Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Departamento de Computação

Programa de Pós-Graduação em Ciência da Computação

"Um Framework para Gerenciamento de Qualidade de Serviço em Dispositivos Móveis"

Camillo Tannuri Hobeika

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes.

São Carlos – SP

Maio de 2004

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

H681fg

Hobeika, Camillo Tannuri.
Um framework para gerenciamento de qualidade de serviço em dispositivos móveis / Camillo Tannuri Hobeika. -- São Carlos : UFSCar, 2004.
126 p.

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

1. Sistemas multimídia. 2. Framework (programa de computador). 3. Java 2 micro edition. 4. Aplicações multimídia. 5. Qualidade de serviço. 6. Sistemas de comunicação sem fio. I. Título.

CDD: 006.7(20ª)

Agradecimentos

- Á minha família pelo apoio e incentivo que recebi durante todos os momentos.
- Ao Prof. Dr. Sergio Donizetti Zorzo, pela orientação, apoio, paciência e amizade durante esses anos de trabalho e aprendizado.
- Aos meus amigos e colegas de turma, que contribuíram com sugestões, companheirismo e presença.
- Aos gerentes, supervisores, colegas e amigos do time da Motorola, pela oportunidade, suporte e compreensão.
- Ao time da Sun Microsystems, Antti Rantalahti (Nokia Research Center) e Ivan Wong (Motorola - PCS), pelas informações e esclarecimentos.

Sumário

Índice de Figuras

Índice de Tabelas

Lista de Abreviações e Siglas

AMPS	Advanced Mobile Phone Service
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
AuC	Autentication Center
BTC	Base Transceiver Controller
BTS	Base Transceiver Station
CDC	Connected Device Configuration
CEPT	Conference European Post and Telegraphf
CLDC	Connected Limited Device Configuration
CPU	Central Process Unit
CRS	Component Reconfiguration Scheme
CSCW	Computer-Supported Cooperative Work
DiffServ	Differentiated Services
DRS	Delay Reconfiguration Scheme
EIR	Equipaments Identity Registers
ERB	Estação de Rádio Base
FDDI	Fiber Distributed Data Interface
GAS	Graceful Adaptation Scheme
GSM	Global System for Mobile Communications
HLR	Home Location Registrer
HTTP	Hyper Text Type Protocol
IEEE	Institute of Electrical and Electronic Engineers
IMEIC	International Mobile Equipament Identity Code
IMTS	Improved Mobile Telephone Service
IMST	International Mobile Subscriber Identity
IP	Internet Protocol
ISO	International Standardization Organization
IVS	The INRIA Videoconferencing System
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JPEG	Joint Photographic Experts Group
JMF	Java Media Framework
JSR	Java Specification Requests
JVM	Java Virtual Machine
Kbps	Kylobits per second
LCL	Lower Control Line
LSL	Lower Specification Line
MCAM	Movie Control, Access and Management
MIDP	Mobile Information Device Profile
MMAPI	Mobile Media API
MOST	Mobile Open Systems Technologies

MPEG	Moving Picture Experts Group
MPLS	Multi Protocol Labeling Switching
MSC	Mobile Switching Center
MSHN	Management System for Heterogeneous Networks
MTSO	Mobile Telephone Switching Office
OSI	Open System Interconnection
QoS	Qualidade de Serviço
RRS	Resource Reconfiguration Scheme
RSVP	Reservation Protocol
RTP	Real Time Transport Protocol
SBM	Subnet Bandwidth Management
SIM	Subscriber Identity Module
TCP	Transmission Control
UCL	Upper Control Line
USL	Upper Specification Line
VLR	Visitor Location Registrer

Resumo

Este trabalho apresenta um Framework para Aplicações Multimídia com objetivo de possibilitar o uso dessas aplicações em Dispositivos Móveis com capacidade computacional limitada, considerando os aspectos de Qualidade de Serviço que podem ser utilizados nesses dispositivos atualmente.

A evolução tecnológica causa um crescimento acentuado no número de usuários de dispositivos móveis, uma vez que a qualidade dos serviços tende também a aumentar, principalmente no que se refere às aplicações multimídia. Devido às características destes dispositivos e das redes em que operam, o tráfego de informações multimídia tende a aumentar os problemas típicos da comunicação. Esse fato levou os fabricantes a restringirem suas funcionalidades a apenas envio e recebimento de dados, não considerando as aplicações de transmissão ou tempo real. Porém, parâmetros de Qualidade de Serviço (QoS) podem ser estabelecidos e ações podem ser tomadas para tentar diminuir o impacto dos problemas inerentes à computação móvel nas aplicações multimídia.

Assim, este trabalho apresenta o projeto e o desenvolvimento de um framework baseado em componentes reutilizáveis em aplicações desenvolvidas sobre a plataforma Java™ 2 Micro Edition (J2ME™), de forma a oferecer maior qualidade na comunicação multimídia para os dispositivos móveis que possuem recursos limitados.

Abstract

This work presents a Framework for Multimedia Applications with objective to make possible the use of these applications in Mobile Devices with limited computational capacity, considering the aspects of Quality of Service that can currently be used in these devices.

The technological evolution cause a accented growth in the number of mobile device`s users, a time that the quality of the services also tends to increase, mainly as for the multimedia applications. Had to the characteristics of these devices and the nets where they operate, the traffic of multimedia information tends to increase the communication typical problems. However, parameters of Quality of Service (QoS) can be established and actions can be taken to try to diminish the impact of the inherent problems to the mobile computation in the multimedia applications.

Thus, this work aims at the research and the development of one framework based on components for applications developed on the platform Java 2 Micro Edition (J2ME^{TM}), of form to offer to greater quality in the multimedia communication for the mobile devices that possess limited resources.

.

Capítulo 1

Introdução

Com o rápido desenvolvimento ocorrido nas áreas de comunicação celular, redes sem fio e dispositivos móveis, os usuários ganharam acesso à informação a qualquer hora e em qualquer lugar. As características das aplicações que utilizam essas tecnologias motivaram intensa atividade de pesquisa para o desenvolvimento de técnicas para oferta de novos serviços.

O desenvolvimento de aplicações utilizadas em dispositivos móveis como telefones celulares e *palmtops* conta com um conjunto completamente novo de desafios para projetistas de interfaces e engenheiros de software. De certa forma, todo o ciclo de desenvolvimento de software é reutilizado e adaptado para os novos requisitos inerentes a dispositivos de pequeno porte.

Esses dispositivos possuem limitações como tamanho de tela, memória disponível e capacidade de processamento, impondo restrições quanto a sua aplicabilidade. Por outro lado, o uso de tais tecnologias deve atender às expectativas do usuário, principalmente no aspecto de usabilidade, simplicidade e segurança.

Apesar dos dispositivos móveis possuírem limitações tecnológicas e físicas, já é possível utilizá-los para oferecer aplicações multimídia e outros serviços como mensagens avançadas onde imagens, áudio e vídeo podem ser transmitidos juntamente com informações textuais; jogos interativos on-line, com avançadas capacidades de som e imagem; transferência de áudio e vídeo; sistemas de segurança e vigilância; comunicação interna e treinamento; áudio e vídeo conferência; comércio eletrônico, dentre outras.

No entanto, para aplicações multimídia a tecnologia atualmente permite apenas a transferência de arquivos para posterior exibição nos pequenos dispositivos. A transmissão dessa informação em tempo real não é suportada

da mesma forma que encontramos em desktops devido a limitações de memória e processamento.

As aplicações que envolvem dados multimídia em ambientes computacionais tradicionais usualmente oferecem gerenciamento da qualidade de serviço garantida e negociada com os usuários. Isso se justifica pela classe de aplicações, tendo em vista oferecer e atender os requisitos impostos. Tais aplicações, utilizando dispositivos portáteis não podem ser tratadas da forma usual, em face das características dos dispositivos.

O desenvolvimento de aplicações para estes dispositivos é normalmente realizado utilizando tecnologias baseadas nas linguagens de programação C ou Java, que algumas vezes possuem ferramentas e APIs disponíveis para auxiliar o desenvolvimento, como no caso de BREW e J2ME respectivamente. Comparativamente, a plataforma J2ME é a mais amplamente utilizada e possui uma maior comunidade de desenvolvedores.

Este trabalho visa oferecer um Framework para o Gerenciamento de Qualidade de Serviço para Aplicações Multimídia em dispositivos com limitações de recursos. Dessa forma, os capítulos a seguir foram estruturados de modo a apresentar conceitos de aplicações multimídia em dispositivos móveis, fundamentos de qualidade de serviço, o suporte oferecido pela plataforma Java, o framework desenvolvido e os resultados obtidos durante a execução de aplicações.

Capítulo 2

Aplicações Multimídia em Dispositivos Portáteis.

A Computação Móvel visa permitir aos usuários trabalharem e operarem em um sistema centrado em uma rede de computadores com mobilidade, criando assim novas classes de aplicações. O ambiente móvel ou ambiente da computação móvel proporciona aos usuários a capacidade de comunicação com a parte fixa da rede, e possivelmente com outros dispositivos móveis, independentemente da sua localização.

A evolução conjunta da comunicação sem fio e da tecnologia computacional permite o atendimento de muitas necessidades do mercado utilizando-se de serviços celulares, redes locais sem fio, transmissão de dados via satélite, rádio, modems e outros.

A classe de aplicações investigadas nesse trabalho é a que faz uso de informações multimídia em dispositivos portáteis. Apresentamos a seguir as características destes três componentes que serão utilizados nesse trabalho: os sistemas celulares, os dispositivos móveis e as aplicações multimídia.

Dentre as diversas formas de representação de dados multimídia, o padrão MPEG foi utilizado devido as suas características que serão apresentadas ainda nesse capítulo.

2.1 Sistemas Celulares

As primeiras conexões de telefones celulares eram baseadas em rádio, onde um operador simples permitia que apenas uma parte pudesse falar de cada vez. Em meados da década de 60 foi implementado o IMTS (*Improved Mobile Telephone Service*) que permitia o acesso direto às redes telefônicas convencionais. A alteração do serviço IMTS foi realizada objetivando o uso de

freqüências de bandas diferentes, sendo renomeado para AMPS – *Advanced Mobile Phone Service*.

Pesquisas têm sido feitas para permitir que um maior número de usuários utilizem uma mesma freqüência limitando a largura de banda disponível. Sistemas celulares surgem com uma tecnologia chamada de "reuso da freqüência" baseado no conceito de célula, que consiste na divisão de uma área em hexágonos, ou seja, pequenas células que podem ou não ser sobrepostas.

Durante a conversação, um usuário pode se mover de uma área de cobertura de uma célula para outra, processo chamado de *hand-off*. Assim que a potência do sinal diminui, a ERB (Estação de Rádio Base) envia uma mensagem de aviso ao MTSO (*Mobile Telephone Switching Office*) que redireciona o controle para uma outra célula que possa propiciar uma melhor comunicação.

A transmissão de dados pode ser utilizada sobre um sistema de comunicação celular. Um computador equipado com um cartão modem PCMCIA celular e um cabo para se conectar ao telefone celular, possibilitam a comunicação sem fio, tornando-se um computador móvel.

A comunicação em um sistema celular é baseada na tecnologia de comutação de circuitos e utiliza freqüências alternadas para evitar interferências entre células vizinhas. Ao passar de uma célula para outra (hand-off), a comunicação sofre atrasos momentâneos, que podem ser considerados excessivamente longos para aplicações de comunicação sensíveis, como as de multimídia.

A mobilidade introduz problemas e desafios que são quase imperceptíveis em ambientes computacionais fixos. Vários problemas já relativamente bem resolvidos em computação fixa ou convencional permanecem praticamente em aberto para ambientes móveis. Os problemas que a mobilidade impõe às redes de computador são os mais

diversos e variados, incluindo desde a velocidade do canal, passando por interferências do ambiente e localização da estação móvel, até a duração da bateria desta estação. Esses e muitos outros parâmetros são fundamentais na elaboração de algoritmos utilizados em aplicações para ambientes móveis, mas não são essenciais nos algoritmos propostos para ambientes computacionais fixos. Além disso, existem novos problemas relacionados com os projetos de hardware e software devido à mobilidade dos elementos computacionais usados na computação móvel. Os principais problemas são apontados a seguir, sendo que a resolução de alguns deles depende da rede utilizada.

A localização de uma ERB deve ser planejada de forma a oferecer um atendimento à demanda e uma exploração eficiente do espectro de freqüências, minimizando custos e mantendo padrões de qualidade de serviço. Neste processo de otimização são considerados fatores conflitantes como a área de cobertura e a quantidade de usuários suportados, ou ainda a área de cobertura e a taxa de transmissão de dados.

Uma vez posicionadas as ERBs, é necessário efetuar a alocação de canais, ou seja, distribuir entre elas o conjunto de canais disponíveis no sistema, sempre observando os níveis de interferência ou distância de reuso. Entre as alternativas adotadas, destaca-se a alocação fixa, onde o mesmo número fixo de canais é alocado a cada ERB. Explorando a mobilidade do usuário, a alocação dinâmica procura alocar os canais conforme as demandas em cada área de abrangência de uma ERB.

Predizer o comportamento dos sinais eletromagnéticos utilizados na comunicação entre as estações e as partes móveis do sistema no momento de sua construção e implantação não é uma tarefa fácil. Faz-se necessário o uso de modelos matemáticos que permitam simular esse comportamento, de forma a permitir aos projetistas testarem várias configurações de sistema até

encontrar uma que satisfaça aos requisitos funcionais, de desempenho e de custos.

O desenvolvimento de modelos genéricos, que podem ser aplicados em qualquer sistema móvel, também é custoso, pois o ambiente no qual esse sistema está inserido varia de sistema para sistema. A simples presença de folhagens e edificações levam a alterações no trajeto da transmissão.

A conectividade entre elementos computacionais não pode ser sempre garantida e, quando existe, possui confiabilidade e vazão variáveis. Em ambientes externos a velocidade de comunicação é geralmente menor que as alcançadas em ambientes internos, onde se pode oferecer uma conectividade mais confiável ao dispositivo móvel.

Dentro de organizações como o IEEE (*Institute of Electrical and Electronic Engineers*) existem vários grupos de trabalho discutindo, projetando e fazendo propostas de protocolos de comunicação para sistemas de comunicação móvel, como técnicas de criptografia, técnicas de compressão e técnicas que evitam a colisão.

GSM, hoje denominado *Global System for Mobile Communications,* foi criado para prover serviços celulares digitais modernos, que permitam às pessoas viajarem sem encontrar problemas com voz e com aplicações que utilizam dados. O desenvolvimento do GSM começou em 1982 quando a CEPT (*Conference European Post and Telegraphf*), formou um grupo de estudo com o objetivo de estudar e desenvolver um sistema celular Pan-Europeia na faixa de 900 MHz.

Alguns dos critérios básicos para o sistema proposto eram a boa qualidade da fala, o baixo custo de terminais e serviços, suporte para *roaming* internacional, entre outros.

Em 1985, a responsabilidade do GSM foi transferida para o Instituto Europeu de padrões e telecomunicações.

Uma rede GSM é constituída pelos mesmos componentes de uma rede celular convencional: antenas, estações base, *backbones* fixos, *gateways* para sistemas de telefones públicos. Os componentes funcionais de um sistema GSM podem ser de um modo geral divididos em: dispositivos de usuário móvel ou estação móvel, subsistema de estações base e subsistemas de rede. Cada subsistema é compreendido de entidades funcionais que comunicam através de várias interfaces, usando protocolos específicos.

A estação móvel possui duas partes distintas: o hardware, ou o dispositivo móvel, e uma informação de assinatura que inclui um identificador único chamado de IMST (*International Mobile Subscriber Identity)*, que é armazenado num módulo SIM (*Subscriber Identity Module*) implementado num *smart card*.

O subsistema da estação base compreende um BTS *(Base Transceiver Station)* e um BTC (*Base Transceiver Controller*). O BTS é formado pelos radio transceptores que definem a célula e o manuseio do protocolo de interface de radio com estações móveis. O BTC gerencia os recursos de rádio para um ou mais BTS, incluindo configuração, freqüência e *hopping*.

Já o subsistema de rede seria o componente central do subsistema de rede normalmente chamado MSC (*Mobile Switching Center*), que provê todas as funcionalidades para adquirir assinantes móveis, incluindo registros, autenticação e atualização de local, roteamento de chamadas, em conjunto com quatro bases de dados inteligentes que juntos formam o subsistema de rede. Essas bases de dados inteligentes são denominadas HLR, VLR, EIR e AuC.

A base HLR *(Home Location Registrer)* contém todas as informações administrativas pertinentes a cada assinante, registrado na respectiva rede na localização atual. VLR (*Visitor Location Registrer*) é a base que contém informações HLR administrativas suficientes para manter o controle de uma chamada e prover todos os serviços para cada cliente móvel na área controlada. A EIR (*Equipaments Identity Registers*) contém uma lista de todos

os equipamentos móveis em uso na rede. Cada peça do equipamento é identificado por um IMEI - (*International Mobile Equipament Identity Code*). O AuC (*Autentication Center*) compreende o banco de dados que contém uma cópia da chave secreta armazenada no cartão SIM.

Embora o projeto desenvolvido nesse trabalho não seja limitado pela tecnologia utilizada na comunicação, o padrão GSM foi citado devido ao fato de ser o padrão 2G mais utilizado mundialmente, superando as tecnologias CDMA e TDMA.

GSM tornou-se um padrão internacional para serviços celulares digitais. Cerca de 74 países assinaram um memorando para adotá-lo como padrão, sendo que o desenvolvimento maior tem sido na Europa.

