Figura 43: Interfaces definidas no MediaClient (framework MMND)

O objeto *ClientDevice* comunica com o *MediaManager* utilizando o recurso de broker da JAMP:

```
JBSession session = new JBSession(javabrokerHost);
this.mm = (MediaManager)session.broker.invoke("MediaManager");
```

MediaReceiver.java: define a interface dos métodos que são implementados pela classe *DatagramMediaReceiver.java.* Inicia uma *thread* de escuta para a recepção da mídia: `mediaConsumer.consume(dp.getData(), 0, dp.getLength()).`

É criado um *buffer* tipo datagrama que irá receber os pacotes UDPs oriundos da aplicação criada a partir da interface definida pela classe *MediaConsumer.java*.

Para cada requisição (figura 44) de mídia é criado um objeto *MediaSender* no *MediaServer* e um *MediaReceive* no *MediaClient* (cliente que requisitou a mídia).

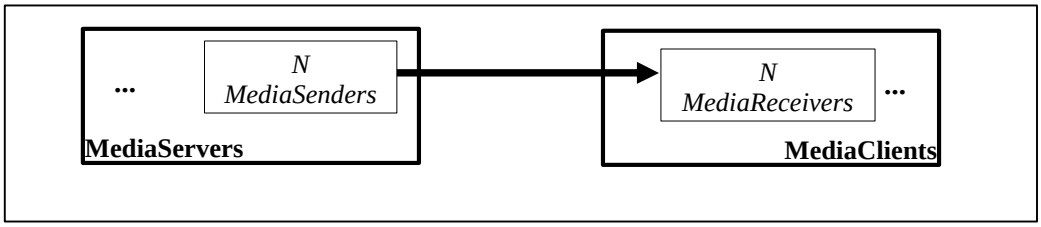

Figura 44: Media Sender X MediaReceiver

MediaConsumer.java: define a interface que deverá ser implementada por uma classe externa (dependerá do estudo de caso). Responsável pela recepção (interpretação) do *stream* gerado pelo *MediaSender.* A contínua recepção destes dados é controlado pelo *MediaReceiver.java.*

ClientDeviceActivator.java: esta classe implementa uma *thread* responsável pela detecção da presença/ausência da pessoa que assiste a mídia. Ela foi criada para detectar a mudança de contexto de localização desta pessoa, geralmente representada pela ausência ou presença da pessoa à frente do dispositivo que recebe uma mídia .

A *thread* implementada na classe *PresenceSensor* recebe a cada tempo (*sleeptime*) uma imagem capturada de uma *webcam* conectada ao dispositivo o qual a pessoa esta assistindo a mídia.

Se o sistema detecta que não houve diferença das imagens em um determinado período (*leaveTimeThreshold*), indica que muito provavelmente não esta capturando movimento algum, ou seja, indica uma ausência ou saída da pessoa, e mediante a este resultado o dispositivo do cliente requisita uma pausa na transmissão

(*clientDevice.pauseAll()*) . Esta diferença é calculada comparando as diferenças *byte* a *byte* entre uma imagem e outra.

```
public class ClientDeviceActivator implements Runnable {
  private int arriveCountThreshold = 2; //detecta presença de 2 amostras
diferentes
  private long diffThreshold = 1800000; //long representa diferença  das
imagens
  private long leaveTimeThreshold = 20000; //tempo em milisegundos para
interromper uma transmissão caso não detecte diferenças
  private long sleeptime = 1000; //tempo em milisegundos entre as
                                 capturas
 ...
```

Quando é detectada uma diferença das imagens quando o sistema estiver como "pessoa ausente", é requisitado um reinício da transmissão (*clientDevice.continueAll();*)

Para que uma aplicação presente em um *MediaClient* requisite uma mídia de um *MediaServer* cadastrado no *MediaManager*, ela deve referenciar um objeto *ClientDevice*, *MediaConsumer*, *MediaReceiver* e um *ClientDeviceActivator* . Exemplo:

```
...
ClientDevice cd = new ClientDeviceImpl(args[0], args[13]);
MediaConsumer mc = new CameraConsumer();
MediaReceiver mr = new DatagramMediaReceiver(Integer.parseInt(args[3]),
mc);
ClientDeviceActivator cda = new ClientDeviceActivator(cd, new
            SCPresenceSensor("cam --type ppm", 500000));
while(true) {
    cd.startReceiving(mr, args[2]);
…
```

A detecção de presença poderia ocorrer com o uso de um dispositivo fotossensor ou um sensor eletromagnético, freqüentemente utilizados em detecção de movimentos em alarmes residenciais (ver tópico 2.5.1). Nesse caso as classes *ClientDeviceActivator* e *PresenceSensor* seriam substituídas por uma simples classe para controlar os sinais oriundos desses dispositivos, e que estariam conectados a uma porta de comunicação (entrada serial/paralela/usb) do computador.

Contudo, como a idéia do sistema é detectar a presença do cliente do mesmo, é preciso tomar cuidado com agentes externos, como por exemplo a detecção de movimentos de pessoas e animais localizados distantes do dispositivo receptor da mídia. Para resolver essa situação é preciso que os dispositivos sensores estejam localizados em posições estratégicas, como por exemplo focalizando apenas a mesa onde se encontra o receptor da mídia.

4.2. Framework MMUD

O MMUD (*Media Manager Ubiquitous Device*) é constituído por um conjunto de classes (figura 45) que dão suporte para que os dispositivos móveis (PDAs, celulares) utilizem os serviços oferecidos pelo *framework* MMND. Vale salientar que os dispositivos computacionais com escassez de recursos, não utilizam RMI, e realizam a conexão via TCP com o *MediaManager*, não utilizando o recurso de *Broker* da JAMP.

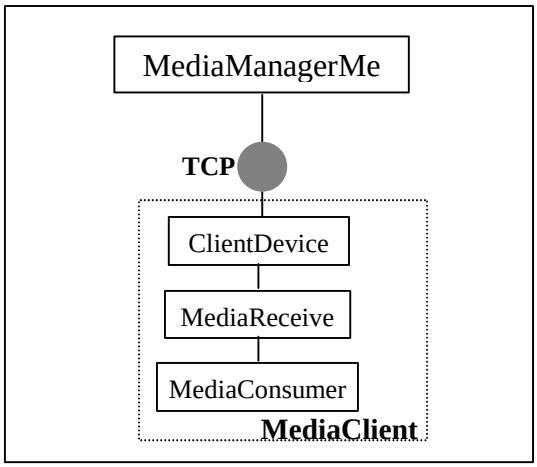

Figura 45: Classes do framework MMUD

4.2.1. MediaManagerMe

O *MediaManagerMe* no MMUD encontra-se representando por duas classes: *MediaManagerMe.java e MediaManagerMe_Stub.java* (figura 46).

O *MediaManagerMe.java* define a interface que é implementada pela classe *MediaManagerMe_Stub.java.* Essa interface define os métodos para iniciar e finalizar a transmissão da mídia destinados aos dispositivos que utilizam a plataforma J2ME com configuração CLDC.

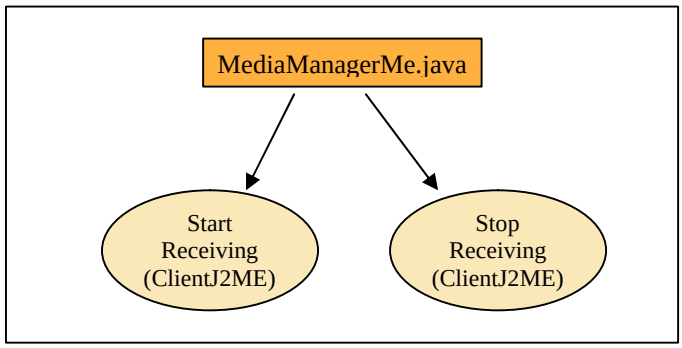

Figura 46: Interfaces definidas no MediaManagerME (framework MMUD)

O *MediaManagerMe_Stub.java* se comunica com o *MediaManagerME_Skel.java* do *framework* MMND (figura 47), no qual é criada uma porta para esta comunicação.