2.2 Dispositivos Portáteis

Diferentemente dos computadores pessoais, os dispositivos como TVs, *pagers* e telefones celulares avançados são especializados e não de propósito-geral. Apesar de ser possível criar outras funcionalidades para estes dispositivos, o projetista deve ter em mente que sua aplicação deve respeitar as limitações de modo a tornar possível executá-la em qualquer dispositivo móvel.

Dentre os dispositivos portáteis, o telefone celular tem se destacado pelo crescente conjunto de funcionalidades, englobando aplicações de comunicação, entretenimento e outros acessórios. Podemos destacar as seguintes características e limitações encontradas nos modelos disponíveis no mercado nacional nos dias atuais [55]:

- Saída de dados – As informações são apresentadas ao usuário através de uma tela que possui uma resolução aproximadamente de 176x220 *pixels* (640x200 para PDAs) e suporte para 65536 cores.

- Entrada de dados – A operação do dispositivo pode ser realizada através de um teclado numérico, com teclas funcionais que permitem a navegação por menus ou ainda utilizando telas sensíveis ao toque.
- Memórias – Possuem normalmente de 160kB a 64MB de memória disponível, sendo que no mínimo 128 kB de memória não volátil é utilizada por componentes java, 16kB para o armazenamento da aplicação e 32kB de memória volátil para a execução.
- Sistema de Comunicação – O dispositivo deve utilizar uma rede com comunicação bi-direcional, com ou sem fio, provavelmente intermitente e com largura de banda limitada. Os dispositivos que operam no sistema CDMA1X usam taxas de até 144Kbps, enquanto no sistema GSM existe a possibilidade de utilização do padrão Edge (tecnologia 3G) que permite transmissão de dados de 118 kbps até 473 kbps. O sistema UMTS oferece taxas médias de dados de 300 Kbps e taxas máximas de 2,4 Mbps.
- Energia – Por utilizarem normalmente baterias, a autonomia dos dispositivos móveis é restrita.

Além destas características, existem ainda algumas variações da capacidade do software do sistema. Alguns podem possuir um sistema operacional funcionalmente completo, que suporta multi-processamento e sistema de arquivos hierárquico, enquanto outros possuem sistemas operacionais pequenos, baseados em *threads* e sem conhecimento de sistema de arquivos. Diante de tamanha variedade, deve-se levar em consideração os seguintes requisitos de software do sistema:

- Um *kernel* mínimo para o gerenciamento do hardware, tratamento de interrupção, exceções e agendamento de recursos;
- Um mecanismo para ler e escrever em uma memória não-volátil;
- Acesso de leitura e escrita para dispositivos de rede sem-fio;

- Um mecanismo que ofereça o tempo base no o uso de *time-stamps* em registros escritos para o armazenamento persistente;
- Capacidade para escrever no dispositivo gráfico de saída de dados;
- Capacidade para capturar a entrada de dados do usuário por um ou mais dispositivos.

2.3 Aplicações Multimídia

As informações computacionais podem ser representadas através de áudio e/ou vídeo, além de textos, imagens, gráficos e animações [11] de forma a melhor apresentar situações dinâmicas em diferentes áreas, como esportes ou culinária, por exemplo.

A integração desses meios oferece possibilidades adicionais para o uso do poder computacional atualmente disponível, como por exemplo para a apresentação interativa de grandes quantidades de dados. Além disso, esses dados podem ser transmitidos não apenas através de redes de computadores, mas também por redes de telecomunicações, o que possibilita o uso de aplicações nas áreas de distribuição da informação e trabalho cooperativo. A multimídia possibilita uma ampla variedade de novas aplicações, muitas das quais estão em destaque hoje, como vídeo sob demanda, telemedicina, vídeo fone ou vídeo conferência, trabalho cooperativo e aplicações educacionais.

Mídias dinâmicas como animações, áudio e vídeo, são chamadas de isócronas ou contínuas, pois devem ser reproduzidas continuamente a uma taxa fixa. A principal característica das mídias dinâmicas é a existência da dimensão “tempo” cujo significado depende da taxa na qual a mídia é apresentada. Como exemplo, a reprodução de uma mensagem de voz ou de uma música de forma acelerada ou mais lenta ocasiona distorções de significado ou de qualidade do som.

O termo mídia contínua refere-se à dimensão temporal da mídia, como o vídeo e áudio digital em seu nível mais baixo, onde os dados são uma

seqüência de quadros, cada um com uma posição no tempo. As restrições temporais são reforçadas durante a exibição, quando os dados estão sendo reproduzidos. Esse fator possui grande influência no *design* do núcleo de serviços dos sistemas computacionais multimídia, que englobam a captura, geração, armazenamento, busca, processamento, transmissão e apresentação de informações multimídia.

Aplicações multimídia distribuídas podem ser apresentacionais ou conversacionais, embora a maioria das aplicações possua ambos aspectos. Aplicações apresentacionais provêem acesso remoto para documentos multimídia tais como serviços de vídeo sob demanda, enquanto aplicações conversacionais tais como CSCW - *Computer-Supported Cooperative Work* tipicamente envolvem comunicação multimídia em tempo real. Aplicações conversacionais podem ser classificadas: serviços broadcast e sob-demanda. O tipo de aplicação tem influência decisiva nos parâmetros requeridos do sistema. Por exemplo, atraso é menos importante para aplicações apresentacionais do que conversacionais.

Nos Sistemas Multimídia Distribuídos devem ser considerados os componentes do sistema, como interfaces do usuário, sistema operacional, codificação da informação, protocolos de comunicações, base de dados e servidores de arquivo.

A fonte primária de requisitos de qualidade de serviço é a interface com o usuário, que deve ser fornecida para facilitar a escolha de parâmetros. Uma discussão geral da perspectiva do usuário introduziu a "*Quality Query by Example*" [8]. A essência desse método é esconder ao máximo, os parâmetros de qualidade de serviço internos do sistema e apresentar uma escolha de exemplos de qualidade variável, tais como tamanhos de imagens, resolução, cor ou qualidade do áudio. As escolhas do usuário serão automaticamente mapeadas para parâmetros do sistema. Esse método é adequado para

parâmetros apresentacionais tais como vídeo, imagens e som, e menos adequado para parâmetros como tempo de resposta e sincronização.

Os parâmetros do sistema podem ter um grande impacto na percepção da qualidade de serviço pelo usuário. Por exemplo, a velocidade da CPU e do barramento podem limitar a taxa de quadros na apresentação de vídeo, e um monitor preto e branco não exibe imagem colorida. Entretanto, tais parâmetros escondidos estão sendo considerados através dos parâmetros da qualidade de serviço do sistema, pois atuam sobre questões críticas como desempenho e reserva de recursos.

O método de codificação de vídeo influencia os parâmetros de qualidade de serviço. Podemos classificar os esquemas de codificação de vídeo em uma hierarquia [8], onde são utilizadas: compressão intraquadro, compressão combinada intraquadro e interquadro ou ainda compressão em camadas.

O primeiro nível contém esquemas de codificação que usam codificação intraquadro, em que cada quadro é comprimido e codificado independentemente, como o JPEG. Tais métodos permitem variações na qualidade de serviço somente pela diminuição da taxa de quadros. Vários algoritmos de pontilhamento (*dithering*) podem também diminuir a qualidade original.

O segundo nível contém esquemas que usam codificação intraquadro e interquadro, como MPEG e o padrão H.261 para vídeo-telefone. Esse nível de codificação permite abordagens mais sofisticadas, em particular, interação com o sistema de transporte. Delgrossi [12] sugeriu enviar os quadros I, P e B de um vídeo codificado em MPEG sobre *streams* com diferentes prioridades. Os quadros I, que contém código intraquadro, possuem alta prioridade.

O terceiro nível contém esquemas de codificação em camadas ou escaláveis. A idéia aqui é codificar o vídeo em diferentes camadas, sendo que

a camada mais baixa contenha informações básicas do quadro tais como luminância, enquanto as camadas mais altas conduzam as informações adicionais tais como crominância e bits extras para aumentar resolução. O terceiro nível permite a otimização da quantidade de dados transmitidos.

A hierarquia de protocolos oferece mecanismos de qualidade de serviço, utilizando protocolos de enlace, transporte, rede e aplicação.

Analisando a camada de enlace é possível perceber que o protocolo Ethernet não permite reserva de recursos, enquanto tecnologias *Token Ring* e FDDI podem, de acordo com a política de controle de *token,* limitar o atraso máximo e reserva de recursos para garantir desempenho. ATM talvez seja o melhor protocolo de nível enlace para aplicações multimídia, pois fornece facilidades explícitas para manipulação de qualidade de serviço [8].

Requisitos para o transporte de dados visavam principalmente a entrega do pacote não corrompido, largamente satisfeito pelos protocolos TCP/IP. Entretanto, mídias contínuas possuem diferentes necessidades de comunicação: o servidor de arquivo de mídia contínua deve transmitir e entregar os dados como um fluxo contínuo, especialmente para aplicações apresentacionais, porque irregularidades no fluxo de dados causarão degradação na qualidade de áudio e vídeo. Conseqüentemente, parâmetros usuais de qualidade de serviço para protocolos de transporte multimídia são tamanho máximo da unidade de dados do serviço de transporte, vazão e atraso fim-a-fim. Garantir um dado valor do parâmetro de qualidade de serviço requer algum tipo de reserva de recursos, todavia diferentes projetos usam diferentes abordagens, dentre as quais podemos citar Dash, Tenet, ST-II, HeiTS e Berkom [18].

Muitos protocolos no nível da aplicação, tais como RSVP, assumem mídia escalável. Por exemplo, a abordagem de Delgrossi [12] é também baseada na divisão do dado multimídia em *streams* separados. Cada *stream*

teria diferentes características de qualidade de serviço, usando codificação intraquadro e interquadro para otimizar a largura de banda. Outras abordagens provêem primitivas para negociação entre vários componentes de uma aplicação multimídia distribuída. O protocolo para acesso, controle e gerência de filmes (MCAM- *Movie Control*, *Access and Management*) baseado no protocolo X, inclui parâmetros de qualidade de serviço, tais como, confiabilidade, velocidade, modo, qualidade, seção e direção. MCAM provê primitivas para configuração, mas não para negociação destes parâmetros.

2.4 O Padrão MPEG

Aplicações de vídeo utilizam algoritmos de compressão e algumas técnicas de correção de erros para permitir uma eficiente armazenagem ou transmissão de informações multimídia [54]. As imagens transmitidas podem ser estáticas ou dinâmicas, armazenadas previamente ou ainda construídas no momento de solicitação da transmissão, levando a algoritmos de compressão diferentes. Os principais padrões estáticos são Joint Photographic Experts Group (JPEG) para imagens coloridas ou tons de cinza e Joint Bi-Level Group (JBIG) para imagens em preto e branco.Os padrões dinâmicos incluem ITU-T H.261 video-conferencing, H.263, e MPEG-1, -2, e -4. A lista de tipos definida pela Multipurpose Internet Mail Extension (MIME) [24] que é especificamente recomendada em sistemas UMTS incluem AMR narrowband e wideband, MPEG-4 AAC áudio, MPEG-4 video e H.263 vídeo para mídias contínuas, JPEG e Graphics Interchange Format (GIF) para bitmaps.

O grupo MPEG (Motion Picture Expert Group) desde 1980 tem trabalhado com sucesso na padronização de informação áudio-visual (vídeo e áudio), tendo como resultado dois padrões, conhecidos como MPEG-1 (IS-11172) e MPEG-2 (IS-13818). O primeiro especifica o armazenamento de áudio e vídeo à taxas de 1,5 Mbps e o segundo manipula a codificação genérica de TV digital e sinais de HDTV (High Definition TV) [51], o que

proporciona uma alta qualidade mas inviabiliza o uso na maioria dos dispositivos portáteis. Estes padrões têm proporcionado um grande impacto na indústria eletrônica.

A variedade de aplicações torna a representação dos dados áudio-visuais um grande problema, porque a maioria das aplicações pretende possuir a multimídia como característica comum para interatividade com usuário.

As aplicações impõem conjuntos de especificações que variam muito de uma aplicação para outra. A diversidade de aplicações implica em diferentes de conjuntos de especificações. Cada aplicação é caracterizada por: o tipo de dado a ser processado (vídeos, imagens, textos, etc.), a natureza do dado (natural, sintética, médica, gráfica, etc.), a taxa de bits (baixa, média e alta), o atraso admissível máximo, o tipo de comunicação (ponto-a-ponto, multiponto, etc.), e um conjunto de funcionalidades oferecidas (escalabilidade, manipulação de objetos, edição, etc.).

Assim os padrões correntes para multimídia, não podem atender adequadamente as novas expectativas e requisitos dos usuários devido à diversidade de aplicações[52].

Dentro deste contexto dois novos grupos de trabalho MPEG foram criados, para fornecer padrões com o objetivo de atender os requisitos das aplicações multimídias correntes e futuras. Estes grupos são MPEG-4 e MPEG-7.

O padrão MPEG-1 (IS 11172) é um esforço comum da ISO (International Standardization Organization) e IEC (International Electrotechnical Commission) para a padronização de uma representação codificada de vídeo e áudio. MPEG-1 necessita de uma taxa de 1,5 Mbps de largura de banda, podendo variar de 500 Kbps a 4 Mbps. O padrão MPEG é basicamente uma especificação do fluxo de bits e um processo típico de decodificação que suporta a interpretação do fluxo de bits. São previstos três diferentes tipos de quadros [50].

Quadro I (Intra coded picture) é codificado sem referência a outros quadros e proporcionam pontos de acesso onde a decodificação pode começar. A compressão é apenas moderada, similar a uma compressão JPEG.

Quadro P (Predictive coded picture) utiliza codificação preditiva de compensação de movimento de um quadro I ou P e é geralmente usado para outras predições, possuindo um terço do tamanho do quadro I.

Quadro B (Bidirectionally predictive coded picture) é codificado a partir da interpolação entre um quadro anterior (I ou P) e um quadro posterior I ou P. Provê um alto grau de compressão da ordem de 2 a 5 vezes menor que um quadro P.

O padrão é completamente flexível quanto à configuração dos quadros em um fluxo.

MPEG-2 suporta uma vasta variedade de formatos de áudio e vídeo, incluindo TV, HDTV e som "*surround*" de cinco canais. Ele necessita de 4 a 15 Mbps de largura de banda, o que dificulta seu uso para transmissões principalmente para dispositivos móveis.

A arquitetura MPEG-4 permite o uso de objetos áudio-visuais (AVO), a codificação separada de objetos de vídeo e áudio, e a multiplexação de fluxos de dados elementares (elementary streams) separados de objetos em um único fluxo de dados. Similar ao MPEG-1/2, o sistema MPEG-4 é desenvolvido para fornecer multiplexação de fluxo de dados elementares, sincronização e empacotamento.

Adicionalmente, o sistema MPEG-4 fornece parâmetros de representação/manipulação básicos no cabeçalho da camada de fluxo de dados de cada objeto como translação, rotação e zoom. Esse padrão foi explicitamente projetado para lidar com problemas do envio de imagens móveis e videoconferência sobre redes móveis. Diferentemente do padrão MPEG-2 que foi projetado para operar em redes ATM, esse padrão é independente do protocolo de transporte utilizado.

O MPEG-7 é um padrão que define a descrição do conteúdo ao invés de métodos de codificação para compressão de dados. Ele é oficialmente conhecido por Multimedia Content Description Interface e oferece a linguagem de definição de descrição (Description Definition Language - DDL) para definir as características do conteúdo multimídia.

Dessa forma, foram apresentados alguns conceitos fundamentais dos sistemas celulares, dispositivos portáteis e suas características, sistemas multimídia e os componentes que fazem parte destes sistemas. O padrão MPEG apresentado foi utilizado nesse trabalho, sendo que outros padrões podem ser utilizados se suportados pela API utilizada e não havendo necessidade de modificações nesse trabalho.Para uma melhor apresentação dessas informações foi desenvolvido o conceito de Qualidade de Serviço.

Todos os componentes de um sistema distribuído têm seus próprios parâmetros. Alguns desses parâmetros são mutuamente dependentes, dependência essa expressada por mapeamentos entre camadas da arquitetura do sistema. Uma aplicação deve tomar estes parâmetros para contabilizar e negociar seus valores que satisfaçam os limites de todos os componentes envolvidos. Além da negociação inicial, um sistema multimídia distribuído deve planejar o monitoramento e renegociação, como será mostrado no capítulo seguinte.

Capítulo 3
Qualidade de Serviço

Qualidade de Serviço é um conceito altamente importante para todos os componentes dentro de um Sistema Multimídia [11]. Existem parâmetros de QoS em protocolos de comunicação, sistemas operacionais, bases de dados multimídia e servidores de arquivos, além daqueles que afetam diretamente o usuário.

Qualidade de Serviço pode ser definida como “o conjunto das características quantitativas e qualitativas de um sistema distribuído, indispensáveis para atingir a funcionalidade necessária da aplicação”; aqui funcionalidade inclui tanto a apresentação dos dados multimídia ao usuário como a satisfação geral do mesmo.

A QoS de um dado sistema pode ser expressa por um conjunto de parâmetros que estão sujeitos à negociação. Vale ressaltar ainda, que o tipo de aplicação tem papel fundamental na determinação dos parâmetros de QoS, por exemplo: aplicações apresentacionais (acesso remoto a multimídia) toleram certo *delay* (atraso) na transmissão, enquanto que as conversacionais (comunicação multimídia em tempo-real, ex. CSCW) não o fazem.