O código da conexão entre *MediaManagerMe_Stub.java* do *framework* MMND com o *MediaManagerME_Skel.java* do *framework* MMUD se encontra a seguir:

MediaManagerME_Skel:

```
...
this.mediaManager = mediaManager;
this.serverSocket = new ServerSocket(port);
this.thread = new Thread(this, this.getClass().getName());
this.thread.setDaemon(true);
this.thread.start();
System.out.println( "MediaManagerME executando na porta " + port );
...
```

MediaManagerMe_Stub:

```
...
sc = (StreamConnection) Connector.open("socket://" + mediaManagerHost);
dis = new ObjectInputStream(sc.openDataInputStream());
dos = new ObjectOutputStream(sc.openDataOutputStream());
...
```

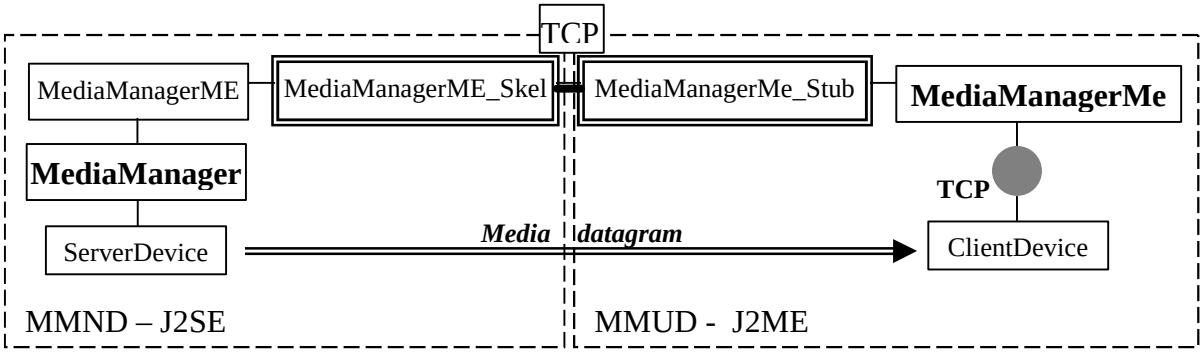

Figura 47: Conexão e envio de mídia entre um ClientDevice do framework MMUD com um ServerDevice do framework MMND

4.2.2 MediaClient

O *MediaClient* é responsável pela recepção de uma mídia via UDP oriunda de um *MediaServer*. Ele é formado por seis classes (figura 48): três que definem interface, três que implementam estas interfaces. As interfaces definidas são: ***ClientDevice.java, MediaReceiver.java, MediaConsumer e PresenceSensor.java.***

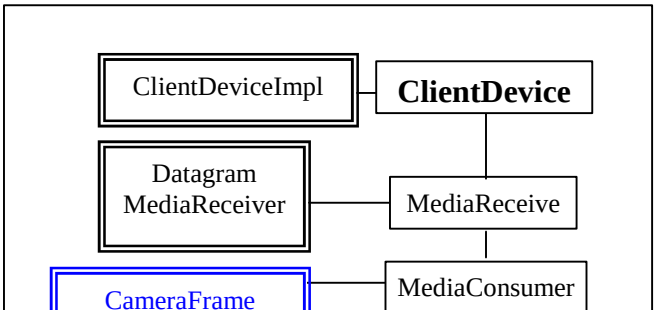

Figura 48: Classes relacionadas ao MediaClient (framework MMND)

A detecção da presença ou ausência da pessoa que assiste a mídia nos dispositivos referentes a plataforma J2ME ocorre de forma manual, ou seja, a pessoa que deverá com o toque no teclado do dispositivo, iniciar ou finalizar a transmissão de uma mídia.

ClientDevice.java*:* define a interface dos métodos (figura 49) que são implementados na classe *ClientDeviceImpl.java*. São eles: 1) iniciar recepção de mídia; 2) interromper recepção de mídia; 3) adicionar atributos das características do cliente; 4) remover atributos; 5) alterar atributos; 6)pausar recepções de mídias; 7) continuar recepções de mídias e 8) interromper todas as recepções de mídias.

A implementação dos três últimos métodos recebem parâmetros provindos da aplicação responsável pela detecção de mudança contextual

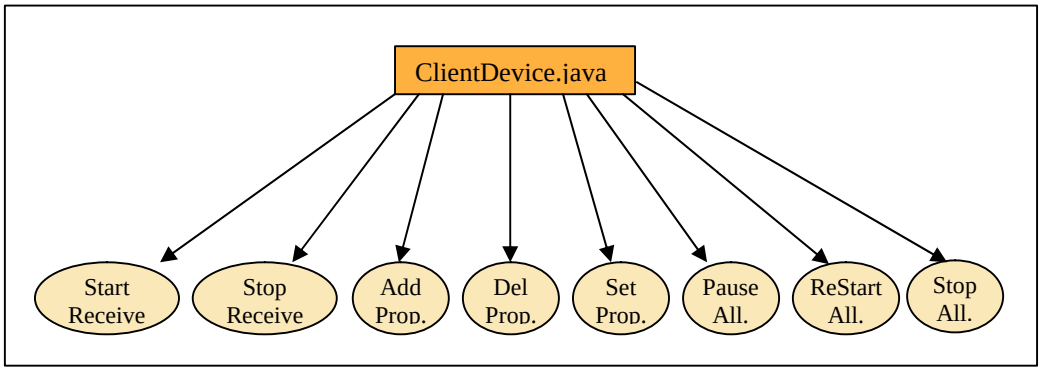

Figura 49: Interfaces definidas no Mediaclient (framework MMUD)

O objeto *ClientDevice* comunica com os objetos definidos no *framework* MMND criando um novo objeto *MediaManagerME_Stub*:

```
this.mm = new MediaManagerME_Stub(mediaManagerHost);
```

As interfaces ***MediaReceiver.java*** e ***MediaConsumer.java*** definem os mesmos métodos já definidos para o *framework* MMND.

5. Estudo de caso

O estudo de caso realizado foi baseado na seguinte possibilidade:

"Um sistema de vigilância possui várias câmeras espalhadas em um prédio. Cada câmera constitui um servidor (PC + câmera) e estará ligada a uma LAN/WLAN. Um servidor deve enviar suas imagens capturadas pela câmera para que pelo menos um funcionário/guarda visualize-as. Para isso é preciso que o sistema detecte uma mudança contextual de localização do funcionário, ou seja, detecte quando ele não estiver a frente do dispositivo o qual recebia a mídia, para que possa proporcionar o envio para um outro dispositivo ao alcance do mesmo. Por exemplo, ele pode sair da frente do PC de uma sala onde assistia as imagens capturadas, e continuar assistindo através do seu PDA, enquanto caminha para uma outra sala/banheiro."

Foram realizados testes com 3 modelos de *webcams* (*Creative* PD0040, *Creative* PD1001 e *QuicCam* I0) para verificar qual o formato da mídia a ser utilizado para a transmissão da mesma. Inicialmente optou-se por trabalhar com a mídia vídeo. Foram realizados testes para verificar os formatos de vídeos gerados com a captura e suas respectivas taxas de transmissão exigidas em KBps (*Kilobytes* por segundo).

Foi realizada uma amostra de vídeos a 5 fps (*frames* por segundo) capturadas em um intervalo de tempo de 10 segundos e tamanho 352 x 288 pixels. Para esta amostra foi utilizada a *webcam* Creative PD1001. Foram realizados testes com a Creative PD0040 e constatou-se que ela possui uma variação de até 60% nas taxas de amostragem comparando com o padrão *DirectShow* da Creative PD1001. Notou-se também que a qualidade da captura da PD0040 em relação a PD1001 é inferior.

É observado nesta amostra na tabela 2 que o formato de vídeo que possui uma menor taxa de transmissão (KBps – Kilobytes por segundo) é o DivX 5.05. Vale lembrar que o tamanho da tela do vídeo para completar esta tabela foi de 352 x 288 pixels, que é considerado um bom tamanho. Se utilizarmos o tamanho 160 X 120 pixels, teríamos uma taxa de transmissão reduzida em aproximadamente 3 vezes. Trabalharíamos então com uma taxa em torno de 20 KBps para uma amostragem de 5 fps para o formato DivX. Lembrando que uma conexão discada convencional via modem atinge no máximo 7 KBps, não poderia ser utilizado este formato para uma transmissão de vídeo.

Tamanho tela captura: 352 x 288 pixels Tempo: 10s		AVI			
		DIRECTSHOW		VFW	
FORMATO	**FPS**	**KB**	**KBps**	**KB**	**KBps**
DivX 5.05	5	580	**58**	760	76
ligos indeo r3.2	5	795	80	1080	108
indeo 5.11	5	1060	106	1418	141,8
indeo 5.10	5	1070	107	1560	156
ligos indeo 5.11	5	1125	113	1453	145,3
Cinepac	29	1500	150	2530	253
ligos indeo 4.5	5	1535	154	1900	190
microsoft rle	5	3015	302	4351	435,1
microsoft video1	5	4890	489	6240	624
iyuv intel	5	7065	707	10945	1094,5
sem formato	5	12560	1256	24465	2446,5
Mjpeg	29	14345	1435	21490	2149

Tabela 2: taxa em KBps exigidas por vídeos capturados pela webcam Creative PD1001

O formato de vídeo aceito em grande maioria dos celulares e PDAs atuais é o MPEG1. Como as *webcams* utilizadas não possuem este formato para captura, seria necessária uma adaptação de conteúdo capturado. Um vídeo no formato MPEG1 (figura 50) com uma tela de 160 x 120 pixels exige uma taxa de transmissão de no mínimo 110 KBps.