Dessa forma, são definidas três etapas para o uso dos parâmetros de QoS:

- Especificação de QoS:

A especificação correta de parâmetros é um requisito essencial para oferecer garantias de desempenho. É necessária a criação de uma forma geral e estruturada de especificação, uma vez que as aplicações podem possuir uma ampla variação de requisitos. A especificação realizada na aplicação deve sempre ser realizada utilizando um alto nível de abstração, sendo então automaticamente derivados para outros parâmetros de baixo nível.

- Mapeamento de QoS:

A especificação citada acima é sempre realizada na camada de aplicação, porém vários recursos utilizados (processamento, memória e conexões de rede) são afetados por essas especificações. O processo de mapeamento das especificações para os recursos de sistema (processador, memória, acesso à rede e conexão) pode ser exemplificado pela figura 1 a seguir:

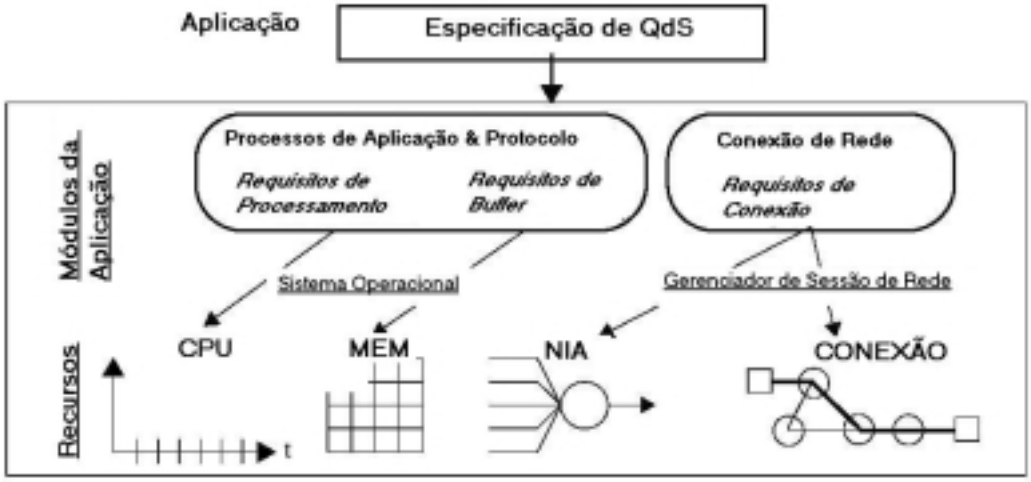

Figura 1 – Mapeamento de QoS

- Gerenciamento de QoS:

O gerenciamento da QoS é realizado por um conjunto de funções baseadas nos parâmetros especificados e mapeados como descritos na figura anterior. Ele envolve o agendamento dos recursos compartilhados para satisfazer as alocações.

Os parâmetros de QoS que são gerenciados para aplicações multimídia podem ser divididos em dois grupos: Parâmetros de Usuário e Parâmetros Internos.

Do ponto de vista do usuário são parâmetros importantes:

- Qualidade da Apresentação: qualidade desejada pelo usuário para atributos como resolução da imagem, áudio e cores.
- Tempo de Resposta: tempo decorrido entre o envio de uma solicitação por parte do usuário e uma resposta por parte do sistema.
- Disponibilidade: porcentagem de tempo que um servidor está disponível durante um dado período de observação.
- Prestabilidade: porcentagem de tempo em que um servidor está disponível e pode aceitar uma requisição do usuário.
- Vazão do Sistema: quantidade de serviço que está sendo oferecida pelo sistema em um determinado instante.
- Custo: estima quão difícil será oferecer a QoS desejada pelo usuário, sendo usualmente medido em termos de uma unidade fictícia de custo.

Parâmetros Internos dizem respeito às questões de desempenho dos diferentes componentes de um sistema e podem ser definidos como:

- Atraso de Propagação na Rede: tempo que um pacote demora para ir de sua origem até o seu destino, em uma rede.
- Capacidade de Acesso à Rede: taxa máxima de dados que um computador pode enviar através de uma rede (determinada pelo link de comunicação).
- Taxa Máxima Efetiva de Rede: quantidade efetiva de dados que uma rede pode receber (normalmente inferior à sua taxa teórica).
- Parâmetros de Baixo Nível: dados das CPUs, das memórias e da qualidade do sinal dos dispositivos móveis, bem como estatísticas relacionadas à transmissão de dados através dessas redes.

O processamento de QoS em Sistemas Multimídia distribuídos envolve várias atividades. Partindo de desejos subjetivos de desempenho, mapeando os parâmetros de QoS para as várias camadas do sistema e iniciando o

processo de negociação (que pode resultar em um acordo “garantido”, *“best-effor*t” ou “estocástico”), para garantir que todos os componentes possam alcançar consistentemente os parâmetros necessários. Além disso, durante a sessão podem ocorrer mudanças nos sistemas que fazem parte do processo de comunicação (motivo pelo qual a QoS atual deve ser continuamente monitorada), o que ativaria um mecanismo de renegociação dos parâmetros.

A especificação da QoS num dado sistema pode ser realizada pelo usuário ou desenvolvedor. A interface do usuário deve esconder, até onde for possível, os parâmetros de QoS (já que são incompreensíveis para o mesmo), mostrando, ao invés disso, escolhas de variação de qualidade. No entanto, o usuário deve saber que as suas escolhas não são independentes, para evitar que ele, automaticamente selecione a melhor qualidade disponível sem se preocupar com as conseqüências.

Os parâmetros dos sistemas finais têm um forte impacto na QoS que o usuário percebe, sendo que as maiores limitações são relativas ao comportamento de tempo-real do sistema. A QoS sofre ainda a influência dos métodos de codificação de dados, especialmente no caso de vídeos, como visto no capítulo anterior.

Três diferentes níveis de QoS podem ser oferecidos pela hierarquia de protocolos. Os protocolos de baixo nível oferecem largura-de-banda suficiente e *delay* aceitável para tráfego multimídia. Os protocolos de rede e de transporte possuem mecanismos para lidar com QoS em redes heterogêneas, mapeando parâmetros de QoS das camadas superiores para as inferiores. Por fim, os protocolos de camada de aplicação suportam uma negociação geral de QoS entre os componentes envolvidos na aplicação.

Outro componente importante para os sistemas multimídia distribuídos é a base de dados. Ela oferece armazenagem persistente e coerente de objetos multimídia, além de acesso concorrente a estes objetos. Numa base de dados, a informação armazenada pode ser dividida em: (1) informação multimídia, que são os próprios objetos e (2) informação de controle, como cenários de

sincronização, regras de localização, parâmetros de QoS, *layout*s, etc. A base deve fornecer ainda, linguagens que possam definir e manipular estes dois tipos de informação.

Todos os componentes de um sistema distribuído têm os seus próprios parâmetros de QoS e alguns destes são mutuamente dependentes. A aplicação deve levar em consideração todos esses parâmetros e negociar os valores que satisfazem a capacidade dos componentes envolvidos.

3.1 Especificação e mapeamento

O primeiro passo na requisição de garantias de desempenho (QoS) é a especificação do que são essas requisições e a quantificação precisa das mesmas.

Como tais especificações devem ser repassadas para todas as camadas da arquitetura, é conveniente que se obtenha certas entradas do usuário e que se mapeie essas especificações de QoS em parâmetros de QoS para as camadas inferiores [16] .

A fase de especificação de QoS permite que o usuário expresse seus "requisitos de percepção", em termos de uma combinação objetiva (ex. tamanho da imagem) ou subjetiva (qualidade da imagem) de entradas. A QoS para aplicações multimídia pode ser especificada através de características temporais (ex. *frame rat*e, *sampling rat*e, tamanho dos dados, razão de compressão, etc.) e espaciais (ex. resolução da imagem, quantidade de cores, etc.). Além disso, a especificação deve conter o preço máximo que o usuário deseja pagar pelo serviço (ou o nível máximo poderia sempre ser solicitado) e as políticas a serem seguidas durante os próximos passos. Tais passos são: negociação, necessária para que todos os componentes envolvidos na comunicação cheguem a um acordo sobre a QoS; admissão de recursos; alocação dos mesmos; monitoramento; e adaptação de recursos, executado caso o monitoramento detecte alguma violação na QoS acertada.

A semântica dos parâmetros de QoS é expressada pelo grau de "compromisso" do provedor de serviço, sendo os extremos: *best-effort*, onde não são fornecidas garantias, e determinística, onde são alocados recursos para garantir o QoS requisitado. Como as aplicações multimídia geralmente conseguem tolerar uma certa degradação no nível de serviço, uma solução "intermediária" é usada.

Após a especificação das requisições de QoS, estas devem ser mapeadas para as camadas inferiores. Tal processo é chamado de mapeamento de QoS, e compreende desde a simples cópia dos parâmetros do usuário até procedimentos mais complexos, como a conversão de *frame rate*, resolução, etc. em valores de largura de banda no nível de rede.

Durante este estágio, uma nova questão surge: a necessidade de uma arquitetura que consiga dar suporte à especificação de QoS. Por exemplo, as camadas do modelo OSI não conseguem dar suporte para aplicações multimídia distribuídas, pois apesar da noção de QoS estar presente nos padrões da ISO, ela é inconsistente e incompleta.

Nas diversas camadas de arquiteturas que seguem este modelo, existem diferentes parâmetros para a especificação de QoS. No nível da aplicação, existem os parâmetros que são especificados pelo usuário e, em seguida, traduzidos para termos quantitativos. No nível do sistema, os parâmetros se referem às garantias de operação em tempo real e à reserva de recursos (CPU, *buffer*s, *bus* de Entrada/Saída e dispositivos multimídia). Na camada de transporte, os parâmetros são necessários para a reserva de recursos através de várias redes, e na camada de rede, os parâmetros são aplicáveis através de simples conexões de rede. Os parâmetros de QoS comumente usados são: *throughput* (vazão), *delay* (atraso de propagação), *jitter* (diferença entre atrasos máximo e mínimo) e taxa de perdas.

A vazão significa quão rápido os dados fluem, seja numa linha de comunicação ou no barramento de um computador. Quanto maior a largura de banda, mais informações podem ser enviadas num dado intervalo de tempo.

Pode ser expressa em bits por segundo (bps), bytes por segundo (Bps) ou ciclos por segundo (Hz).

Como mencionado, a vazão é um fator limitante que representa a medida da capacidade de trafegar informações de um meio físico. Para os cabos de pares trançados de cobre, utilizados em redes locais, a largura de banda depende em grande parte da freqüência na qual se transmite o sinal. No caso da fibra óptica, a largura de banda pode ser definida como a quantidade de informações que uma fibra pode transportar sobre uma distância especificada, medida em MHz/Km e, ao contrário dos cabos de cobre, outros fatores afetam a largura de banda na fibra óptica. Por exemplo, um dos fatores principais é a dispersão (ou espalhamento) que o pulso de luz sofre ao trafegar pelo núcleo da fibra óptica. Quanto maior o comprimento do cabo, maior será a dispersão do sinal óptico e, com uma dispersão excessiva, o sinal poderá não ser reconhecido no ponto de recepção.

Atraso de propagação ou *delay* é o tempo que o sinal leva para percorrer a distância entre o transmissor e o receptor (expresso em ns). A variação deste atraso é conhecida como *jitter*.

Cada pacote responsável pelo transporte de dados é submetido a um atraso variável, de acordo com o tráfego presente na rede e do processamento realizado sobre ele. Se todos os pacotes recebessem o mesmo atraso, este atraso seria propagado integralmente sobre o dado e seria o atraso final do dado.

O problema é a variação deste atraso. Como os pacotes chegam com diferentes atrasos, estes pacotes não podem reproduzir o dado diretamente ao usuário, pois, desta forma a reprodução sofreria cortes em função de atrasos maiores.

Quanto maior o *jitter*, maior será o reflexo deste atraso sobre os dados, de forma a incluir cortes e tornar a informação inteligível. Desta forma, é necessário haver um armazenamento dos pacotes durante um certo período de tempo, evitando cortes durante a execução da informação transmitida, suprindo

as variações do *jitter*. Este armazenamento resulta em um atraso inicial da reprodução da informação.

O *delay* tem influência direta na qualidade de serviço da telefonia sobre IP. O usuário do sistema percebe um intervalo entre suas interlocuções igual a duas vezes o *delay*. Isto se dá porque o seu interlocutor só perceberá o fim da sua fala depois de um tempo igual ao *delay*, e só depois começará a falar. Como sua fala também terá um atraso igual ao *delay*, o usuário terá a impressão do *delay* como um atraso em dobro.

Erros na transmissão podem ser ocasionados por perdas de pacotes, corrupção dos dados ou ainda duplicação de pacotes. Perda de pacotes é o índice que mede a taxa de sucesso na transmissão de pacotes IP entre dois pontos da rede. Taxas elevadas de perdas de pacotes inviabilizam a transmissão de multimídia e precisam ser evitadas. É possível medir a perda de pacotes inserindo números de série no cabeçalho dos pacotes. É normalmente exibida como uma porcentagem, indicando o percentual de pacotes perdidos. Quanto menor a perda de pacote, maior a eficiência da rede.

O cálculo da perda é a razão da diferença entre o número de pacotes enviados e número de pacotes recebidos sobre o número de pacotes enviados. Em relação à perda de pacotes, calcula-se também a proporção de pacotes perdidos e a função de probabilidade de massa (FPM) do número de pacotes consecutivos perdidos, ou seja, a probabilidade do número de pacotes perdidos ser igual a um valor dado. O motivo para o cálculo desta última medida é que, através dela, pode-se avaliar se as perdas ocorrem em rajadas ou não, o que não é possível de ser analisado com a fração de pacotes perdidos. Com estas duas medidas, é possível caracterizar, de forma bastante precisa, o processo de perdas na rede.

3.2 Adaptação

O processo de negociação de QoS garante que cada componente irá contribuir com sua parte na reserva destes recursos. No entanto, com o

aumento de complexidade das redes e aplicações, os Sistemas Multimídia Distribuídos não podem sustentar o nível de QoS negociado por longos períodos de tempo, principalmente devido ao fato de muitas redes serem do tipo *best-effor*t, onde a reserva de recursos é praticamente impossível de ser alcançada. Além disso, os sistemas são heterogêneos, envolvendo diferentes tipos de comunicação e componentes de processamento e mudanças na carga da rede resultam em congestionamento temporário, acarretando aumento de *delay* e da taxa de erros.

O monitoramento dinâmico dos parâmetros de QoS e a adaptação às inevitáveis flutuações de desempenho do ambiente são responsáveis pela extensão da faixa de condições, o que torna a execução do programa aceitável**.**

Os mecanismos básicos de adaptação possuem um único objetivo: manter o ponto de operação da aplicação dentro de uma região aceitável (na qual se considera que esta funciona corretamente).

Considerando os sistemas como sendo estratificados em três componentes: usuário (U), aplicação (A) e sistema (S), pode-se distinguir os seguintes elementos de adaptação:

- Controlador, que pode estar localizado com o usuário (por exemplo, se ele necessita imagens de alta resolução durante um certo ponto da aplicação), com a aplicação (pois ocorreu um aumento no número de *frames* perdidos) ou em partes diferentes do sistema;
- A adaptação pode ocorrer em “S” (quando ocorre a mudança de protocolos, aumento no número de ciclos de CPU), em “A” (mudança do *display* de colorido para branco-e-preto) ou em “U” (aceitação de áudio com qualidade de telefone);
- Parâmetros observados: por exemplo, satisfação do usuário, *dela*y, taxa de perdas, etc;
- Algoritmo de adaptação: Conjunto de regras que disparam o procedimento de adaptação e o resultado desejado;

- *Time frame*: A adaptação pode ser ativada estaticamente (na iniciação, por exemplo) ou dinamicamente (durante a execução);
- Papel do usuário: Ele pode não estar ciente do processo de adaptação, pode somente ser notificado, ou pode controlar diretamente quando e como ele será realizado.

O conceito de adaptatividade pode ser implementado em diferentes contextos e podem ser classificados em adaptatividade centrada no usuário, no sistema ou mista.

A adaptatividade centrada no usuário enfatiza a interface e a interação do usuário. A aplicação controla a adaptação, que ocorre em "A" e "U". Exemplos de implementações: MOST *(Mobile Open Systems Technologies*), utilizados por engenheiros de campo para conduzir e supervisionar trabalhos em linhas de tensão, e *The Krakatoa Chronicl*e, sistema interativo de notícias que apresenta grande adaptatividade às preferências do usuário.

A implementação centrada no sistema funciona como um sistema de controle de *feedback* que tenta manter constante um dado parâmetro. Exemplos: IVS – *The INRIA Videoconferencing Syste*m, sistema de videoconferência para a Internet, e *Vosaic*, sistema que integra áudio e vídeo em documentos HTML da Web.

A adaptatividade mista é baseada na observação de parâmentros do sistema e do usuário. Exemplos: *FastWe*b, projeto que investiga a possibilidade de entrega de áudio e vídeo através da Web; *Multimedia News on Deman*d, na qual o cliente pode procurar pela base de dados multimídia pré-elaborada.

3.3 Propostas para adaptação centrada no sistema

Com relação à necessidade de garantias de QoS para aplicações de natureza isócrona (ex. teleconferência, vídeo sob demanda), dispomos de quatro diferentes abordagens para a tentativa de recuperação automática do nível de QoS combinado, sem degradação[17]: GAS *(Graceful Adaptation*

Scheme), CRS *(Component Reconfiguration Scheme);* RRS *(Resource Reconfiguration Scheme);* DRS *(Delay Reconfiguration Scheme).*

Uma abordagem relativamente conhecida é a GAS que gerencia dinamicamente a QoS através da mudança de parâmetros que a especificam. No entanto, o GAS apenas reage a violações de desempenho na rede estabelecendo uma rota alternativa, além de ser extremamente relacionado ao protocolo Tenet.

A idéia básica do CRS é substituir o componente sobrecarregado (que é detectado através de medições periódicas do desempenho dos elementos da rede) por outro componente de mesma funcionalidade. Quando uma violação é detectada, componentes alternativos são selecionados pelo gerenciador de QoS (com a colaboração do agente de QoS de cada elemento) e uma transição "transparente ao usuário" é realizada. Caso não for possível realizar esta transição, inicia-se uma fase de renegociação de QoS com o usuário.

Como principais diferenças entre o CRS e o GAS pode-se notar que o GAS é relacionado apenas aos sistemas de comunicação, onde cada componente é apenas um nó da rede, enquanto que o CRS é relacionado aos sistemas finais e de comunicações, em que cada componente pode ser de qualquer tipo. Além disso, o GAS é extremamente relacionado ao protocolo Tenet, enquanto CRS independe dos mecanismos usados dentro de um componente. Por fim, o GAS considera a localização do componente-fonte estática, isto é, os dados podem ser produzidos apenas pelo componente-fonte da configuração inicial, em contraposição, o CRS pode tomar decisões de reconfiguração de componentes do sistema.

O RRS tenta manter o *dela*y, *jitter* e a taxa de perda combinados na negociação, através da mudança da quantidade de recursos reservados pelos componentes da configuração estabelecida, tentando compensar a violação ocorrida pelo componente sobrecarregado. Uma vez estabelecida uma nova configuração, ela perdura até que uma outra violação ocorra ou até que o componente sobrecarregado se recupere do problema.

O RRS pode ser implementado sob três critérios: centralizado, onde um gerenciador de QoS centralizado interage com os diversos agentes de QoS, reagindo às violações através da reorganização dos recursos disponíveis; circular com política de contribuição ótima, na qual não há o gerenciador de QoS centralizado, ocorrendo assim uma interação direta entre os diversos agentes de QoS, e tornando o agente do componente sobrecarregado o responsável por realizar as funções desempenhadas pelo gerenciador no modelo centralizado; circular com política de contribuição imediata*,* semelhante ao circular de contribuição ótima, mas no qual os componentes vizinhos tentam compensar totalmente a violação ocorrida no componente sobrecarregado, não se buscando a solução ótima (baseada em um critério pré-definido) para o problema.

A quarta proposta, o DRS, compensa a violação de *delay* para cada unidade de dados. Se o valor do *delay* medido para uma unidade de dados for superior ao combinado, um sinal de violação é enviado junto aos dados em questão, e o próximo componente na cadeia aplica, se possível, um padrão de QoS superior ao que deveria, com a finalidade de compensar o problema que ocorreu anteriormente.

3.4 Aplicações adaptativas

As características da rede em um ambiente sem fio estão expostas a grandes variações, desde pequenas degradações até a perda total do *link* de comunicação. É responsabilidade do sistema-final compensar estas variações, para oferecer uma qualidade de serviço aceitável ao usuário. Surgem aqui dois novos tipos de aplicações: as adaptativas e as pró-ativas [15].

As aplicações adaptativas são caracterizadas pela habilidade de reagir a mudanças na quantidade de recursos. Tais aplicações oferecem ao usuário o melhor serviço possível frente às circunstâncias atuais. Esse comportamento é extremamente vantajoso em ambientes móveis. Há ainda, a possibilidade de

fazer a aplicação se “auto-encerrar” caso os níveis de QoS caiam abaixo de um patamar pré-estabelecido.

Diferentemente das aplicações adaptativas, as pró-ativas afetam ativamente o agendamento de um recurso. Se um recurso se torna escasso, essas aplicações primeiramente tentam se adaptar à nova situação. Caso a adaptação não consegue suprir os parâmetros de QoS especificados pelo usuário ou pela aplicação, esta influencia o agendamento de recursos em um segundo passo, para alcançar um melhor oferecimento de QoS para o usuário. Isto permite que a aplicação atinja seus próprios parâmetros de QoS por um período de longo-prazo, mesmo que mais aplicações venham a requisitar os recursos disponíveis. Note que esse tipo de intervenção não aumenta a disponibilidade de recursos. Em vez disso, os recursos que são agendados para um processo ou tarefa serão aumentados.

Assim, aplicações pró-ativas são indicadas para ambientes móveis com recursos limitados e variações imprevisíveis na qualidade das características de comunicação.

3.5 Protocolos, algoritmos de QoS

Há algumas maneiras de se caracterizar Qualidade de Serviço (QoS). Podemos definir QoS como a habilidade que um elemento da rede possui para oferecer o mesmo nível de garantia para entrega consistente de dados através da rede.

Algumas aplicações são mais exigentes em suas requisições de QoS do que outras, e por isso tem-se dois tipos básicos de QoS [18] :

- Reserva de recursos (serviços integrados): os recursos da rede são divididos de acordo com a requisição de QoS da aplicação e sujeitos a uma política de gerenciamento de largura de banda;

- Priorização (serviços diferenciados): o tráfego da rede é classificado e os recursos são divididos de acordo com critérios da política de gerenciamento de largura de banda. Para permitir QoS, os elementos da rede dão tratamento preferencial ao que foi classificado como mais exigente.

As aplicações, a topologia da rede e certas políticas ditam que tipo de QoS é mais apropriado. Para acomodar a necessidade destes diferentes tipos, existem diferentes protocolos e algoritmos de QoS [18] :

- *Reservation Protocol* (RSVP): Oferece a sinalização para ativar a reserva de recursos da rede (também conhecido como *Integrated Service*s);
- *Differentiated Services* (DiffServ): Oferece uma maneira simples de dividir em categorias e priorizar agregados de tráfego na rede;
- *Multi Protocol Labeling Switching* (MPLS): Provê gerenciamento de largura de banda para agregados via controle de roteamento da rede, de acordo com *"labels"* encapsuladas em *headers* nos pacotes;
- *Subnet Bandwidth Management* (SBM): Oferece a divisão em categorias e a priorização na camada 2 (de "enlace" no modelo OSI) em redes comutadas e compartilhadas IEEE 802.

Estes e outros protocolos foram utilizados para o desenvolvimento de arquiteturas e frameworks de QoS, sendo que os principais serão apresentados em 3.8.

3.6 O impacto da mobilidade na QoS

Só recentemente foi reconhecido que as características especiais dos sistemas móveis exercem grande influência em sua habilidade para suportar QoS, e que uma possível solução é a adaptação às mudanças causadas pela mobilidade, em vez de tentar oferecer garantias rígidas de QoS.

Uma característica essencial dos sistemas móveis é que a qualidade do *link* se torna extremamente variável, freqüentemente de maneira randômica,

embora algumas partes deste efeito possam ser previstas, ou modeladas estatisticamente.

Os principais parâmetros de QoS – confiabilidade de conexão, largura de banda, latência e *jitter* são todos afetados pela mobilidade, tanto durante como entre conexões.

O gerenciamento de QoS pode ser aplicado para lidar com os efeitos destas mudanças. No entanto, diferentemente da situação da rede fixa, os métodos usados para alcançar a qualidade negociada podem ser mais drásticos e iniciados mais vezes. Sugere-se que, devido a natureza das comunicações móveis, não seja razoável tentar garantir níveis de QoS. Assim, uma proposta adaptativa se faz necessária, onde o gerenciamento de QoS especifica uma faixa de resultados aceitáveis.

Alguns aspectos que devem ser considerados no *design* de soluções em computação móvel distribuída são:

- As conexões serão geralmente intermitentes, seja por efeitos do ambiente ou por efeitos do *handoff*, ou por longas desconexões devido a limitações dos sistemas finais;
- Maior latência nas conexões, e particularmente no estabelecimento das mesmas;
- Perdas de conexão;
- Amplas variações na largura de banda.

Estes efeitos requerem que os algoritmos utilizados sejam capazes de gerenciar freqüentes entradas e perdas de nós na rede, e que o *overhead* seja minimizado durante períodos de pouca conectividade.

A movimentação tem dois impactos diretos na QoS: a escolha de recursos para atingir eficientemente as requisições é complicada devido a movimentação do sistema-final; e segundo, no caso de migração, a provisão de recursos e das especificações de QoS associadas é muito difícil. Esses problemas incluem o gerenciamento da replicação dinâmica dos dados

pessoais, a seleção de recursos para conveniência e o gerenciamento de listas de nomes.

O problema do *hand-off* é similar ao da telefonia móvel, com a adição de considerações relativas à realocação de processamento e de dados.

Pode-se medir, também, a taxa de *hand-off* para que se efetue previamente a reserva de mais de uma célula. Através das informações obtidas e do conhecimento do ambiente, pode-se obter predições confiáveis das requisições que serão feitas no futuro, o que possibilita melhor utilização dos recursos, evitando reservas desnecessárias.

Em geral, é possível listar as restrições dos dispositivos portáteis como sendo poder computacional restrito, interfaces restritas, disponibilidade limitada devido às limitações da bateria e dos sistemas de comunicação móvel.

O gerenciamento de QoS em ambientes móveis deve permitir o "escalonamento" das informações entregues, além de oferecer interfaces mais simples ao usuário (quando na conexão utiliza-se mistura de dispositivos portáteis a dispositivos não portáteis de alta-capacidade).

Como observado anteriormente, para oferecer um nível de serviço aceitável aos usuários, muitas aplicações requerem um nível mínimo de serviço da rede subjacente. Em uma rede de alta velocidade, esses requisitos mínimos são especificados através de um conjunto de parâmetros de QoS.

Obviamente, esses parâmetros também podem ser usados em um ambiente móvel. No entanto, o fato de os usuários serem móveis e a imprevisibilidade dos resultados em várias situações onde ocorrem as violações de QoS faz com que se criem parâmetros adicionais, únicos ao ambiente móvel.

Um importante problema de tais ambientes é a degradação de serviço, que ocorre quando muitos usuários com conexões abertas entram em uma célula cuja largura de banda requisitada exceda a capacidade da mesma. Assim, nos deparamos com a questão de como dividir a limitada largura de banda entre todos os usuários.

Identifica-se aqui, um novo parâmetro de QoS : “Perfil de perdas”. Assim, pode-se requisitar aos usuários que, durante o estabelecimento da conexão, informem como preferem que os dados sejam descartados quando houver uma violação na capacidade de largura de banda da célula. Em conjunto com os outros parâmetros de QoS, o “Perfil de perdas” permite alocar banda corretamente para todos os usuários da célula.

3.7 Sistemas de *Buffers* para QoS

Ainda tendo em foco o desenvolvimento da aplicação, podemos colocar em prática algumas técnicas de gerenciamento da QoS, como por exemplo, a remoção da variação do atraso (*jitter*) através do uso de *buffers* de equalização. Com essa técnica, um *buffer* criado no cliente irá armazenar os pacotes que chegam a uma taxa variável. Quando for requisitado, amostras ou trechos serão retirados do *buffer* a uma taxa fixa. A figura 2 abaixo exemplifica este processo que consiste em identificar o pior caso (dmax) e aplicar um tempo de espera a cada pacote relativo ao tempo do pior caso menos o seu próprio tempo(d).

Figura 2 – Buffer de Equalização

Apesar de eficiente, a utilização de *buffers* de Armazenamento Temporário deve ser bem monitorada, pois para suprir os requisitos das aplicações multimídia esses *buffers* não devem se esgotar ou transbordar durante a comunicação. Tais acontecimentos ocasionariam problemas de interrupção de comunicação e perda de informação, respectivamente. Devemos lembrar também que os dispositivos alvos desse trabalho não disponibilizam grandes espaços de armazenamento, de forma que os *buffers* criados devem ter seu tamanho limitado. As figuras 02, 03 e 04 ilustram o comportamento de *buffers* segundo a taxa de recebimento de dados (largura de banda, atraso e variação). É considerado um modelo produtor – consumidor, onde a função A representa a chegada de dados a uma taxa variável e a função C representa o consumo de dados a uma taxa constante[26].

Observamos na figura 3 abaixo que a função A deve sempre ser maior ou igual à função C. A região compreendida entre as funções A e C representa o tamanho do *buffer* utilizado, ou seja, quanto mais afastadas maior o *buffer*.

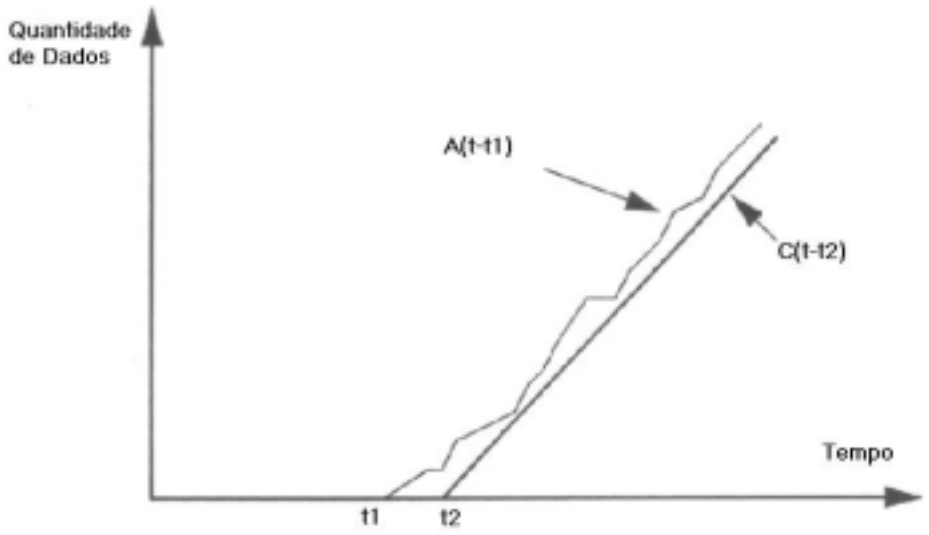

Figura 3 – Razão de Chegada próxima à Razão de Consumo.

A figura 4 a seguir mostra o que acontece se a taxa de chegada de dados for maior que a taxa de consumo. Essa característica permitiria a reprodução dos dados assim que o primeiro pacote estivesse disponível (t1 = t2). Por outro lado, a ocupação do *buffer* aumentaria em função do tempo.

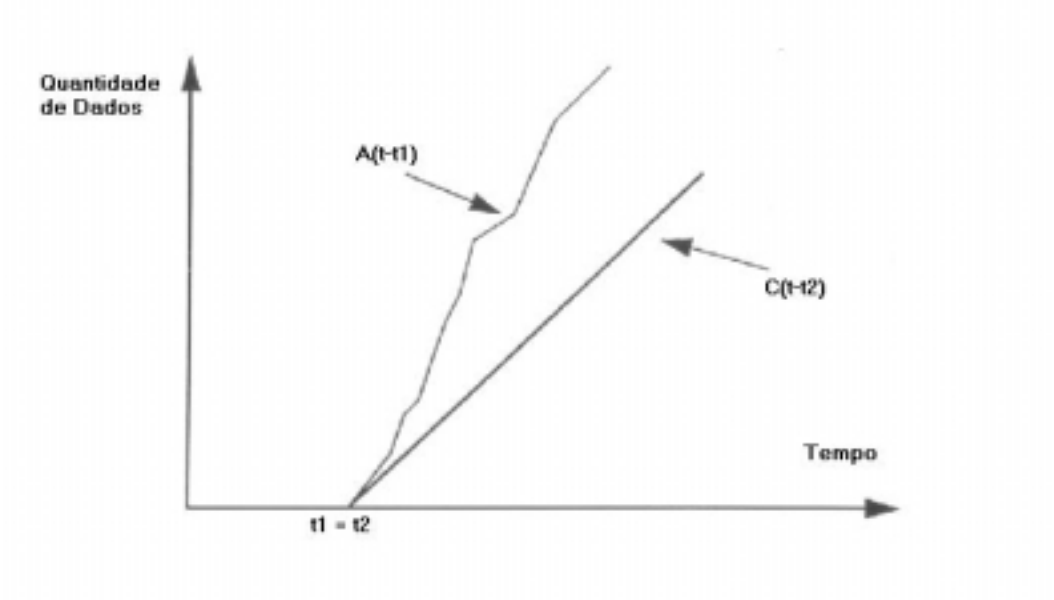

Figura 4 – Razão de Chegada maior que a Razão de Consumo.

A figura 5 abaixo mostra o que pode ser feito caso a taxa de chegada de dados seja inferior à taxa de consumo. O início do consumo de dados é retardado, utilizando grande espaço de armazenamento. Assim como no caso anterior, o uso de mídias longas necessita maior quantidade de recursos.

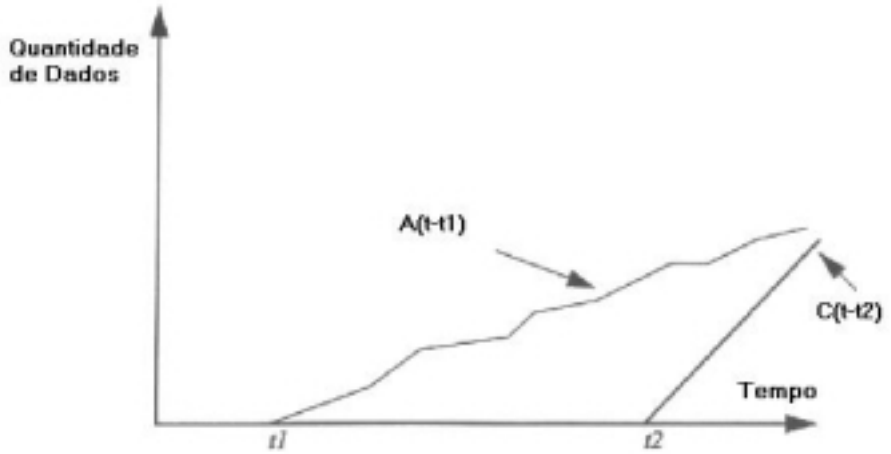

Figura 5 – Razão de Chegada menor que a Razão de Consumo.