Figura 50: Imagem capturada de um vídeo convertido para o formato Mpeg1

O padrão utilizado para transmissão de dados via a tecnologia de celular GSM é o GPRS (*General Packet Radio Services*) que atualmente trabalha com uma transmissão de no máximo 44 kbps (Kilobits por segundo) ou 5,5 KBps (Kilobytes por segundo). Isto torna inviável o uso do formato MPEG1, que segundos testes exige 110 KBps

contra 5,5KBps suportado pela tecnologia GPRS atualmente. Optou-se por utilizar, para o estudo de caso, seqüências de imagens JPEG, minimizando as chances de incompatibilidade de padrões de mídias enviadas e recebidas pelo servidor e cliente da mesma, além de um enquadramento na taxa de transmissão aceita pelo dispositivo cliente.

Alguns dispositivos embarcados só aceitam o padrão de vídeo MPEG1, portanto, caso o servidor não trabalhe com este padrão, ele teria que encaminhar os pacotes oriundos da *webcam* conectada a ele, para um *proxy* que teria a função de adaptador de conteúdo.

O tratamento digital da imagem que é capturada por uma *webcam* pode se enquadrar em duas arquiteturas: *DirectShow* e VFW (*Vídeo for Windows*). A tecnologia *DirectShow* suporta uma grande quantidade de formatos de vídeo e áudio, tais como: ASF (*Advanced Streaming Format*), MPEG (*Montion Picture Experts Group*), AVI (*Audio-Video Interleaved*), etc. O *DirectShow* substitui o antigo VFW permitindo a criação de aplicações como DVD players, de edição de vídeo, conversores de AVI puro para ASF, MPEG, etc.

A tabela 3 mostra a foto de cada *webcam* utilizada, e o tamanho em kilobytes do arquivo .JPEG gerado a partir de uma captura de cada uma (*screenshot*), variando o tamanho da imagem capturada (160 x 120 *pixels* ou 352 x 288 *pixels*).

			
Tamanho	**Creative PD1001**	**Creative PD0040**	**QuicKCam I**
160 x 120	3,1KB	2,4KB	2KB
352 x 288	10,5KB	7,2KB	4,5KB

Tabela 3: As 3 webcams utilizadas no projeto e tamanho do "screenshot" de cada

Foram utilizados (tabela 4) os seguintes recursos disponíveis no laboratório GSDR da UFSCar:

- PC Athlon 1,43 – 256Ram (2 unidades)
- PC Athlon 1,5 – 256Ram (1 unidade)
- PC Athlon 2,2 – 256Ram (1 unidade)
- WebCam Creative Modelo PD0040 (2 unidades)
- WebCam Creative Modelo PD1001 (2 unidades)

- Recursos de LAN

Como observamos, foram utilizados quatro PCs (PC1, PC2, PC3 e PC4): O **PC1** ficou responsável somente pela execução da JAMP (PC1j).

ID	IP	Objeto	Dispositivo	Webcam	S.O.
PC1j	200.18.99.123	JAMP	PC	-	Linux
PC2m	200.18.99.113	MediaManager	PC	-	Linux
PC2s	200.18.99.113	MediaServer1	PC	QuickCam	Linux
PC3s	200.18.99.115	MediaServer2	PC	Creative	Linux
PC3c1	200.18.99.115	MediaClient1	PC	Creative	Linux
PC3c2	200.18.99.115	MediaClient2	Telefone Celular (simulador)	-	Linux
PC4c1	200.18.99.110	MediaClient3	PC	-	WinXP
PC4c2	200.18.99.110	MediaClient4	Telefone Celular (simulador)	-	WinXP

Tabela 4: Dispositivos usados para o estudo de caso

O **PC2** foi o responsável pelo gerenciamento dos serviços oferecidos para transmissão de mídia (MediaManager: PC2m). Estes serviços foram exportados para a JAMP (figura 51) como o MediaManager.

Figura 51: Media Manager no Broker View da plataforma JAMP

O PC2 representou também um servidor de mídia (MediaServer1: PC2s) com a função de capturar imagens de uma determinada WebCam e deixá-las disponíveis em um *buffer* de saída. O trecho principal do código executado pela aplicação presente em um servidor de mídia é:

```
a. String name = args[0];
b. String javabrokerHost = args[13];
c. MediaSenderFactory        mediaSenderFactory        =        new
   DatagramMediaSenderFactory();
d. MediaProducer           mediaProducer           =           new
   ExternalBlockMediaProducer("gqcam  -b  150  -w  140  -c  120  -v
   /dev/video1 -t JPEG -d -");
e. new   ServerDeviceImpl(name, javabrokerHost, mediaSenderFactory,
   mediaProducer);
```

O código presente em “d” resulta na criação de um objeto o qual recebe de uma aplicação externa (*gqcam* do S.O. Linux) *screenshots* (imagens capturadas seqüencialmente), e as deixa disponível para o envio em um *buffer*. Em “e” é passado o nome do servidor da mídia, para que seja cadastrado no MediaManager.

O **PC3** representou um outro servidor de mídia (MediaServer2: PC3s) com sua respectiva *WebCam.* Nele foram testados também o cliente PC (MediaClient1: PC3c1) e o cliente celular (MediaClient2: PC3c2), simulado pela ferramenta Wireless Toolkit 2.1.

Um cliente (figura 52 e 53) deve receber a mídia de um determinado servidor conectado no MediaManager. O trecho principal do código executado pela aplicação presente em um cliente (PC) de mídia é:

```
a. ClientDevice cd = new ClientDeviceImpl(args[0], args[13]);
b. MediaConsumer mc = new CameraConsumer();
c. MediaReceiver               mr               =               new
   DatagramMediaReceiver(Integer.parseInt(args[3]), mc);
d. ClientDeviceActivator cda = new ClientDeviceActivator(cd, new
                          i.  SCPresenceSensor("cam   --type   ppm",
                              500000));
e. while(true) {
f. cd.startReceiving(mr, args[2]);
   …
```

Em “a” o cliente passa os seus dados e os dados do servidor o qual deseja receber a mídia. Em “b” é criado um objeto para receber a mídia compatível com a mídia gerada pelo objeto *MediaProducer* criado no servidor da mídia. Já “c” representa uma *thread* responsável pela recepção dos pacotes de mídia enviados pelo objeto *MediaSender* criado no servidor da mídia.

Em “d” é iniciado o processo de detecção de presença do usuário. Neste momento é passado como parâmetro qual a *WebCam* responsável por este processo. O item “f” representa o início do processo de recepção. Este comando requisita do

ClientDevice do *framework*, que por sua vez requisita implicitamente do *MediaManager* o início da transmissão. O *MediaManager* requisita então do *ServerDevice* o início da transmissão.

Figura 52: Aplicação executando em um MediaClient (PC)

Os simuladores de telefone celular usados estavam trabalhando com o perfil MIDP 1.0 e configuração CLDC 1.0 da plataforma J2ME. Veja na figura 53 exemplos do simulador da ferramenta *Wireless Toolkit* executando *MediaClient´s* do *framework* MMUD.

Figura 53: Aplicações executando em MediaClient´s (simuladores de telefone celular)

O PC4 representou também dois clientes (figura 54) de mídia (PC e celular simulado). Nele foi testada a recepção de duas mídias em um mesmo instante, oriundas de dois servidores distintos.

Figura 54: Recepção de duas mídias em um PC, oriundas de dois servidores distintos

6. Resultados obtidos

Os *frameworks* MMND e MMUD, desenvolvidos e implementados no decorrer deste trabalho de mestrado, foram utilizados no desenvolvimento do estudo de caso, tendo fundamental importância para sua criação. As funcionalidades e abstrações contidas nos *frameworks* tornam a programação de sistemas como o apresentado no estudo de caso mais simples e flexível. Simples por possuir poucas linhas de código para a particularização das operações do sistema e flexível por permitir diversas mudanças, tais como protocolo de rede, hardware utilizado e política de transmissão, sem que haja a necessidade de modificar todo o código.

Para o estudo de caso, foi desenvolvido um sistema de segurança com câmeras, semelhante aos encontrados em prédios residenciais e comerciais. Para o projeto foram utilizadas *webcams* conectadas a PCs do laboratório GSDR do departamento de computação da UFSCar. Nessa situação, pode-se reproduzir uma condição de rede local entre os recursos, semelhante às condições encontradas em prédios residenciais.

O sistema desenvolvido nesse projeto de mestrado é capaz de detectar a ausência ou presença de um usuário em um determinado dispositivo através de sensores. O sistema proporciona também o envio de uma mídia para diversificados dispositivos computacionais, como por exemplo dispositivos embarcados, de forma transparente e com mobilidade, possibilitando assim, o enquadramento do mesmo, como um exemplo de computação ubíqua.

Para a implementação do sistema proposto, constatou-se que o envio de seqüências de imagens, pelo servidor a um cliente, pode substituir de maneira aceitável *streams* de vídeo em muitas aplicações. Um dos favorecedores dessa opção adotada é que foi exigida uma largura de banda de aproximadamente 7,2 KBps (*Kilobytes* por segundo) para o envio de uma seqüência de imagens a um dispositivo cliente. Isso porque a taxa de *frames* (imagens) por segundo foi de 3 fps, e cada imagem possuía um tamanho de arquivo fixo de 2,4KB. Esses valores referem-se a imagens capturadas respeitando a proporção de 160 x 120 *pixels* à partir de uma *webcam Creative* PD0040. Em um dos testes realizados, foi relatado que dentre 10.000 imagens enviadas pelo servidor, 9.996 foram recebidas pelo dispositivo do cliente e 9.994 foram visualizadas na tela do mesmo (renderizadas).

Em uma situação semelhante de envio de um vídeo, caso fosse utilizado o padrão de vídeo MPEG1, aceito em grande parte dos celulares e PDAs atuais, para uma

taxa de 5 fps seria necessária uma largura de banda de aproximadamente 110KBps. Observe que este valor ultrapassa a marca de 15 vezes o valor exigido ao utilizar uma seqüência de imagens no formato JPEG.

Foram encontrados porém alguns problemas na criação do ambiente de teste para o projeto desenvolvido, podendo-se destacar dois deles. O primeiro foi detectado quando se tentou aproveitar o máximo de máquinas disponíveis no laboratório, rodando a aplicação cliente em uma máquina SUN com o sistema operacional Linux e processador SPARC. Devido a versão *Java* disponível para este S.O ser desatualizada, apenas máquina virtual com jre 1.3.1, há classes de manipulação gráfica que diferem da versão utilizada, j2sdk1.4.1_03. Dessa forma, não foi possível visualizar as imagens recebidas nas máquinas com esta configuração, na aplicação do estudo de caso. Já o segundo problema detectado ocorreu quando se notou que o "*garbage colector*" do *Java* não estava executando a devida limpeza, o que ocasionava um aumento contínuo de uso de memória no dispositivo cliente da mídia. Ao tentar resolver esse problema, foram utilizados dois mecanismos diferentes de manipulação de imagens compactadas JPEG, os quais obtiveram diferentes desempenhos. O mecanismo que melhor se adequou à situação, utiliza uma biblioteca específica da máquina Virtual SUN para manipular imagens sem utilizar arquivos.

Apesar desses pequenos problemas, o objetivo de se criar uma arquitetura comum de comunicação que permitisse a recepção de mídias em diferentes dispositivos foi concluída de forma satisfatória.

A idéia de que o usuário continuasse a assistir a um vídeo em um dispositivo móvel quando saísse do local onde estava assistindo-no também foi concretizada. Entretanto, nos dispositivos móveis não foi implementado nenhum sensor para que fosse detectada implicitamente a presença ou ausência do usuário diante dele. Nesse caso o usuário teria que requisitar explicitamente no dispositivo móvel o início da transmissão da mídia. Essa escolha não só foi feita pela dificuldade de integração de dispositivos móveis com dispositivos sensores, mas também se baseando no fato de que dispositivos dessa categoria costumam ser usados apenas em situações restritas.

Todavia, caso um cliente do sistema esteja visualizando a mídia em um PC e esse possuir uma *webcam*, ela poderá ser utilizada como detector de presença/ausência. Nesse caso, quando o usuário sair da frente do PC, será detectada sua ausência, e o sistema interromperá a transmissão para aquele PC. Caso o usuário caminhe para uma outra sala que contenha um outro PC com *webcam* e esse estiver conectado ao sistema,

será detectada a sua presença, e iniciar-se-á a transmissão da mídia para este último PC. Este situação foi testada e validada de forma satisfatória.

Essa detecção de presença poderia ocorrer com o uso de um dispositivo fotossensor ou um sensor eletromagnético, freqüentemente utilizados em detecção de movimentos em alarmes residenciais. Para isso deve-se substituir as classes *ClientDeviceActivator* e *PresenceSensor* do *framework* MMND por uma classe que controle os sinais oriundos desses dispositivos, e que estariam conectados a uma porta de comunicação (entrada serial/paralela/usb) do computador.

A escolha pela implementação de um *framework* obedecendo a padrões de programação cria uma grande diversidade de acréscimos e modificações no mesmo, deixando-o rico em contribuições.

Um dos possíveis trabalhos futuros para esse projeto, dar-se-á por meio de um estudo sobre a tecnologia WCORBA - *Wireless* CORBA da OMG, substituindo a tecnologia JAVA-RMI utilizada. Isto porque em uma rede sem fio com mobilidade o dispositivo do usuário pode trocar de endereço IP a medida que faz *handoffs* entre diferentes redes, e o JAVA-RMI não prevê esta particularidade. Isso implicaria em uma implementação de um novo Broker, uma vez que o JBROKER utilizado da Jamp utiliza a tecnologia JAVA-RMI.

Um outro trabalho futuro sugerido, seria a implementação de um adaptador de conteúdo para ser usado em conjunto ao *MediaManager* e os servidores de mídia. Com isso poderia existir uma situação na qual um dispositivo embarcado requisita uma mídia em que o sistema detecte que não poderá ser visualizada no *display* do cliente, seja por limitação de tamanho do *display*, ou por limitações de recursos de *software* e *hardware*. Essa mídia seria encaminhada então para o adaptador de conteúdo, para depois ser enviada ao cliente.

7. Referências Bibliográficas

[HRef-1] http://www.palmbrasil.com.br/conheca-palm/index.html, pesquisa realizada em 03/2004.

[HRef-2] http://www.wirelessbrasil.org, pesquisa realizada em 03/2004.

[HRef-3] JVC-R Home Page - http://www.mcrlab.uottawa.ca/index.html, pesquisa realizada em janeiro de 2003.

[HRef-4] VideoLAN HomePage – http://www.videolan.org, pesquisa realizada em novembro de 2003.

[HRef-5] Microsoft, Inc. "Windows Media Technologies" - http://www.microsoft.com/windowsmedia/ , pesquisa realizada em janeiro de 2004.

[HRef-6] RealnetWorks. "RealServer Administration guide RealSystem G2" - http://www.real.com/serveradminguide2.pdf , pesquisa realizada em novembro de 2003.

[HRef-7] RealnetWorks. "RealServer Administration guide RealSystem G2" - http://www-306.ibm.com/*software*/data/videocharger/vcserverkey.html, pesquisa realizada em dezembro de 2003.

[HRef-8] Home Page Mark Weiser - http://www.ubiq.com/weiser/ - site visitado em novembro de 2003.

[Href-9] Home Page – Aware Home - http://www.awarehome.gatech.edu/projects /index.html , pesquisa feita em 04/2003

[HRef-10] Sun – Especificação: CDC - http://java.sun.com/products/cdc/ - pesquisa feita em 10/2003

[HRef-11] Sun – Especificação: CLDC - http://java.sun.com/products/cldc/ - pesquisa feita em 10/2003

[HRef-12] WirelessBR – Rosa, Hélio F. - Histórico do Sistema Móvel Celular - A Evolução Nas Comunicações Celulares - http://www.wirelessbrasil.org /wirelessbr/secoes/sec_telefonia.html - pesquisa realizada em fevereiro de 2004

[HRef-13] Tecnologia sem fio - http://www.anydatadobrasil.com.br/technology/ gsm.html, pesquisa realizada em fevereiro de 2003.

[HRef-14] HTTP IEEE - http://standards.ieee.org/getieee802/802.11.html - pesquisa realizada fevereiro de 2004.

[HRef-15] *Java*Spaces Services Specifications - http://wwws.sun.com/*software*/jini/ specs/jini1.1html/jsTOC.html - pesquisa realizada em janeiro de 2004.

[HRef-16] *Gaia* Active Spaces for Ubiquitous Computing - http://choices.cs.uiuc.edu/ gaia/index.html - pesquisa realizada em maio de 2004.

[HRef-17] The Motion Book – Allen Bradley – http://www.ab.com/sensors - pesquisa realizada em agosto de 2004.

[HRef-18] M2SAT – Detectores de movimentos por RF – http://www.m2sat.com.br/detec_micro.htm - pesquisa realizada em agosto de 2004.

---

[1] Huand, Polly; Lenders, Vicent; Minnig, Philipp; Widmer, Mario, - Jini for Ubiquitous Devices – Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology – Zurich, June 2002

[2] Banavar, Guruduth; Bernstein, Abraham, - Software Infraestructure and Design Challenges for Ubiquitous Computing Application – Communication of the ACM, December 2002.

[3] Arrowood, A. - "CU-SeeMe Communications in an Emergent Technology" - LCC/IDT, OIT/NS GRA, Georgia Institute of Technology, February 1996

[4] ROESLER, Valter; ANDRADE, Maiko de; "On-line remote class with live video transmission: a study case". Computers and Advanced Technology in Education Conference / Web Based Education - CATE/WBE 2003. Rhodes.

[5] França, Lúcio A.; Couto, Rainer R.P.; Loureiro, Antonio A.F. – Adaptabilidade de Compressão em Ambientes Móveis: Como e Quando comprimir – WCSF2001, Dep. Comp. UFMG, 2001.

[6] Jauane, C. "TVS – Um Sistema de Videoconferência" – Dissertação de Mestrado – Departamento de Computação – PUC/RJ, 1996.