Dessa forma, o sistema de controle de memória formado pelos *buffers* de Equalização de Atraso e de Armazenamento Temporário de Dados irá possibilitar o recebimento de dados de forma eficiente e dará suporte para a utilização dos componentes de Gerenciamento de QoS em uma grande classe de aplicações.

3.8 Frameworks de QoS e Projetos Relacionados

As pesquisas voltadas para a transmissão multimídia em dispositivos restritos como telefones móveis são recentes e se restringiam até recentemente a possibilitar a visualização de informações semelhantes às transmitidas aos dispositivos convencionais. Para alcançar esse objetivo os projetos alteravam o formato da informação através do uso de *proxies*, por exemplo, que interceptavam o *stream* de vídeo MPEG, removiam o áudio e convertiam o vídeo em uma seqüência de imagens JPEG [53].

Muitos frameworks de QoS foram desenvolvidos para suprir as necessidades da comunicação multimídia e possibilitar o uso eficiente dos

recursos como processamento, memória, repositórios de dados e comunicação de rede. Esses projetos assumem diferentes abordagens, oferecendo grandes contribuições para o desenvolvimento dos Sistemas Multimídia para sistemas convencionais, contribuições estas que poderão agora ser utilizadas também para dispositivos restritos.

Vários projetos de pesquisa foram desenvolvidos, entre eles *Management System for Heterogeneous Networks* (MSHN) [32], Globus [30] e Legion [31], nos quais os usuários podem selecionar e utilizar recursos de diferentes domínios de forma transparente na execução de suas aplicações.

Aplicações multimídia amplamente utilizadas a alguns anos como *vit* [34], *vat* [33] *rat* [35] são bons exemplos do uso de elementos de adaptação. Na maioria dos casos essa adaptação envolve o escalonamento de suas operações para controlar a quantidade de recursos consumidos.

Arquiteturas de gerenciamento de QoS foram propostas para prover suporte de QoS em redes e sistemas finais. Inicialmente, essas arquiteturas ofereciam garantias de serviço, como em [36, 37 e 38], sendo posteriormente desenvolvidas arquiteturas adaptativas como em [39 e 40].

As arquiteturas e frameworks de QoS podem ser classificadas de acordo com a área em que atuam. Enquanto alguns focam em mecanismos da camada de rede ou camadas inferiores, outros trabalham nas camadas superiores chegando a prover funcionalidades integrais.

*OSI QoS Framewor*k [44] ofereceu uma contribuição inicial no campo das arquiteturas de QoS, definindo terminologias e conceitos e identificando objetos de interesse para os padrões dos sistemas abertos.

A arquitetura *Omega* [7], desenvolvida na Universidade da Pennsylvania, reside na camada de aplicação e seu enfoque é QoS nos sistemas finais. A arquitetura é construída sobre uma rede que suporta QoS (o trabalho foi baseado em tecnologia ATM) mas não atua sobre os nós intermediários. Existe um modelo de camadas nos sistemas terminais, sendo que no topo o usuário possui alguns requisitos de QoS. Esses requisitos são passados a uma

aplicação que por sua vez é executada por um sistema operacional que satisfaz os níveis de QoS desejados para o sistema. Um *broker* de QoS orquestra, controla e mantém todos os tipos de parâmetros de QoS, interagindo com o usuário, aplicação, sistema operacional e rede.

A plataforma Odyssey [28] é baseada no conceito de fidelidade (*fidelity*), o qual é definido como o grau de "similaridade" entre uma versão apresentada para o cliente e a sua cópia de referência no servidor. Essa plataforma provê uma API para possibilitar a negociação de recursos, tais como largura de banda, espaço em *cache* e ciclos de processador, ou recursos específicos de uma determinada aplicação.

Uma aplicação inicialmente tenta acessar os dados no nível mais alto de fidelidade. Se os recursos necessários para isso não estiverem disponíveis, Odyssey notifica a aplicação que, então, seleciona um nível de fidelidade compatível com o nível de recursos disponível. Ela também registra uma "janela" de tolerância para cada recurso de interesse. Posteriormente, se a disponibilidade de recursos melhorar ou degradar além dos limites da janela registrada, o Odyssey notificará a aplicação. A aplicação deve renegociar os recursos necessários para um nível de fidelidade apropriado.

O framework NEC [25] que possui arquitetura baseada em CORBA, tenta introduzir adaptação de QoS em cada camada dos sistemas multimídia heterogêneos. Para isso, os parâmetros são especificados por meio de contratos entre o cliente e o servidor utilizando uma API única e genérica, a qual pode ser aplicada recursivamente para todos os níveis da hierarquia utilizando diferentes níveis de detalhamento. Não existe uma entidade para orquestrar e gerenciar a QoS entre as camadas.

AQUA [40] é uma arquitetura que oferece gerenciamento de recursos, não considerando mecanismos de QoS da rede e utilizando ATM como base. O projeto dessa arquitetura utiliza um paradigma gerenciador-controlador, onde para cada fluxo de comunicação existe apenas um gerenciador e vários controladores para um dado recurso.

Lancaster QoS-A [41] incorpora gerenciamento de QoS da rede, mecanismos de filtragem e escalonamento proporcionando gerenciamento de QoS nos sistemas terminais e nos nós da rede. O gerenciamento do fluxo de informações permite o monitoramento e manutenção dos níveis de QoS que foram especificados no contrato de serviço. As camadas mais baixas (física, enlace e rede) oferecem suporte básico fim-a-fim. A camada de transporte possui um conjunto de protocolos configuráveis e a camada de sessão permite a correção do *jitter* e a sincronização multimídia através dos diferentes fluxos múltiplos. QoS-A pode ser vista como uma arquitetura de QoS completa, integrando aspectos da rede e do sistema final, porém é empregado apenas em aplicações que utilizam sua API, de forma que aplicações legadas são dificilmente suportadas.

AMNet [42] é voltada principalmente para comunicação multimídia heterogênea e oferece filtros para QoS e escalonamento dos recursos de rede, onde o protocolo RSVP é utilizado. O foco de sua utilização é o uso de mecanismos dinâmicos para o controle de informações na rede, adaptando a informação para os requisitos do cliente em cenários de comunicação *multicast*. AMNet não atua sobre a camada de aplicação.

CooQoS [43] é uma arquitetura de gerenciamento de recursos de rede para tratamento de comunicação *multicast* e controle de escalabilidade da rede. O gerenciamento é distribuído e co-operativo feito através do uso de agentes de QoS, que são responsáveis pela renegociação, mapeamento, reserva de recursos, monitoramento e adaptação.

Extended Integrated Reference Model (XRM) [1] utiliza um modelo subdividido em cinco planos: network management (N-plane), resource control (M-plane), connection management and control (C-plane), user transport plane (U-plane) e data abstraction and management plane (D-plane). Esses planos oferecem suporte a funções de gerenciamento de rede e sistema, agendamento de processos, gerenciamento de memória, roteamento, controle

de admissão e de fluxo entre outras funções, formando uma arquitetura completa.

Heidelberg QoS Model [1] oferece garantias a sistemas finais e à rede, utilizando o protocolo ST-II que suporta tanto o nível garantido quanto o estatístico. *Heidelberg* utiliza ainda HeiRAT (Resource Administration Technique), um modelo de gerenciamento que inclui negociação, controle de admissão e agendamento de recursos.

Tenet Architecture [45] desenvolveu um conjunto de protocolos que são utilizados sobre uma rede ATM. Esse conjunto é formado pelo "*Real Time Channel Administration Protocol"* (RCAP) [46], "*Real Time Internet Protocol*" (RTIP) e "*Continuous Media Transport Protocol*" (CMTP) [47]. RCAP oferece suporte para a reserva de recurso e configuração de fluxo, atuando nas camadas de transporte e rede. CMTP é executado sobre o RTIP e oferece entrega periódica e seqüencial de amostras de mídia contínua.

IETF QoS Manager (QM) [6] introduziu o conceito de gerenciador de QoS, que representa uma camada abstrata de gerenciamento, projetada para isolar as aplicações do restante do gerenciamento. Com isso, o QM oferece suporte para aplicações heterogêneas, transparência e escalabilidade.

BRAIN End Terminal Architecture (BRENTA) [27] é uma arquitetura que atua na camada de transporte e é voltada a diversos tipos de aplicações, de forma a abranger desde aplicações legadas até mesmo aplicações distribuídas. As aplicações legadas podem ter acesso a serviços IP de forma convencional (sem QoS) ou podem eventualmente utilizar um painel de controle externo à aplicação para requisitar e monitorar parâmetros de QoS, fazendo uso dos serviços da "*QoS-Enabled Transport Interface"*, também chamada "*BRAIN Service Interface*" (SI). As aplicações com suporte de QoS podem operar normalmente sobre a camada de sessão, enquanto que as aplicações distribuídas podem utilizar a "*BRAIN Component Level API"*. Outra característica dessa arquitetura é o uso de componentes gerenciadores-coordenadores, de modo semelhante à arquitetura AQUA.

MASI End-to-End Architecture [1] oferece um framework genérico para especificar e implementar QoS sobre redes baseadas em ATM, assim como a Lancaster QoS-A. A estrutura é formada por um conjunto de camadas (aplicação, sincronização e conexão) e planos de gerenciamento (gerenciamento de QoS, gerenciamento de conexão e gerenciamento de recursos).

Todas as arquiteturas acima identificadas diferem em vários aspectos, o que pode ser resultado das diferentes “comunidades” pelas quais foram desenvolvidas. Dentre essas comunidades podemos citar a de telecomunicações (XRM e TINA), comunicação computacional (Heidelberg e QoS-A) e a comunidade de padrões (OSI QoS Framework e IETF QoS Manager). Portanto, seria inapropriado declarar que uma estratégia é “melhor” que outra, pois até mesmo arquiteturas pertencentes a uma mesma comunidade focam em soluções para diferentes tipos de problemas.

Muitas técnicas, arquiteturas e protocolos foram desenvolvidos para possibilitar e melhorar a comunicação distribuída, de forma a superar os obstáculos provenientes da comunicação móvel, acentuados pelas necessidades das aplicações multimídia. Alguns desses avanços podem ser utilizados para suprir as necessidades ainda mais extremas encontradas no desenvolvimento de aplicações para dispositivos com recursos limitados, porém alguns se tornam inapropriados por serem demasiadamente complexos e consumirem boa parte dos recursos destes dispositivos.

Capítulo 4

Plataforma Java para Desenvolvimento de Aplicações em Dispositivos Móveis

A tecnologia base da plataforma Java para dispositivos móveis (Java 2 Micro Edition - J2ME) é uma pequena máquina virtual denominada KVM, a qual é utilizada apenas em pequenos dispositivos. No entanto, essa máquina virtual é completamente compatível com a tecnologia J2SE, que é utilizada por dispositivos com maior capacidade de processamento, memória e com sistemas operacionais mais avançados. A figura 6 abaixo mostra a plataforma Java, que cobre desde dispositivos de maior porte (lado esquerdo) até dispositivos de pequeno porte (lado esquerdo), como por exemplo servidores (J2EE), desktops e laptops (J2SE), palmtops e celulares (J2ME):

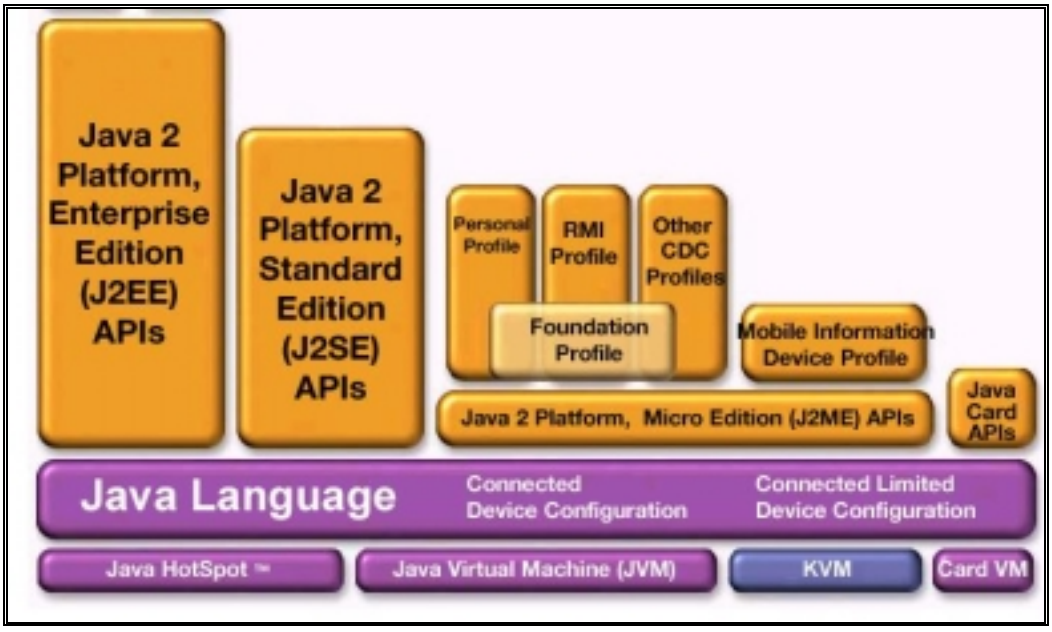

Figura 6 – Plataforma Java

Devido à grande diferença de capacidade dos dispositivos móveis, foram criadas duas configurações: CDC (*Connected Device Configuration*) e CLDC (*Connected Limited Device Configuration*). Essas configurações definem as capacidades da máquina virtual, as capacidades da linguagem Java e as classes de APIs para cada ambiente de configuração.

O CDC cobre todos os dispositivos que possuem uma grande escala de capacidades de interfaces para o usuário e capacidades de memória de 2 a 16 megabytes. Dessa forma, sua utilização é voltada a dispositivos que requerem uma completa implementação da máquina virtual Java e um conjunto de APIs que devem incluir toda a plataforma Java 2.

O CLDC por sua vez, cobre os dispositivos que possuem uma interface mais simples e pouca capacidade de armazenamento, como telefones móveis pagers e PDAs. Essa configuração necessita apenas de 160kB a 512kB de espaço de armazenamento e as aplicações que a utilizam também podem ser executadas na plataforma CDC.

Embora as configurações consigam classificar algumas características e restrições dos dispositivos, ainda não são suficientes para diferenciar os comportamentos e demais necessidades das aplicações. Logo, devido ao fato de que aplicações podem possuir uma mesma configuração e comportamentos diferentes, foi criado o conceito de perfis (*profiles*). Assim, um perfil define o ambiente no nível da aplicação e a configuração define o ambiente no nível da máquina virtual. A aplicação é desenvolvida para um perfil especifico, e os perfis por sua vez são construídos para uma configuração especifica. Ambos definem um conjunto de classes de APIs Java que podem ser utilizadas pela aplicação. Portanto, um dispositivo que é projetado para suportar um dos vários perfis implementa todas as funcionalidades e classes de APIs especificadas pelo perfil e configuração correspondentes.

O MIDP (*Mobile Information Device Profile*) define um modelo de aplicação, sendo projetado para operar sobre o CLDC, que possui um sistema

de arquivos simples e é capaz de realizar comunicação em rede. As aplicações por ele definidas, chamadas MIDlets, consistem de uma classe principal cujo uso é similar ao da classe Applet. Assim como os Applets, os MIDlets são controlados pelo software que os executa, nesse caso um dispositivo que suporta o MIDP e CLDC. Para formar um ambiente completo para estes dispositivos, o MIDP adiciona bibliotecas suplementares que oferecem APIs não tratadas pelo CLDC, como a interface do usuário, base de dados e comunicação de rede específica do dispositivo.

Para alcançar portabilidade, as APIs usam um alto nível de abstração. Como conseqüência, as APIs de alto-nível limitam o controle que o desenvolvedor possui sobre a aparência da interface. A implementação das APIs de interface do usuário, definida pelos fabricantes dos dispositivos, é responsável pela definição e adaptação da interface do usuário ao hardware do dispositivo.

Assim, o MIDP define o comportamento da aplicação, de acordo com as seguintes características:

- Funções de Sistema: Faz uso de algumas funções de sistema. Entre elas podemos citar a função *getResourceStream* para os arquivos de recursos da aplicação e as funções *system.exit* e *runtime.exit* para suporte de notificação finalização de tarefa do MIDlet.

- *Timers*: Para suprir a falta de funções de controle tempo no CLDC, o MID possui as classes *java.util.Timer* e *java.util.TimerStack*, permitindo assim que o MIDlet possa criar *timers*, receber notificações e agendar suas atividades.

- Comunicação em Rede: A comunicação é suportada por um subconjunto do protocolo HTTP, que pode ser implementado usando protocolos TCP/IP ou ainda WAP e i-mode, usando um *gateway* para oferecer acesso a servidores HTTP.

Devido à grande variedade de redes sem-fio, cabem ao dispositivo e à rede oferecer o serviço à aplicação, podendo ser necessário o uso de gateways, servidores de nomes e políticas de segurança. Não é necessário que o MIDlet ofereça essa informação. Além disso, outras áreas de funcionalidade podem ser citadas como não pertencentes ao escopo do MIDP, como APIs de sistema, gerenciamento da aplicação e segurança de baixo-nível, nível de aplicação e fim-a-fim.