[7] Soares, L.F.G.; Martins, S. de L.; Bastos, T.L.P. - "Lan Based Real Time Audio-Graphics Conferencing System, General Overview" - CCR066 Technical Report Rio Scientific Center-IBM Brasil, Novembro de 1988.

[8] Designing Internetwork for Multimedia – http://www.cisco.com/univercd/cc/td/doc/cisintwk/idg4/nd2013.htm#xtocid0; consultado em janeiro de 2003

[9] Vasconcellos, Saulo Vaz, - Videonconferência: Uma visão geral – GTA – COPPE/PEE, 1999. Disponível em 01/2003 no site: http://www.gta.ufrj.br/~vaz/research/vdcnf/videoconf1.html

[10] Hummelgen, F.L.; Fonseca, K. V. O. – Redes TCP/IP Integradas a Garantia de Qualidade de Serviço em Aplicações Multimídia – RNP, CELEPAR e CPGEI – CEFET/PR, 2000.

[11] Oliveira, S. Rômulo; Furtado, J.V. Olinto – "Um mecanismo de adaptação para aplicações tempo real baseado em computação imprecisa e reflexão computacional" – artigo SBES, 2002.

[12] Weiser, Mark; Brow, J. S – The Coming Age of Calm Techonoly – XEROX PARC, October 1996.

[13] Abowd, Gregory D., Elizabeth D. – Charting Past, Present, and Future Research in Ubiquitous Computing - Georgia Institute Tecnology, 2000.

[14] Lyytinen, Kalle; Yoo, Youngjin, - Issues and Challenges in Ubiquitous Computing – Case Western Reserve University in Cleveland, OHIO, ACM, 2002.

[15] Igushi, Fábio; Wills, Willian – Pervasive Computing - http://www.gta.ufrj.br /~rezende/cursos/eel879/trabalhos/ubiquitous/, pesquisa feita em janeiro de 2003

[16] Dey, Anind K., - Understanding and Using Context - Georgia Institute Tecnology, Future Computing Environment Group, 2000.

[17] Hairong Qi, S. Sitharama Iyengar, Krishnendu Chakrabarty, "Distributed sensor networks - a review of recent research", Journal of The Franklin Institute, PERGAMON, 2001, págs. 655 a 668.

[18] Fresco, Nedim; - The CVM Virtual Machine: A *Java* Techonoly-based VM for Consumer Eletronics – Staff Engineer, Sum Microsystem, WJDC, 2001.

[19] Lamont, Adams - Exploring J2ME - Builder.Com, March, 2003.

[20] Deitel, H. M.; Deitel, P. J., - *Java* Como Programar – 3ª edição, Bookman, 2001

[21] Trevelin, L. C. & Ferreira,M. M. - The *Java* Broker System: Concepts &*Java* Programming Guide - Technical Report.DC,UFSCar,1998.

[22] Tanenbaum, Andrew S. Redes De Computadores 3ª edição ed.Campus,1997

[23] Baptista, Bruno A. D. - Projeto do Subsistema de Comunicação e Distribuição e da Camada de Serviços da Arquitetura OpenReality para suporte à criação de Aplicações de Visualização Distribuída. – Dissertação de mestrado – UFSCar – fevereiro 2004

[24] MENDONCA, P. S. et al. – Development of object oriented distributed systems (DOODS) using *framework*s of the JAMP platform. – ICSE´99, Los Angeles, USA, mai. 1999

[25] Ferreira,M. M. & Trevelin, L. C. A Platform for Developing Distributed Multimedia Application. . Techinical Report.DC,UFSCar,1998.

[26] – Gonçalves, Kelli - Enquanto se espera a tecnologia 3G – artigo escrito 10/06/2002 - http://pcworld.terra.com.br/pcw/testes/tecno_hard/0069.html, pesquisa em 05/2003

[27] Luiz Cláudio Rosa e José Mário Bertolini Serra, (*)in "Sistemas Móveis de Terceira Geração", GuidelineBRISA - Agosto/2002.

[28] Technical Specification Group Service and system Aspects – “Packet Switched Streaming Services – MMS (Multimedia Messaging Service), December 2002.

[29] Ansi/IEEE – Wireless LAN Médium Access Control (MAC) and Physical Layer (PHY) Specification – 1999 Edition.

[30] IEEE Std 802.16 – IEEE Standard for Local and Metropolitan Area Networks – 2001 Edition.

[31] Thomas G. Xydis Ph.D; Simon Blake-Wilson - Security Comparison: Bluetooh Communications vs. 802.11 - Bluetooth Security Experts Group, 2002

[32] Souza, Wanderley, Lopes – Adaptação de conteúdo em perfis de dispositivos, conteúdo, usuário e serviço de rede; SBRC2002 – 20° Simpósio Brasileiro de Redes de Computadores; Maio 2002.

[33] Abowd, Gregory D; Atkeson, C.G.; Hong, Jason; Long, Sue; Kooper, Rob; Pinkerton, Mike – Cyberguide, A Mobile Context-Aware Tour Guide - Georgia Institute Tecnology, 1996.

[34] Burrel, J; Gay, G; Kupo, K; Fa, N – Context-aware computing: A test case. In Proceedings of the Internacional Conference on Ubiquitous Computing – October 2002.

[35] Brotherton, J.A.; Abowd, G.D.; Troung, K.N. – Suporting Captured and Access Interfaces for Informal and Opportunistic Meetings – GVU Center, Georgia Institute of Techonoly, January 1999.

[36] Jardim, Fernando – "*Framework*s MMND - Media Manager Normal Device e MMUD - Media Manager Ubiquitous Device" – Relatório técnico, disponível em http://www.gsdr.dc.ufscar.br/~fmj/download/relatorio2004.pdf, junho 2004.

ANEXO I

Classes e Interfaces do *Framework* MMND

MediaManager (MMND) – Inteface = MediaManager.java

```
package br.ufscar.dc.fmj.framework;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.*;
import javabroker.*;
import java.util.Enumeration;
public interface MediaManager extends Tradeable {
  public void addServer( String serverID, ServerDevice server ) throws RemoteException;
  public void removeServer( String serverID ) throws RemoteException;
  public Enumeration getServers( ) throws RemoteException;
  public boolean startReceiving( String serverID, ClientDevice client ) throws RemoteException;
  public boolean stopReceiving( String serverID, ClientDevice client ) throws RemoteException;
  public ServerDevice getServer( String serverID ) throws RemoteException;
}
```

MediaManager (MMND) – Implementação = MediaManagerImpl.java

```
package br.ufscar.dc.fmj.framework;
import java.rmi.server.UnicastRemoteObject;
import javabroker.*;
import java.lang.*;
import java.util.*;
import java.net.*;
import java.rmi.*;
import java.io.*;
public class MediaManagerImpl extends UnicastRemoteObject implements MediaManager {
  protected JBSession session;
  protected Hashtable mediaServers = new Hashtable();
  public static final String about = "MediaManager - JBroker System\n\n" +
                                     "Universidade Federal de São Carlos - UFSCar\n\n" +
                                     "Authors:\n" +
                                     "       Marcelo M. Ferreira            e-mail: ferreira@dc.ufscar.br\n" +
                                     "       Luis C. Trevelin               e-mail: trevelin@dc.ufscar.br\n\n" +
                                     "       Fernando de Moraes Jardim      e-mail: fmj@dc.ufscar.br\n\n" +
                                     "       Bruno do Amaral Dias Baptista  e-mail: bruno@dc.ufscar.br\n\n";

  public static final String syntax = "MediaManager\n\n" +
                                      "The correct syntax is :\n" +
                                      "       java MediaManager brokerHost\n\n";


  /** Creates a new instance of MediaManager */
  public MediaManagerImpl( String javabrokerHost ) throws RemoteException {
    System.setSecurityManager( new RMISecurityManager() );
    this.session = new JBSession( javabrokerHost );
    try {
      this.session.broker.export("MediaManager", this);
    } catch ( java.rmi.ServerException e ) {
      this.session.broker.remove("MediaManager");
      this.session.broker.export("MediaManager", this);
      System.out.println( "Exportado!" );
    }
  }

  public String aboutServer() throws RemoteException {
    return about;
```

```
  }
  public synchronized void addServer( String serverID, ServerDevice server ) throws RemoteException {
    this.mediaServers.put( serverID, server );
    System.out.println( "Media Server[" + serverID + "] add to the manager.");
  }

  public synchronized void removeServer( String serverID ) throws RemoteException {
    this.mediaServers.remove( serverID );
    System.out.println( "Media Server[" + serverID + "] removed from the manager." );
  }

  public synchronized ServerDevice getServer(String serverID) throws RemoteException {
    System.out.println( "getServer" );
    // testar o servidor antes de retorná-lo
    return (ServerDevice)this.mediaServers.get(serverID);
  }
  public synchronized Enumeration getServers() throws RemoteException {
    return this.mediaServers.elements();
  }

  public synchronized boolean startReceiving( String serverID, ClientDevice client ) throws
RemoteException {
    System.out.println( "startReceiving - " + client.getProperty("address"));
    ServerDevice server = (ServerDevice)this.mediaServers.get( serverID );
    if (server == null) return false;
    server.startSend(client);
    return true;
  }
  public synchronized boolean stopReceiving( String serverID, ClientDevice client ) throws
RemoteException {
    System.out.println( "stopReceiving" );
    ServerDevice server = (ServerDevice)this.mediaServers.get( serverID );
    if (server == null) return false;
    server.stopSend(client);
    return true;
  }

  public static void main( String args[] ) {
    if( args.length == 0 ) {
      System.out.println( syntax );
      return;
    }

    try {
      MediaManagerImpl mm = new MediaManagerImpl( args[ 0 ] );
      new MediaManagerME_Skel(mm, 5678);
    } catch( Exception e ) {
      e.printStackTrace();
      return;
    }
    System.out.println( "MediaManager ready!" );
  }
}
```