A aplicação deve estender a classe MIDlet para que o software de gerenciamento de aplicação a controle, recupere propriedades do descritor da aplicação, notifique e requisite mudança de estados. Todos os MIDlets são derivados da classe MIDlet, que é a interface entre o ambiente de execução e o código da aplicação. Essa classe oferece APIs para chamar a aplicação, parar a execução temporariamente, reiniciar, ou terminar a execução.

O software de gerenciamento de aplicação pode gerenciar múltiplas aplicações dentro de um ambiente de execução. No entanto, a aplicação pode iniciar algumas mudanças de estado e notificar o software de gerenciamento.

As mudanças de estados são simples e estão descritas na figura 7 abaixo:

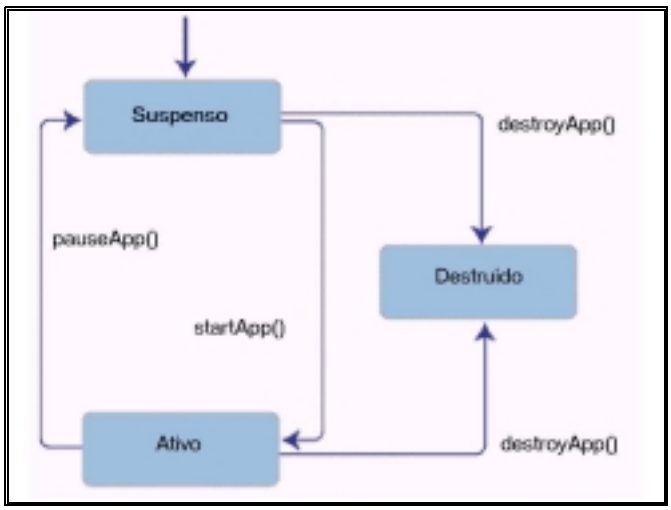

Figura 7 - Ciclo de Vida de um MIDlet.

A aplicação permanece no estado suspenso até que seja chamado o método *startApp*. Com isso, o MIDlet passa para o estado ativo e pode ser colocado novamente no estado suspenso pelo método *pauseApp*. Esses dois estados podem ser destruídos pelo método *destroyApp*.

O desenho real da tela do MIDlet é realizado pela implementação do dispositivo. Essa mesma implementação controla a aparência dos componentes visuais para navegação, barras de rolagem e outras interações. As APIs de alto nível, utilizadas por aplicações como controle de estoque por exemplo, não conhecem essas interações e conseqüentemente não possuem acesso a dispositivos concretos de entrada de dados. Por outro lado, APIs de baixo-nível como as utilizadas por aplicações multimídia e jogos possuem um controle mais preciso dos elementos gráficos, oferecendo no entanto pouca abstração e dificultando o desenvolvimento de um código portável.

O MIDP utiliza quatro tipos de telas: listas, alertas, caixas de textos e *forms*. As listas são similares às existentes em aplicações convencionais e são utilizadas para a criação de menus. Os alertas utilizam todo o espaço da tela e as caixas de texto possuem apenas capacidades básicas de edição, não suportando formatação de dados. O *form*, por sua vez, é uma tela com um conjunto de itens, incluindo imagens, caixas de texto editáveis ou apenas de leitura, campos de data e hora, mostrador de intensidade (como controle de volume) e grupos de escolha. Teoricamente não há limite de quantidade de itens que pode ser adicionado em um mesmo *form*.

A aplicações multimídia podem utilizar as APIs definidas na plataforma. Essas APIs são a Java Media Framework (JMF), desenvolvida para J2SE e a MultiMedia API (MMAPI), desenvolvida para J2ME.

A MMAPI foi influenciada pelo projeto da JMF e as duas APIs compartilham um certo conjunto de similaridades. Ambas são APIs de alto-nível para o tratamento de mídias de tempo contínuo, utilizam o conceito de um Player e um DataSource e oferecem funcionalidades similares.

Diferentemente da MMAPI, o JMF foi projetado para ser utilizado em ambientes servidores ou clientes, onde há poder computacional para execução de aplicações Java, além de um conjunto de características não encontradas na MMAPI como a habilidade de codificar e enviar dados por stream. JMF possui um certo número de funcionalidades que fazem com que seja inapropriado para uso em dispositivos com poucos recursos, como AWT e um número de objetos Java especificados para JMF que tornam seu tamanho um problema.

Por isso, MMAPI é destinada a aplicações clientes que interagem com software e hardware nativo em dispositivos com recursos limitados, havendo poucos benefícios em utilizá-la em ambientes que suportem JMF. Sua implementação não leva em consideração especificação e gerenciamento de parâmetros de QoS como feito em JMF, pois seu projeto visa apenas reprodução de dados previamente armazenados. Veremos no próximo capítulo a proposta de modificação e utilização dessa API para o uso com o framework.

Capítulo 5

O Framework (QoS-KMMAF)

Apesar de todo conhecimento adquirido nas áreas de Computação Móvel, Sistemas Multimídia e Gerenciamento de Qualidade de Serviço descrito anteriormente, o desenvolvimento de soluções para dispositivos como telefones móveis e PDAs apresenta alguns desafios na implementação de uma comunicação de qualidade. Esses desafios provêm das características inerentes desses dispositivos, como limitações de memória e poder computacional. Para enfrentar esses desafios, apresentamos neste trabalho o uso de uma combinação de técnicas relacionadas à Adaptação de Conteúdo e ao Gerenciamento de Qualidade de Serviço, oferecendo componentes que facilitem o desenvolvimento dessas aplicações.

Uma aplicação de vídeo sob demanda, por exemplo, dividiria obrigatoriamente com as demais aplicações o pouco espaço de armazenamento disponível nesses dispositivos. Sendo assim, além da necessidade da criação de uma aplicação simples e compacta, seria inviável o armazenamento da mídia para posterior reprodução ou ainda a criação de *buffers* de maior capacidade para o tratamento de grandes variações de atraso nas transmissões. Com relação ao uso do processamento surge ainda outro problema: a aplicação de vídeo pode ser interrompida a qualquer momento para que uma outra aplicação de maior prioridade possa ser atendida, como no caso do recebimento de uma chamada telefônica. No caso desses dispositivos, o escalonamento dos processos é feito de forma a limitar o tempo máximo de uso do processador por cada evento, permitindo que aplicações com alta prioridade tenham acesso ao processador assim que for necessário. Dessa forma, a aplicação de vídeo não terá condições de realizar processamentos

prolongados, como os que são necessários para o tratamento das características dos dados multimídia.

Portanto, para que seja possível superar estes e outros obstáculos, as aplicações devem ser capazes de oferecer e monitorar parâmetros de Qualidade de Serviço, e os dados devem ser enviados no formato apropriado para as capacidades e características dos dispositivos móveis.

O Framework criado, denominado QoS-KMMAF (*Quality of Service – Kilobyte MultiMedia Application Framework*), recebeu este nome por residir e atuar na camada de aplicação, oferecendo mecanismos de QoS a aplicações multimídia que possuem um código compacto, na ordem de alguns kbytes. A plataforma J2ME foi utilizada como base para o desenvolvimento, pois é suportada pela maioria dos dispositivos móveis encontrados atualmente.

O objetivo do Framework é possibilitar o desenvolvimento de aplicações que utilizem ou não comunicação em tempo real, adicionando mecanismos de equalização e controle da taxa de recebimento de dados quando necessário, de forma a realizar uma reprodução com qualidade da informação multimídia enviada.

Quatro aspectos chave o motivaram:

- Comunicação Móvel;
- Aplicações Multimídia;
- Qualidade de Serviço;
- Dispositivos com recursos limitados.

Apresentamos a seguir as características que compõem o Framework para Gerenciamento de QoS de Aplicações Multimídia em Dispositivos Portáteis, a arquitetura e a sua validação.

5.1 Características do Framework para Gerenciamento de QoS em Dispositivos Portáteis

O QoS-KMMAF é um framework para a camada de aplicação, assim como a arquitetura *Omega* e outras citadas anteriormente. Seu foco está em oferecer mecanismos que permitam a criação e utilização de aplicações multimídias que atendam aos princípios de QoS.

Os mecanismos de Sistemas de *Buffers* descritos anteriormente serão utilizados para auxiliar a manutenção da QoS. O atraso e a variação do atraso são fatores fundamentais que devem ser considerados para a realização de transferências de mídias contínuas e devem ser mantidos dentro de limites particularmente rigorosos de forma a preservar a compreensão da informação do áudio e vídeo no dispositivo que executa a reprodução.

Durante o processo de oferta de um serviço *streaming*, o servidor abre uma conexão para o cliente e inicia o envio da mídia a uma taxa aproximadamente igual à taxa de reprodução. Durante a recepção, o cliente reproduz os dados com um pequeno atraso. Este tipo de serviço não apenas possibilita reprodução dos dados sem que toda a informação já esteja armazenada, deixando memória livre no dispositivo cliente ou ainda permitindo a reprodução de grandes quantidades de dados, mas também permite que os dados sejam transmitidos em tempo real, a medida em que os eventos acontecem.

O usuário faz uso de um programa que realize a descompressão dos dados e envie as informações para os mecanismos de áudio e vídeo do dispositivo. Esta aplicação cliente deve ser capaz de controlar os fluxos da *stream* e gerenciar os fluxos de mídia.

5.2 Arquitetura e Implementação do Framework

Como vimos, as Aplicações Multimídia podem ser divididas em dois grandes grupos: Conversacionais e Apresentacionais. No primeiro grupo, as informações são geradas e transmitidas em tempo-real, o que torna o atraso muito crítico. A aplicação típica seria a comunicação entre dois dispositivos móveis.

Já no segundo grupo, em que as informações são previamente geradas e armazenadas, o modelo de comunicação é cliente-servidor e a aplicação típica é o serviço de vídeo sob demanda. Para esse tipo de aplicação é utilizada normalmente a prática de obtenção prévia da mídia para posterior reprodução, evitando vários problemas de transmissão de dados. Porém esse recurso não é viável para dispositivos móveis, que possuem grandes restrições de espaço de armazenamento, existindo assim as mesmas restrições de tempo das aplicações conversacionais.

O QoS-KMMAF irá tornar possível o uso destes dois grupos de aplicações em dispositivos portáteis, utilizando a plataforma J2ME como base do desenvolvimento, como mostra a figura 8 a seguir.

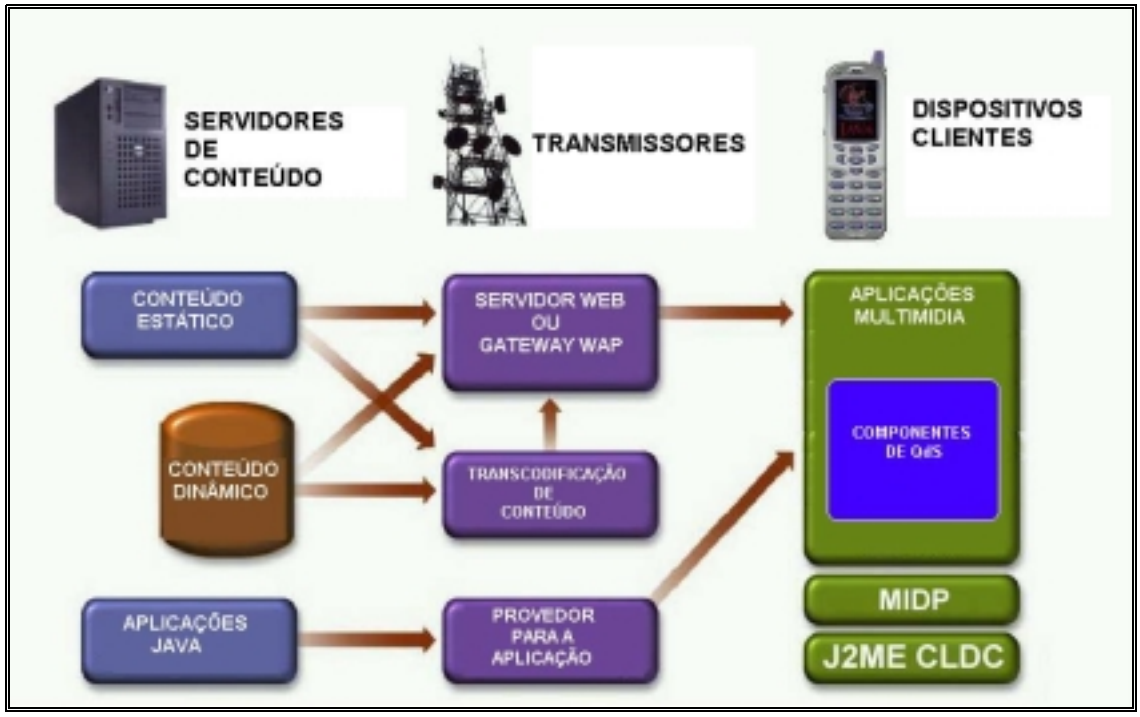

Figura 8 – Comunicação Cliente-Servidor na Plataforma J2ME

Apesar da disponibilidade de todos os recursos da plataforma J2ME apresentados anteriormente, o MIDP versão 2.0 (JSR-118) [49] possibilitou apenas alguns avanços relacionados a sons e segurança. A MMAPI (JSR-135) [48] foi criada para dispositivos com CLDC, possibilitando a criação de aplicações com suporte de geração de tons ou até mesmo reprodução de dados multimídia, como áudio, seqüências MIDI, filmes, clipes e animações. Podemos então utilizar esse novo recurso para criar aplicações multimídias de tempo real, que permite ao usuário visualizar informações assim que os dados comecem a ser recebidos.

A API projetada pela especificação JSR-135 permite apenas o *download,* ou seja, a aquisição completa do arquivo multimídia, mesmo que a fonte de origem seja uma conexão *streaming*. Apesar dessa estratégia sofrer menor influência de parâmetros como atraso e variação de atraso, ela apresenta aumento no tempo de espera inicial de reprodução, inviabiliza a

utilização de mídias de longa duração e não pode ser utilizada em aplicações conversacionais que necessitam de comunicação em tempo real.

Visando possibilitar o uso desse mecanismo, a API foi modificada de modo a substituir o processo de aquisição de dados para um formato de tempo real. Podemos observar na figura 9 abaixo os cinco estados da aplicação *player* e os métodos de transição de estados definidos originalmente.

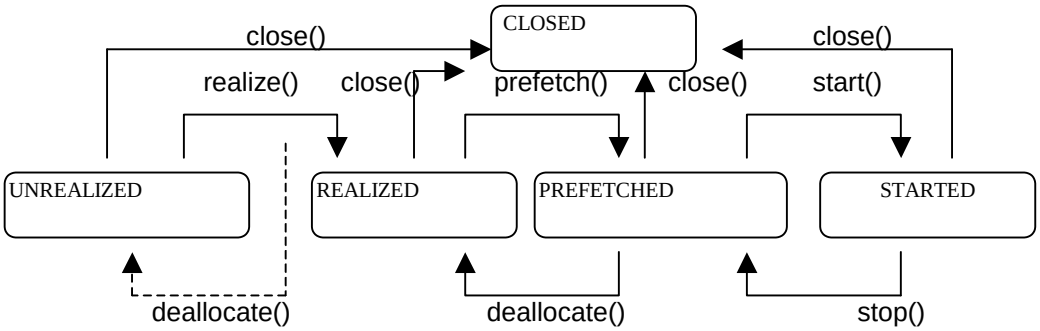

Figura 9 - Estados definidos pela JSR-135.

A transição do estado *Unrealized* para *Realized* é a responsável pela busca e aquisição do arquivo multimídia. Assim que toda a informação tenha sido armazenada, as verificações e preparações serão executadas pelo método *prefetch*() e caso tudo esteja preparado e verificado o método *start*() será chamado para iniciar a reprodução.

Como mostramos anteriormente, este sistema não atende algumas necessidades de aplicações de tempo real e por esta razão foi realizada a alteração indicada na figura 10, criando a capacidade de adquirir e reproduzir os dados ao mesmo tempo.

Podemos notar nesse novo projeto, que durante a transição para o estado *Realized* apenas uma parte inicial dos dados é requisitada. Esta parte inicial é armazenada no sistema de *buffers* e utilizada para a verificação de

jitter, tratamento inicial de *delay* e demais verificações necessárias para a realização do *prefetch*.

A principal alteração, no entanto está localizada no estado *Started*, que originalmente era responsável apenas pela reprodução da mídia até o seu término. Com a nova implementação, esse estado passa a adquirir os dados subseqüentes juntamente com o processo de reprodução dos dados já preparados. O processo criado é um laço que segue o mesmo padrão do processo original completo, ou seja, sub-estados são criados de forma que os novos dados executem os procedimentos de *realize* e *prefetch* antes de serem reproduzidos.

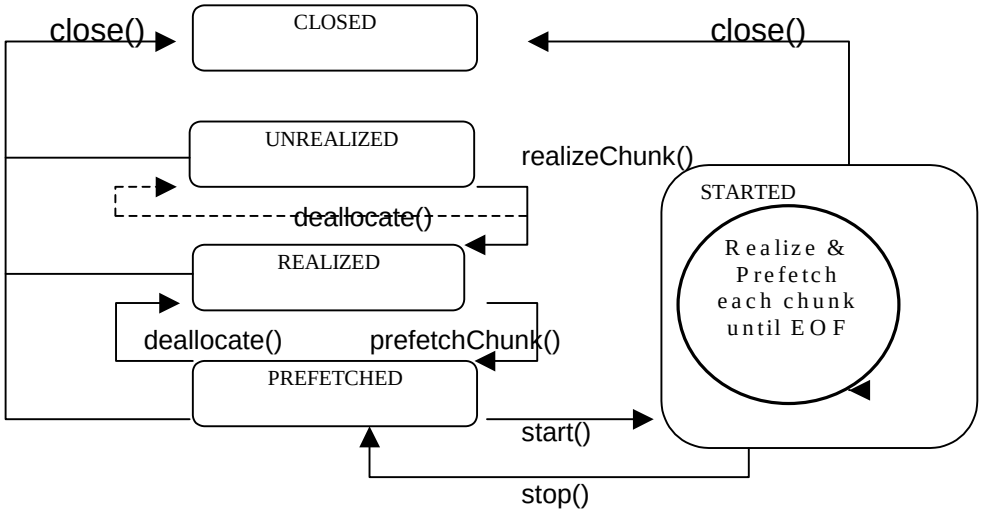

Figura 10 - Alterações na API.

Devemos ressaltar que esse novo desenvolvimento não altera o formato original da mídia. Ele permite, por exemplo, a reprodução de arquivos que ocupam dezenas de Mbytes em dispositivos com memória disponível da ordem de 1 ou 2 Mbytes, sem mudanças na qualidade. Os dados apresentados, no entanto, estão restritos ao desempenho e características de cada dispositivo.

A aplicação desenvolvida através desse framework pode ser criada com os mesmos mecanismos disponibilizados para os demais MIDlets convencionais como *forms*, *lists*, *splash screens*, *notices* e outros, mas deve possuir os elementos básicos do framework apresentados a seguir na figura 11, que ilustra como é a arquitetura de uma aplicação com a utilização do Framework.

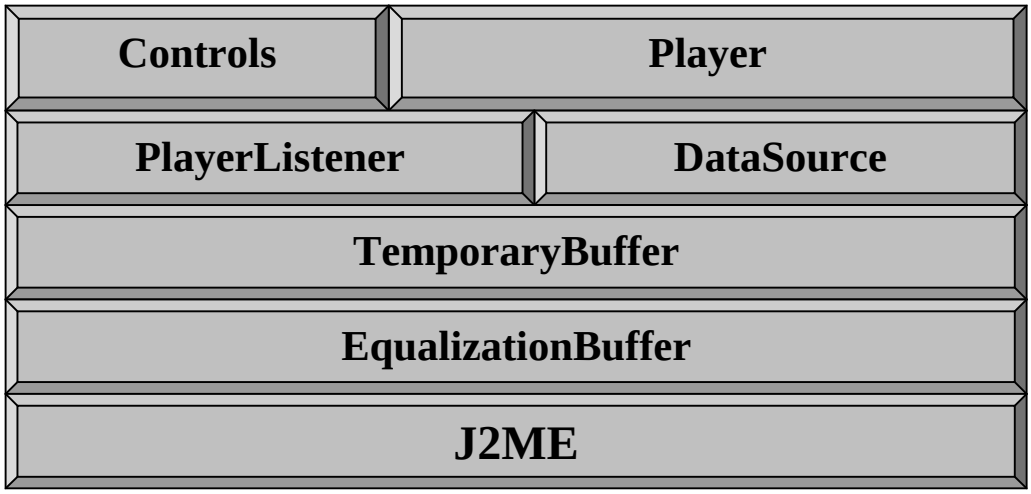

Figura 11 – Arquitetura das Aplicações

Podemos notar na figura acima seis componentes que devem ser considerados para esse modelo conceitual:

- O *Player* recebe e decodifica o dado multimídia, sendo completamente neutro ao tipo de dado que recebe, ou seja, a única diferença entre um *Player* de áudio e um *Player* de vídeo é o conjunto de controles disponíveis para cada um.

- Um *DataSource* oferece tratamento para protocolos, métodos que controlam a reprodução da mídia e sincronização básica. Ele esconde os detalhes de como os dados são lidos na origem (um arquivo ou servidor *streaming*, por exemplo). Os mecanismos de transporte de mídia podem usar protocolos HTTP ou RTP.

- O PlayerListener receberá os eventos assíncronos gerados pelos controles e irá executar o monitoramento das condições do *Player*, do *DataSource* e do *TemporaryBuffer*.

- Controles são associados ao *Player* oferecendo suporte para a manipulação dos dados, como navegação, volume e qualidade. O conjunto de componentes de controle forma a funcionalidade da aplicação. Assim, componentes já existentes, como os de áudio, por exemplo, serão utilizados e modificados visando a formar a nova aplicação. Podemos citar como controles principais:

FramePositioningControl	Controla a posição dos frames de vídeo.
GUIControl	Provê GUI funcionalidades.
MetaDataControl	Recupera informações inseridas nas mídias.
MIDIControl	Provê acesso às funções para arquivos MIDI.
RateControl	Controla a relação entre o tempo da mídia e o tempo base.
RecordControl	Controla a gravação da mídia do Player.
StopTimeControl	Especifica o tempo de parada do Player.
VideoControl	Controla a apresentação do vídeo.
VolumeControl	Manipula o volume do Player.

Tabela 1 - Controles de Aplicação

- O Buffer Temporário é responsável pelo tratamento do atraso introduzido por problemas de comunicação. Esse atraso pode ser criado durante a busca dos dados, codificação, transferência e decodificação. Durante seu processo o buffer deve seguir duas regras simples: não esvaziar e não transbordar. O esvaziamento do buffer significa uma interrupção na

comunicação, pois os dados não estavam disponíveis no momento em que deveriam ser apresentados ao usuário. Por outro lado, caso o buffer receba mais dados do que seja capaz de suportar, alguns dados serão descartados e uma nova requisição deveria ser feita. No entanto, a natureza das transmissões em tempo real invalida estes dados perdidos não havendo razão para a retransmissão após um certo período uma vez que a requisição pode provavelmente não chegar a tempo e em vários casos a informação causaria menor impacto negativo se descartada ao invés de requisitada novamente.

- O Buffer de Equalização é o componente responsável pela remoção da variação do atraso que pode ser adicionada pelos mesmo meios descritos anteriormente. A equalização do atraso através de armazenamento em memória é uma técnica conhecida para transmissões de dados multimídia sobre redes de comunicação não isócronas [26]. Quando os pacotes são recebidos de forma desordenada ou atrasados o *buffer* inicia um processo para homogeneizar os problemas de transmissão e com isso introduz uma maior latência. Durante esse processo adiciona-se a cada pacote o atraso necessário para que o atraso final seja o mesmo para todos os pacotes, ou seja, o atraso final será o mesmo do pior caso, de forma que a aplicação receba os dados a uma taxa quase constante.

Além desses componentes, devemos destacar o *Manager,* que é o ponto de acesso para se obter recursos dependentes do sistema, possibilitando a implementação de mecanismos específicos para a construção de *Players* e *DataSources*.

Em uma visão mais detalhada da implementação, o *Player* é criado a partir da classe *Manager* e formado por componentes, como ilustrado na figura

12. Uma vez iniciada a comunicação com o servidor de mídia, o *Player* utilizará um *DataSource* para a recepção da informação multimídia transmitida.

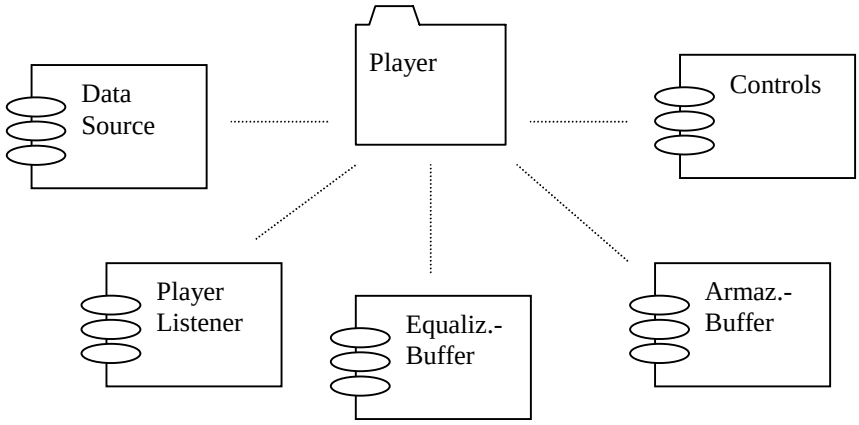

Figura 12 – Componentes da Aplicação.

O *Player* não precisa ser necessariamente a aplicação inicial, ou seja, não precisa estender a classe MIDlet. Ele pode ser chamado por um menu ou browser, por exemplo, fazendo parte de uma aplicação com funcionalidade maior. Deve implementar um CommandListener para especificar o tratamento dos comando de deseja implementar e um PlayerListener para tratar os eventos gerados para ele. Seus métodos principais possuem as funções de se desenhar na tela, adicionar e remover comandos, se atualizar e realizar as chamadas dos comandos que receber. Seus principais atributos são o seu estado corrente e o *DataSource* que utiliza.

A figura 13 ilustra as classes e métodos que juntos compõem a aplicação desenvolvida.

Figura 13 - Classes do Package Application

O *DataSource* é responsável por realizar a conexão e retornar informações sobre os dados, como origem, tipo, descrição e tamanho. O principal atributo dessa classe é a origem dos dados de onde será construído o objeto DataSource. É responsável também por oferecer métodos que retornem o tipo e tamanho do conteúdo que estará oferecendo, iniciem a conexão e comunicação, terminem essa conexão e faça a liberação dos recursos alocados, iniciem e interrompam a transferência dos dados. As classes que formam o pacote DataSource estão indicadas na figura 14.

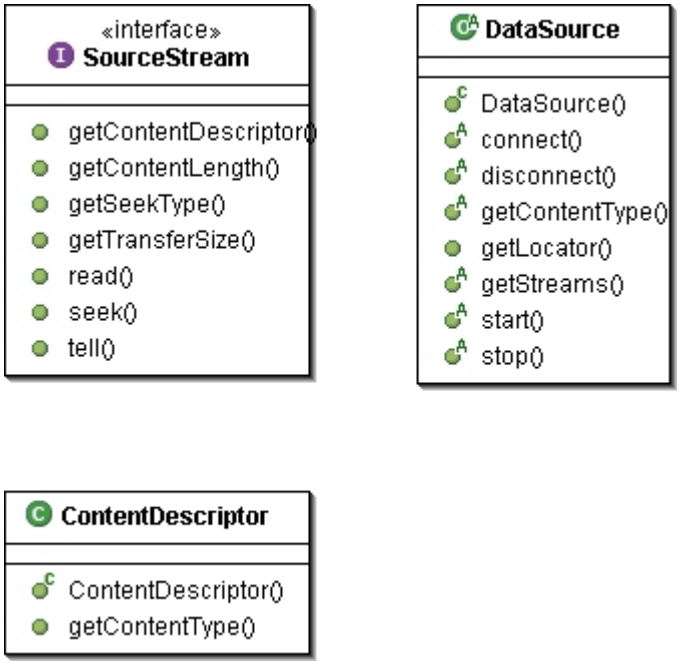

Figura 14 - Classes do Package DataSource

Os três mecanismos que se destacam nesse componente são o de busca, verificação de tamanho do bloco de transferência e o de leitura de dados. O mecanismo de busca é utilizado para determinar o ponto exato onde a transferência deve ser iniciada. A verificação do tamanho do bloco de transferência é utilizada para determinar a quantidade de dados que deve ser

requisitada caso a aplicação tenha o controle do fluxo. O método de leitura especifica o *buffer*, o *offset* e a quantidade de dados a ser lida.

O componente *PlayerListener* será utilizado para prover tratamento para os eventos assíncronos gerados para a aplicação. A aplicação deve implementar um *PlayerListener* que deve ser registrado através do método *addPlayerListener*. Esse listener possui eventos como *started*, *stopped*, *buffering_started, buffering_stopped, end_of_media* entre outros. Novos eventos podem ser atribuídos e o método *playerUpdate* é acionado para acionar a notificação.

Os sistemas de *buffers*, figura 15, oferecem o suporte à comunicação com qualidade, removendo a variação do atraso e possibilitando a aquisição de informações sobre o andamento da recepção dos dados.

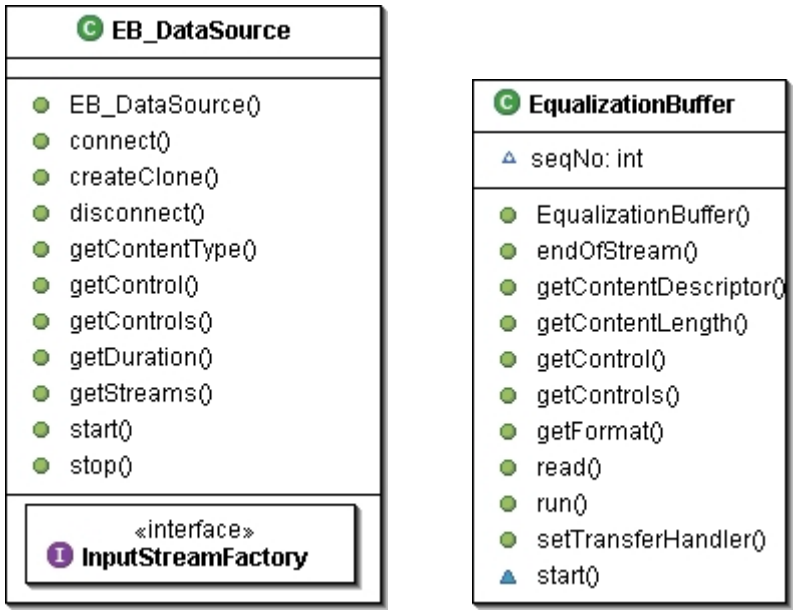

Figura 15 - Classes do Sistema de Buffers

Com as informações adquiridas pelo *DataSource*, *PlayerListener* e pela verificação da ocupação do sistema de *buffers*, serão acionadas medidas para a manutenção da QoS, quando necessário. As medidas previstas são o uso do componente de controle *RateControl*, descarte de quadros e alteração da mídia transmitida. Outras alterações podem ser realizadas na origem dos dados e não no cliente como ocorre normalmente em técnicas de aplicação de filtros, pois os dispositivos clientes não possuem recursos computacionais suficientes para a realização dessa tarefa.

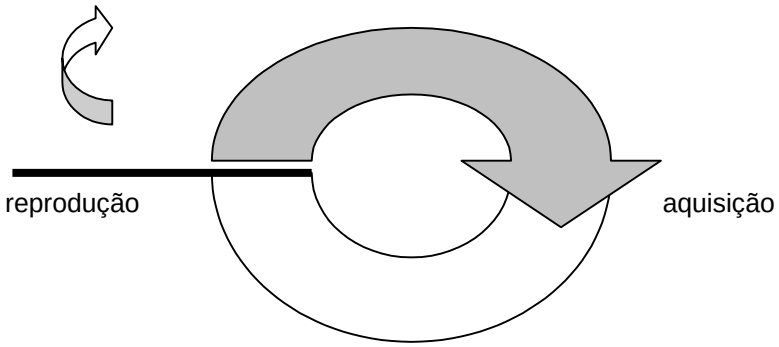

Figura 16 - Buffer Circular

Esta implementação utilizou o formato buffer circular para este processo, como mostra a figura 16, de forma a evitar requisições de alocação e liberação de memória durante a execução, pois este processo consumiria recursos excessivos com chamadas ao *garbage collector* ou “coletor de lixo”. Também foi introduzido um processo de descarte de quadros no processo de reprodução caso a ocupação do buffer exceda 85%. A aceleração da reprodução visa evitar a escrita de dados sobre dados ainda não reproduzidos.

Os componentes criados e utilizados neste framework possuem um baixo grau de dependência, sendo que dessa forma eles podem ser eventualmente substituídos por novas versões. Novos componentes podem ser adicionados ao framework, como por exemplo, um gerenciador/negociador de QoS, listeners para condições da rede ou ainda novos controles. Componentes podem ser removidos do framework. Um exemplo seria a criação de um buffer de equalização “perfeito”, o que tornaria dispensável o uso de um buffer de armazenamento temporário.

Veremos a seguir informações sobre o ambiente de execução e os resultados obtidos durante o uso da aplicação multimídia desenvolvida.