MediaServer (MMND) – Inteface = ServerDevice.java

```
package br.ufscar.dc.fmj.framework;
public interface ServerDevice extends java.rmi.Remote {
  public boolean isRunning() throws java.rmi.RemoteException;
  public void startSend(ClientDevice client) throws java.rmi.RemoteException;
  public void stopSend(ClientDevice client) throws java.rmi.RemoteException;
  public Object addProperty(String name, Object property) throws java.rmi.RemoteException;
  public Object removeProperty(String name) throws java.rmi.RemoteException;
  public Object getProperty(String name) throws java.rmi.RemoteException;
}
```

MediaServer (MMND) – Implementação = ServerDeviceImpl.java

```
package br.ufscar.dc.fmj.framework;
import java.rmi.server.UnicastRemoteObject;
import java.util.Hashtable;
import java.net.InetAddress;
import javabroker.*;
public class ServerDeviceImpl extends UnicastRemoteObject implements ServerDevice {
   private Hashtable properties = new Hashtable();
  private Hashtable senders = new Hashtable();
  private MediaSenderFactory msf;
  private MediaProducer mp;
   public ServerDeviceImpl(String name, String javabrokerHost, MediaSenderFactory
mediaSenderFactory, MediaProducer mediaProducer) throws javabroker.JBException,
java.rmi.RemoteException {
    JBSession session = new JBSession(javabrokerHost);
    MediaManager mm = (MediaManager)session.broker.invoke("MediaManager");
    mm.addServer(name, this);
    this.msf = mediaSenderFactory;
    this.mp = mediaProducer;
  }
    public boolean isRunning() throws java.rmi.RemoteException {
    return true;
  }
    public void startSend(ClientDevice client) throws java.rmi.RemoteException {
    System.out.println( "Vou mandar para " + client.getProperty("name") );
      senders.put(client, msf.createMediaSender(client, mp));
  }
  public void stopSend(ClientDevice client) throws java.rmi.RemoteException {
    System.out.println( "Vou parar de mandar para " + client.getProperty("name") );
    MediaSender ms = (MediaSender)senders.get(client);
    if (ms == null) System.out.println("É NULL");
    ms.interrupt();
  }
  public Object addProperty(String name, Object property) throws java.rmi.RemoteException {
    return properties.put(name, property);
  }
  public Object removeProperty(String name) throws java.rmi.RemoteException {
    return properties.remove(name);
  }
  public Object getProperty(String name) throws java.rmi.RemoteException {
    return properties.get(name);
  }
}
```

MediaSender (MMND) – Interface = MediaSender.java

```
package br.ufscar.dc.fmj.framework;
import java.net.*;
public interface MediaSender extends Runnable {
 public InetAddress getAddress();
 public int getPort();
 public void interrupt();
}
```

MediaSender (MMND) – Implementação = DatagramMediaSender.java

```
package br.ufscar.dc.fmj.framework;
import java.net.*;
public class DatagramMediaSender implements MediaSender {
  private DatagramSocket ds;
  private DatagramPacket dp;
  private Thread thread;
  private MediaProducer mediaProducer;
  private byte[] value = new byte[65000];
  private int counter = 0;
  public DatagramMediaSender(InetAddress destAddress, int destPort, MediaProducer mediaProducer)
throws SocketException {
    this(new DatagramSocket(), destAddress, destPort, mediaProducer);
  }
  public DatagramMediaSender(DatagramSocket ds, InetAddress destAddress, int destPort,
MediaProducer mediaProducer) {
    this.ds = ds ;
    this.mediaProducer = mediaProducer;
    this.dp = new DatagramPacket(value, value.length, destAddress, destPort) ;
    this.thread = new Thread(this);
    this.thread.setDaemon(true);
    this.thread.start();
  }
  public InetAddress getAddress() {
    return ds.getInetAddress();
  }
  public int getPort() {
    return ds.getPort();
  }
  public void interrupt() {
    thread.interrupt();
  }
public void run() {
    try{
      int c = 0;
      while (true) {
        try {
          System.out.print("*");
          c = mediaProducer.produce(value, 0, value.length);
          if ( c>=0 ) {
            dp.setLength(c);
            counter++;
            ds.send(dp);
          }
        } catch (java.io.IOException error) {
          error.printStackTrace();
        }
        Thread.sleep(1);
      }
    } catch (InterruptedException error) {
```

```
      System.out.print("Enviou " + counter + " frames!");
    }
  }
}
```

ClientDevice (MMND) - Interface: ClientDevice.java

```
package br.ufscar.dc.fmj.framework;
public interface ClientDevice extends java.io.Serializable {
  public boolean startReceiving(String serverID, MediaReceiver mr);
  public boolean stopReceiving(String serverID);
  public void pauseAll();
  public void continueAll();
  public void stopAll();
  public void close();
  public Object addProperty(String name, Object property) throws java.rmi.RemoteException;
  public Object removeProperty(String name) throws java.rmi.RemoteException;
  public Object getProperty(String name) throws java.rmi.RemoteException;
}
```

ClientDevice (MMND) - Implementação: ClientDeviceImpl.java

```
package br.ufscar.dc.fmj.framework;
import br.ufscar.dc.fmj.framework.*;
import java.rmi.server.UnicastRemoteObject;
import javabroker.*;
import java.util.Hashtable;
import java.util.Enumeration;
import br.ufscar.dc.fmj.util.*;

public class ClientDeviceImpl implements ClientDevice, br.ufscar.dc.fmj.util.Serializable {
   private Hashtable properties = new Hashtable();
  private MediaManager mm;
  private boolean paused = false;
  private Hashtable media = new Hashtable();
    public ClientDeviceImpl() {
   // Just to DeSerialize from Micro Edition version
  }

  public ClientDeviceImpl(String name, String javabrokerHost) throws javabroker.JBException,
java.rmi.RemoteException {
    properties.put("name", name);
    JBSession session = new JBSession(javabrokerHost);
    this.mm = (MediaManager)session.broker.invoke("MediaManager");
  }

  public boolean startReceiving(String serverID, MediaReceiver mr) {
    try {
      this.properties.put("address", mr.getAddress());
      this.properties.put("port", new Integer(mr.getPort()));
      if (mm.startReceiving(serverID, this)) {
        this.media.put(serverID, mr);
        return true;
      }
    } catch(Exception e){ e.printStackTrace(); }
    return false;
  }

  public boolean stopReceiving(String serverID) {
    try {
      if (mm.stopReceiving(serverID, this)) {
        this.media.remove(serverID);
        return true;
      }
    } catch(Exception e){ e.printStackTrace(); }
    return false;
  }
```

```
public void pauseAll() {
  if (!paused) {
    for ( Enumeration l = media.keys() ; l.hasMoreElements() ;) {
      String s = (String)l.nextElement();
      try {
        mm.stopReceiving(s, this);
      } catch(Exception e){ e.printStackTrace(); }
    }
    paused = true;
  }
}

public void continueAll() {
  if (paused) {
    for ( Enumeration l = media.keys() ; l.hasMoreElements() ;) {
      String s = (String)l.nextElement();
      MediaReceiver mr = (MediaReceiver)media.get(s);
      try {
        startReceiving(s, mr);
      } catch(Exception e){ e.printStackTrace(); }
    }
    paused = false;
  }
}

public void stopAll() {
  for ( Enumeration l = media.keys() ; l.hasMoreElements() ;) {
    String s = (String)l.nextElement();
    try {
      stopReceiving(s);
    } catch(Exception e){ e.printStackTrace(); }
  }
}

public void close() {
  stopAll();
}

public Object addProperty(String name, Object property) throws java.rmi.RemoteException {
  return properties.put(name, property);
}


public Object removeProperty(String name) throws java.rmi.RemoteException {
  return properties.remove(name);
}

public Object getProperty(String name) throws java.rmi.RemoteException {
  return properties.get(name);
}

 public int hashCode(){
  return properties.get("name").hashCode();
}

public boolean equals(Object o) {
  return this.toString().equals(o.toString());
}
```

```
  public void writeObject(ObjectOutputStream oos) throws java.io.IOException {
    oos.writeInt(properties.size());
    Enumeration keys = properties.keys();
    while ( keys.hasMoreElements() ) {
      String tk = (String)keys.nextElement();
      String tv = (String)properties.get(tk);
      oos.writeUTF(tk);
      oos.writeUTF(tv);
    }
  }

  public void readObject(ObjectInputStream ois) throws java.io.IOException {
    int total = ois.readInt();
    for (int i=0 ; i<total ; i++) {
      String tk = ois.readUTF();
      String tv = ois.readUTF();
      properties.put(tk, tv);
    }
  }

  private void writeObject(java.io.ObjectOutputStream out) throws java.io.IOException {
    out.writeObject(properties);
  }

  private void readObject(java.io.ObjectInputStream in) throws java.io.IOException,
ClassNotFoundException {
    properties = (Hashtable)in.readObject();
  }

}
```

MediaReceiver (MMND) - Interface: MediaReceiver.java

```
package br.ufscar.dc.fmj.framework;
import java.net.*;
public interface MediaReceiver extends Runnable {
public InetAddress getAddress();
  public int getPort();
  public void close();
}
```

MediaReceiver (MMND) – Implementação: DatagramMediaReceiver.java

```
package br.ufscar.dc.fmj.framework;
import java.net.*;
public class DatagramMediaReceiver implements MediaReceiver {
  private DatagramSocket ds;
  private DatagramPacket dp;
  private Thread thread;
  private MediaConsumer mediaConsumer;
  private byte[] value = new byte[65000];
  public DatagramMediaReceiver(int port, MediaConsumer mediaConsumer) throws
java.net.SocketException {
    this.ds = new DatagramSocket(port);
    this.mediaConsumer = mediaConsumer;
    this.dp = new DatagramPacket(value, value.length);
    this.thread = new Thread(this, this.getClass().getName());
    this.thread.setDaemon(true);
    this.thread.start();
```

```
  }

public InetAddress getAddress() {
    InetAddress ia = null;
    try{
      ia = InetAddress.getLocalHost();
    } catch(Exception e) { e.printStackTrace(); }
    return ia;
  }

  public int getPort() {
    return ds.getLocalPort();
  }

 public void close() {
    this.thread.interrupt();
    this.ds.close();
  }

public void run() {
    try {
      while (true) {
        ds.receive(dp) ;
        System.out.print(".");
        mediaConsumer.consume(dp.getData(), 0, dp.getLength());
      }
    } catch(Exception e) {e.printStackTrace();}
  }
}
```

ClientDeviceActivator (MMND) - Implementação: ClientDeviceActivator.java

```
package br.ufscar.dc.fmj.framework;
public class ClientDeviceActivator implements Runnable {
  private int arriveCountThreshold = 2;
  private long diffThreshold = 1800000;
  private long leaveTimeThreshold = 20000;
  private long sleeptime = 1000;
  private Thread thread;
  private PresenceSensor presenceSensor;
  private ClientDevice clientDevice;
  private boolean isPresent = false;
  private int threshold = arriveCountThreshold;
    public ClientDeviceActivator(ClientDevice clientDevice, PresenceSensor presenceSensor) {
    this.clientDevice = clientDevice;
    this.presenceSensor = presenceSensor;
    this.thread = new Thread(this, this.getClass().getName());
    this.thread.setDaemon(true);
    this.thread.start();
  }
  protected void leave() {
    clientDevice.pauseAll();
  }
  protected void arrive() {
    clientDevice.continueAll();
  }
public void run() {
    try {
      long diff = 0, captime=0, motiontime=0;
      while (true) {
```

```
      captime = System.currentTimeMillis();
      diff = presenceSensor.getDifference();
      if ( diff > diffThreshold ) {
        motiontime = captime;
        if (!isPresent && threshold>0)
          threshold--;
      } else if ( threshold < arriveCountThreshold )
        threshold++;

      if ( !isPresent && threshold == 0 ) {
        isPresent = true;
        this.arrive();
      }
      if ( isPresent && threshold == arriveCountThreshold && captime-motiontime >
leaveTimeThreshold) {
        isPresent = false;
        this.leave();
      }
      Thread.sleep(sleeptime);
    }
   } catch(Exception e) {e.printStackTrace();}
  }
}
```

PresenceSensor (MMND) – Inteface: PresenceSensor.*Java*

```
package br.ufscar.dc.fmj.framework;
public interface PresenceSensor {
  public long getDifference() ;
}
```

PresenceSensor (MMND) – Implementação: SCPresenceSensor.*Java*

```
package br.ufscar.dc.fmj.framework;
import java.io.InputStream;
public class SCPresenceSensor implements PresenceSensor {
  private int[] buffer;
  private String command;
  public SCPresenceSensor(String command, int bufferSize) {
    this.buffer = new int[bufferSize];
    this.command = command;
  }
  public long getDifference() {
    try {
      Process cam1 = Runtime.getRuntime().exec(command);
      InputStream is1 = cam1.getInputStream();
      long diff = 0;
      int c = 0, i = 0;
      while ( (c = is1.read()) >= 0) {
        diff += Math.abs(buffer[i] - c);
        buffer[i++] = c;
      }
      cam1.waitFor();
      return diff;
    } catch(Exception e) {
      e.printStackTrace();
      return -1;
    }
  }
}
```

ANEXO II

Classes e Interfaces do *Framework* MMUD

MediaManagerME (MMUD) – Inteface = MediaManagerMe.java

```
package br.ufscar.dc.fmj.framework;
import java.util.Enumeration;
public interface MediaManagerME {
  public boolean startReceiving( String serverID, ClientDevice client );
  public boolean stopReceiving( String serverID, ClientDevice client );
}
```

MediaManagerME (MMUD) – Implementação = MediaManagerME_Stub.java

```
package br.ufscar.dc.fmj.framework;
import javax.microedition.io.*;
import br.ufscar.dc.fmj.util.*;
public class MediaManagerME_Stub implements MediaManagerME {
  private ObjectInputStream dis;
  private ObjectOutputStream dos;
  private StreamConnection sc;
  public MediaManagerME_Stub( String mediaManagerHost ) throws java.io.IOException {
    sc = (StreamConnection) Connector.open("socket://" + mediaManagerHost);
    dis = new ObjectInputStream(sc.openDataInputStream());
    dos = new ObjectOutputStream(sc.openDataOutputStream());
  }
  public synchronized boolean startReceiving( String serverID, ClientDevice client ) {
    System.out.println( "startReceiving" );
    try {
      dos.writeUTF("startReceiving");
      dos.writeUTF(serverID);
      dos.writeObject(client);
      dos.flush();
      return true;
    } catch(java.io.IOException error) { return false; }
  }
  public synchronized boolean stopReceiving( String serverID, ClientDevice client ) {
    System.out.println( "stopReceiving" );
    try {
      dos.writeUTF("stopReceiving");
      dos.writeUTF(serverID);
      dos.writeObject(client);
      dos.flush();
      return true;
    } catch(java.io.IOException error) { return false; }
  }
  public void close() throws java.io.IOException {
    dis.close();
    dos.close();
    sc.close();
  }
}
```

ClientDevice (MMUD) - Interface: ClientDevice.java

```
package br.ufscar.dc.fmj.framework;
public interface ClientDevice extends br.ufscar.dc.fmj.util.Serializable {
  public boolean startReceiving(String serverID, MediaReceiver mr);
  public boolean stopReceiving(String serverID);
  public void pauseAll();
  public void continueAll();
  public void stopAll();
  public void close();
  public Object addProperty(String name, String property);
  public Object removeProperty(String name);
  public Object getProperty(String name);
}
```

ClientDevice (MMUD) – Implementação: ClientDeviceImpl.java

```
package br.ufscar.dc.fmj.framework;
import java.util.Hashtable;
import java.util.Enumeration;
import br.ufscar.dc.fmj.util.*;
public class ClientDeviceImpl implements ClientDevice {
  private Hashtable properties = new Hashtable();
  private MediaManagerME mm;
  private boolean paused = false;
  private Hashtable media = new Hashtable();
  public ClientDeviceImpl(String name, String mediaManagerHost) throws java.io.IOException {
    properties.put("name", name);
    this.mm = new MediaManagerME_Stub(mediaManagerHost);
  }
  public boolean startReceiving(String serverID, MediaReceiver mr) {
    try {
      this.properties.put("address", mr.getAddress());
      this.properties.put("port", String.valueOf(mr.getPort()));
      if (mm.startReceiving(serverID, this)) {
        this.media.put(serverID, mr);
        return true;
      }
    } catch(Exception e){}
    return false;
  }
  public boolean stopReceiving(String serverID) {
    try {
      if (mm.stopReceiving(serverID, this)) {
        this.media.remove(serverID);
        return true;
      }
    } catch(Exception e){}
    return false;
  }
  public void pauseAll() {
    if (!paused) {
      for ( Enumeration l = media.keys() ; l.hasMoreElements() ;) {
        String s = (String)l.nextElement();
        try {
          mm.stopReceiving(s, this);
        } catch(Exception e){ e.printStackTrace(); }
      }
      paused = true;
    }
  }
```

```
  public void continueAll() {
    if (paused) {
      for ( Enumeration l = media.keys() ; l.hasMoreElements() ;) {
        String s = (String)l.nextElement();
        MediaReceiver mr = (MediaReceiver)media.get(s);
        try {
          startReceiving(s, mr);
        } catch(Exception e){ e.printStackTrace(); }
      }
      paused = false;
    }
  }
 public void stopAll() {
    for ( Enumeration l = media.keys() ; l.hasMoreElements() ;) {
      String s = (String)l.nextElement();
      try {
        stopReceiving(s);
      } catch(Exception e){ e.printStackTrace(); }
    }
  }

public void close() {
    stopAll();
  }

public Object addProperty(String name, String property) {
    return properties.put(name, property);
  }

public Object removeProperty(String name) {
    return properties.remove(name);
  }

public Object getProperty(String name) {
    return properties.get(name);
  }

public int hashCode(){

  return properties.get("name").hashCode();
  }
  public boolean equals(Object o) {
    return this.toString().equals(o.toString());
  }

public void writeObject(ObjectOutputStream oos) throws java.io.IOException {
    oos.writeInt(properties.size());
    Enumeration keys = properties.keys();
    while ( keys.hasMoreElements() ) {
      String tk = (String)keys.nextElement();
      String tv = (String)properties.get(tk);
      oos.writeUTF(tk);
      oos.writeUTF(tv);
    }
  }

public void readObject(ObjectInputStream ois) throws java.io.IOException {
    int total = ois.readInt();
    for (int i=0 ; i<total ; i++) {
      String tk = ois.readUTF();
```

```
      String tv = ois.readUTF();
      properties.put(tk, tv);
    }
  }
}
```

MediaReceiver (MMUD) - Interface: MediaReceiver.java

```
package br.ufscar.dc.fmj.framework;
public interface MediaReceiver extends Runnable {
  public String getAddress();
  public int getPort();
  public void close();
}
```

MediaReceiver (MMUD) - Implementação: DatagramMediaReceiver.java

```
package br.ufscar.dc.fmj.framework;
import javax.microedition.io.*;
public class DatagramMediaReceiver implements MediaReceiver {
  private DatagramConnection ds;
  private Datagram dp;
  private Thread thread;
  private MediaConsumer mediaConsumer;
  private int port;
  private boolean running = true;
  public DatagramMediaReceiver(int port, MediaConsumer mediaConsumer) throws java.io.IOException
{
    this.mediaConsumer = mediaConsumer;
    this.port = port;
    this.ds = (DatagramConnection)Connector.open("datagram://:" + port);
    this.dp = ds.newDatagram(65000);
    this.thread = new Thread(this);
    this.thread.start();
  }

public String getAddress() {
   return "localhost";
  }

 public int getPort() {
    return port;
  }

 public void close() {
    this.running = false;
    try{
      this.ds.close();
    } catch(Exception e) {}
  }

public void run() {
    try {
      while (running) {
        dp.reset();
        dp.setLength(65000);
        ds.receive(dp) ;
        mediaConsumer.consume(dp.getData(), 0, dp.getLength());
      }
    } catch(Exception e) {e.printStackTrace();} }}
```

Glossário

ADSL - *Asymmetric Digital Subscriber Line* tecnologia para o acesso à Internet, que aproveita inteligentemente o espaço da sua linha telefônica não utilizado pelo telefone, permitindo assim maximizar a velocidade de navegação e garantindo a estabilidade na ligação.

API - conjunto de bibliotecas de classes e interfaces padrão de uma determinada linguagem de programação

Applet - é um programa que roda em um navegador compatível com *Java*

AVI - formato de vídeo desenvolvido pela Microsoft.

BlueTooth - tecnologia de comunicação por rádio, sem fio, entre equipamentos digitais a curta distância.

Broker - recurso computacional o qual uma de suas funcionalidades é receber chamada remota de um cliente e localizar o objeto que implementa esta chamada, passando os parâmetros necessários a este objeto, viabilizando o retorno dos resultados

CDMA (code-division multiple access) - um dos sistemas de telefonia celular digital mais usado. Funciona de maneira completamente diferente do GSM e TDMA. Os dados são digitalizados e recebem um código identificador. Depois são colocados em um canal junto com os dados de outras chamadas, possibilitando que várias chamadas compartilhem um mesmo canal.

Codec - software responsável pela recepção de uma mídia em um determinado formato

GSM (Global System for Mobile communication) - Sistema digital de telefonia celular derivado do TDMA. É o sistema mais usado no mundo, principalmente por ser muito difundido na Europa e opera na frequência de 900 ou 1800 MHz.

Display – tela de um dispositivo computacional.

Divx - tecnologia de compressão de vídeo, que consegue em um menor espaço uma maior qualidade e quantidade de vídeo, DivX está baseado no formato de compressão MPEG-4.

Download – recurso responsável pela recepção de dados utilizando recursos de rede de computadores.

Encoder - software que converte um determinado arquivo de audio em outro formato

Frame - um quadro/imagem em um determinado instante de um vídeo

*Framework*s – *softwares* que atendem as funcionalidades comuns a várias aplicações que se enquadram a um mesmo domínio de problema.

GPS, módulo GPS - Um dispositivo portátil que permite determinar o local onde se está em coordenadas geográficas. Ele funciona através de comunicação com satélites.

Laptop – computador pessoal resumido em um único equipamento, com um tamanho reduzido, facilitando assim o transporte para diferentes lugares.

Middleware - entidade que possui a função de estabelecer os relacionamentos cliente-servidor entre objetos. Através do middleware, um cliente pode invocar transparentemente um método de um objeto no servidor.

Mbone - uma rede brasileira baseada em uma extensão do protocolo IP chamada IP multicast. Funciona com o protocolo UDP, que não apresenta mecanismos de controle de fluxo e de congestionamento. Desta forma este controle deve ser realizado em camadas superiores ao UDP

Modems – dispositivo computacional responsável pelo envio e recepção de dados na Internet

MMS – serviço de Mensagens Multimídia destinadas a telefones celular.

MOV - formato de vídeo desenvolvido pela Apple denominado Quick Time

MPEG - padrão de compactação desenvolvido por um consórcio que estabelece normas. Encontra-se na versão MPEG-7. Apresenta condições de indexar o conteúdo de um arquivo de vídeo. A partir do padrão MPEG-2 de compactação pode-se obter qualidade DVD

Multicast – tipo de transmissão no qual se permite enviar dados para vários computadores em uma única transmissão para um único endereço.

Notebook - idem *Laptop*

Palmtop - nome comercial que a Hewlett-Packard deu ao seu primeiro handheld. Com o tempo, o termo passou a ser usado para designar os produtos concorrentes, também.

PC – *Personal Computer*, computador pessoal.

PCM - é um método de digitalização de um sinal, ou seja, transformação de um sinal analógico (a voz, no nosso caso) em sinal digital

PDA - *Personal Digital Assistant*, expressão cunhada por John Sculey, da Apple, para descrever o Newton, primeiro handheld sem teclado a ser vendido em escala. Com o tempo o termo passou a designar todos os aparelhos similares, como o Palm.

Pixel - abreviatura para 'picture element' (elemento de uma imagem). É o mais pequeno elemento de uma imagem digital e contém informação acerca da luminosidade e cor. Quanto mais pixels uma imagem tiver melhor é a sua resolução e qualidade..

Players – dispositivo responsável pela execução de uma determinada mídia

progressive-download – recurso que permite a visualização de uma mídia a medida que ela é recebida via download.

Renderizar – processo responsável pela formação de uma imagem ao qual é visualizada na tela do computador.

RM - Formato de vídeo que utiliza padrão de compactação MPEG desenvolvido pelo RealPlayer

RMI - *Remote Metod Invocation*, interface que permite invocação de métodos remotos.

Smartphones - Telefones celulares com os mesmos recursos de um handheld (agendas, calculadora, carga de programas, etc).

SMS (Short Message Service) – recurso que permite o envio e recepção de mensagens de textos entre telefones celulares.

Stream – tipo de transmissão de dados de forma seqüencial.

TDMA (Time Division Multiple Access) - Um dos três sistemas de telefonia celular mais usados no mundo. No TDMA o uso do canal (ou linha) é dividido em três intervalos de tempo, cada um compartilhando uma ligação.

Toolkit - ambiente de desenvolvimento de uma linguagem de programação.

Unicast – tipo de transmissão ponto a ponto entre dois computadores

Upload – envio de dados pela Web.

VHS - (Video Home System). Formato analógico desenvolvido pela JVC em 1976, foi o segundo formato criado para o segmento consumidor

WAP - *Wireless Application Protocol* ou Protocolo de Aplicação sem Fio) é um protocolo desenvolvido para ambientes móveis que necessitem de informações independentemente de sua localidade física.

WebCam – é um tipo de câmera destinada a PCs ligados na Web

Whiteboard - telão sensível ao toque, grava os dados que são escritos através de uma caneta eletrônica

Wireless - comunicação entre aparelhos que usa ondas eletromagnéticas propagadas pelo ar, ao invés de corrente elétrica via cabos.

WBMP – formato de imagens para dispositivos móveis.