5.3 Validação dos Resultados

Este trabalho torna possível a comunicação multimídia em tempo real utilizando a plataforma J2ME. A aplicação de recepção de vídeo criada é testada em um ambiente emulado, através do uso de um emulador de dispositivos (J2ME – Wireless Toolkit) e um emulador de rede (NISTNET), como mostra a figura 17 abaixo:

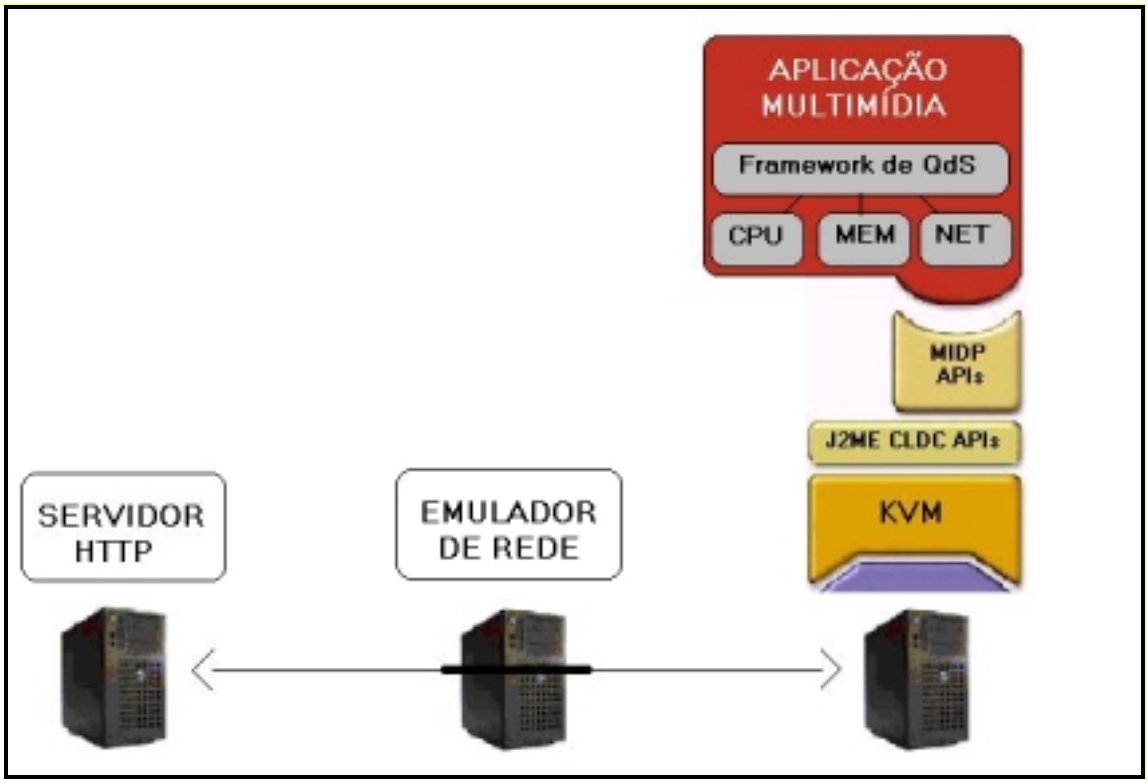

Figura 17 – Ambiente de Teste para Validação

O servidor utilizado para os experimentos foi executado em um PC com CPU de 1100 MHz com 128 MB de memória executando Apache server /1.3.20 (Linux/SuSE) PHP/4.0.6.

A tabela a seguir mostra as propriedades do arquivo multimídia utilizado para testes:

Largura:	320 pixels
Altura:	240 pixels
DPI de X:	96 dpi
DPI de Y:	96 dpi
Formato:	MPG
SubFormato:	Descompactado
Tipo:	RGB de 24 bits
Frame rate:	~25 Frames/seg
Data rate:	Variável bit rate
Áudio:	44100 Hz, 16 Bit, Stereo, 192 Kbps
Tamanho em memória:	De 45.878 a 970.500 bytes
Tamanho em disco:	18.452.000 bytes
Duração:	~4 minutos

Tabela 2 - Propriedades do Arquivo de Mídia

O emulador de dispositivo, que pode ser observado na figura 18, não representa exatamente a aparência de um determinado dispositivo, mas oferece a emulação de um dispositivo genérico que é capaz de executar aplicações. O dispositivo emulado possui como características uma resolução aproximada de 176x208 pixels, suporte para 256 cores, suporte para áudio e um teclado convencional de dispositivos celulares, com duas “softkeys” e um controle direcional.

Figura 18 – Emulador para Dispositivos.

O emulador de rede (figura 19) é uma ferramenta de propósito geral para emulação do desempenho de redes IP. A ferramenta é projetada para permitir experimentos controlados com aplicações sensíveis/adaptativas e protocolos de controle para redes, em um simples laboratório. Operando no nível IP, é possível emular características fim-a-fim críticas para redes os diversos tipos de redes.

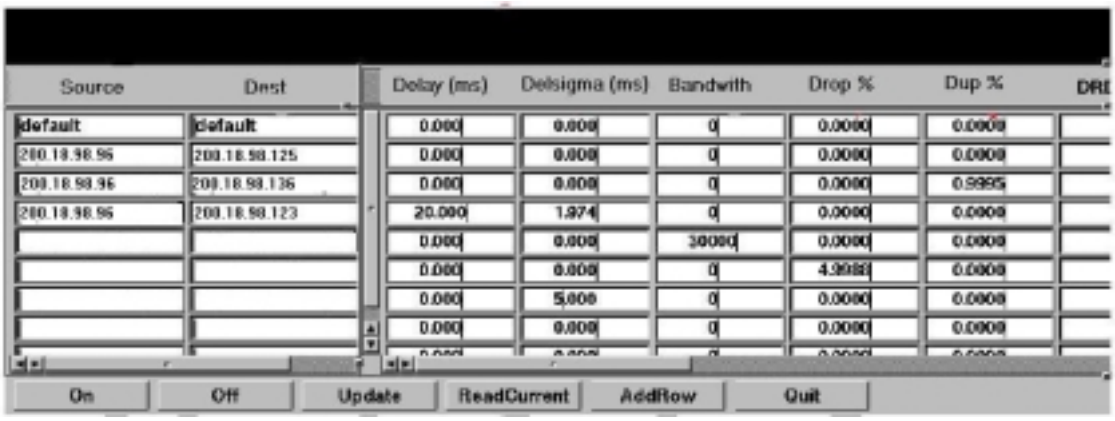

Figura 19 – Emulador de Rede (NistNet)

A ferramenta permite a um roteador emular cenários complexos de desempenho, incluindo: atraso de pacotes, congestionamento e perda, limitação de largura de banda, reordenação e duplicação de pacotes. O tráfego é interceptado de forma transparente para os sistemas terminais. As características de link desejadas são emuladas atrasando, descartando ou modificando pacotes no fluxo. Na figura 19 podemos observar o conteúdo de uma dada origem sendo alterada para três clientes de forma distinta.

Delay (Média):	8.000 ms
Delsigma (Desvio Padrão – Jitter):	5.000 ms
Drop%:	4.9999
Dup%:	0.9999
Bandwidth:	50.000 bytes/seg

Tabela 3 - Parâmetros do Emulador de Rede

Assim, o processo pode ser resumido da seguinte forma, como observado na figura 20:

1- O *Player* envia uma requisição de dados para o servidor;
2- O servidor inicia a transmissão de dados, que é interceptada pelo emulador de rede;
3- O emulador de rede modifica a transmissão, limitando a largura de banda, adicionando atraso, perda entre outros fatores de acordo com sua configuração;
4- Os dados chegam ao cliente, e passam pelo buffer de equalização;
5- A seguir, os dados são armazenados no buffer de armazenamento e começam a ser apresentados;
6- No momento apropriado mais dados serão requisitados, dependendo da ocupação do buffer;

A qualquer momento o usuário pode utilizar os controles disponíveis no Player e todas as notificações são realizadas através to PlayerListener.

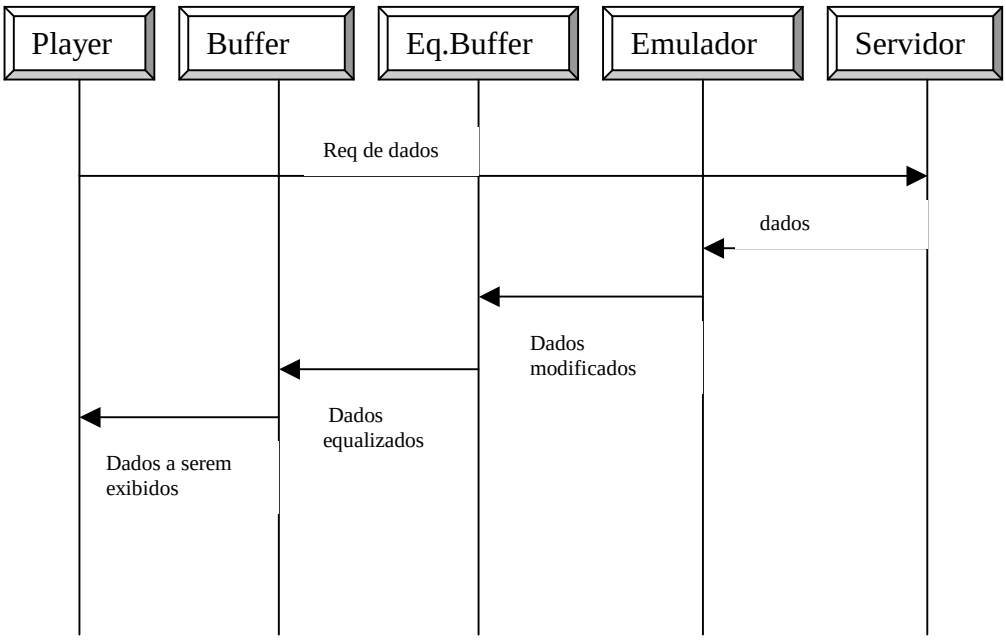

Figura 20 - Diagrama de Sequência

Maiores informações sobre as ferramentas de suporte podem ser encontradas no anexo A.

5.4 Resultados

A figura 20 mostra os resultados dos testes para uma aplicação de vídeo sobre demanda desenvolvida sobre a API JSR-135 utilizando o sistema de buffers do framework desenvolvido. Podemos notar que a chegada dos dados se inicia em t1 e a reprodução em t2, interrompendo a chegada de dados. No instante t3 ocorre uma interrupção da reprodução dos dados, causada por um esvaziamento de buffer. A aquisição de dados é reiniciada, porém a interrupção

permanece até o instante t4. Este fato ocorreu, pois a API original não possui a capacidade de realizar reprodução e aquisição de dados simultaneamente. A reprodução é reiniciada em t4, momento em que dados suficientes foram introduzidos no buffer.

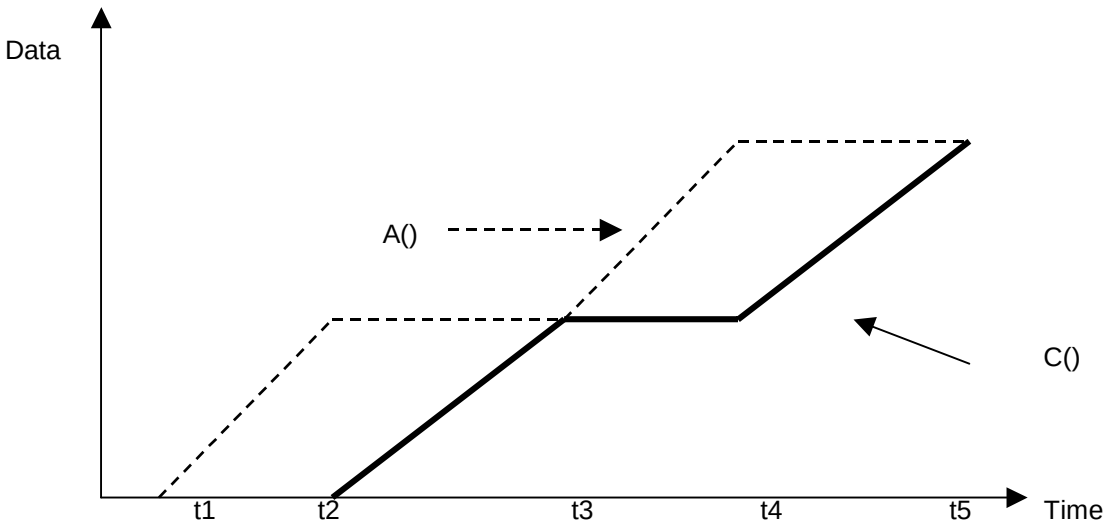

Figura 21 - JSR135 Utilizando Componentes do Framework.

Considerando um cenário onde o controle de fluxo é gerenciado por um servidor de vídeo, a aplicação acaba perdendo os dados que foram enviados durante a reprodução, pois a aplicação não é capaz de armazená-los.

O padrão observado na figura 20 ocorre recursivamente, indicando que a aplicação interrompe o processo de aquisição de dados ao iniciar o processo de reprodução. Para minimizar os efeitos dessa interrupção, o tamanho do buffer pode ser drasticamente reduzido, como mostra a figura 21. Como resultado, a reprodução dos dados não demonstra grandes intervalos de

interrupção, porém apresenta um desempenho ineficiente e pouco apropriado para este tipo de aplicação. Por outro lado, utilizando o cenário de gerenciamento de fluxo pelo servidor, os resultados são equivalentes a uma apresentação com baixa taxa de dados, onde, apesar de realizar uma apresentação compreensível, acaba desabilitando a função principal do buffer de prevenir interrupções.

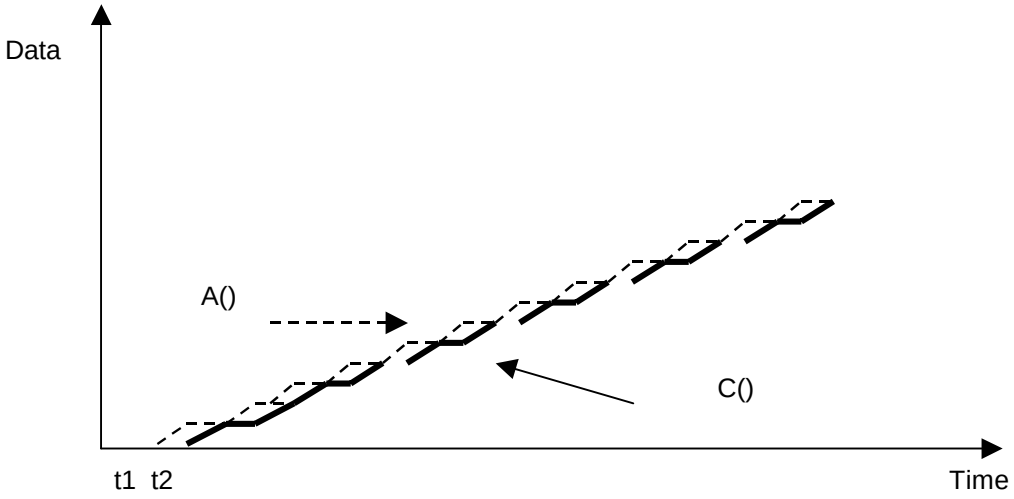

Figura 22 - Minimizando o Tamanho do Buffer.

A figura 22 mostra o resultado para a melhor utilização do framework desenvolvido, que faz uso da funcionalidade do sistema de buffers em conjunto com a API alterada, capaz de realizar a aquisição de dados e a reprodução contínua de forma independente, evitando os problemas de *buffer overflow* e *buffer underflow* no decodificador.

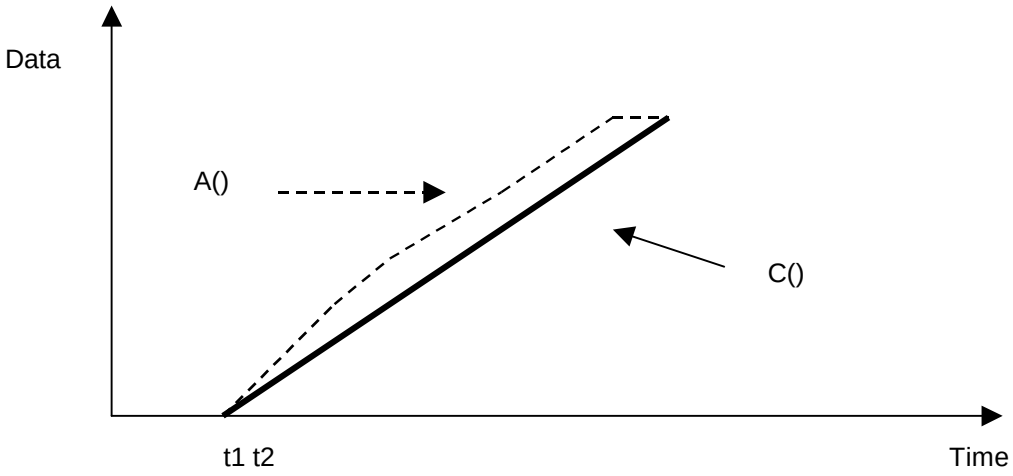

Figura 23 - Visão Qualitativa do Serviço Final

Esta visão não permite a análise detalhada das informações, pois se trata apenas de uma amostra qualitativa. O tamanho do buffer pode ser ajustado para as necessidades e características de cada dispositivo. As informações como o buffer mínimo e recomendado para uma mídia específica serão apresentadas a seguir.

5.5 Análise de Resultados

Nessa sessão serão mostrados os resultados quantitativos extraídos com a utilização de uma ferramenta de monitoramento de memória e a análise dos dados adquiridos através do uso de uma ferramenta de análises estatísticas. Maiores informações sobre as ferramentas podem ser encontradas no anexo A.

Durante a execução dos testes descritos anteriormente a ocupação do buffer de memória foi monitorada e cinqüenta amostras foram retiradas em intervalos iguais de tempo. Os valores individuais das amostras foram analisados através de análise histográfica, teste de normalidade, análise de capacitação e verificação de controle. Os dados analisados estão descritos abaixo. Os principais resultados obtidos são a média da ocupação do buffer que foi de 559.371 bytes, o desvio padrão dos valores de 246.982 bytes, a amplitude de 924.622 bytes entre outras informações.

Descriptive Statistics: Buffer Ocupation

Variable	N	N*	Mean	SE Mean	StDev	Minimum	Q1	Median
Buffer Ocupation	50	0	559371	34929	246982	45878	374945	547345

Variable	Q3	Maximum	Range
Buffer Ocupation	771650	970500	924622

O quadro acima apresenta os resultados estatísticos referentes à análise das 50 amostras inseridas. O histograma foi gerado utilizando 5 barras (figura 23), pois apresentou a melhor visualização da ocupação, mostrando claramente que os valores mínimo e máximo não ultrapassaram 0 e 1000000 bytes, respectivamente (45.8772 e 970.500 para ser mais preciso).

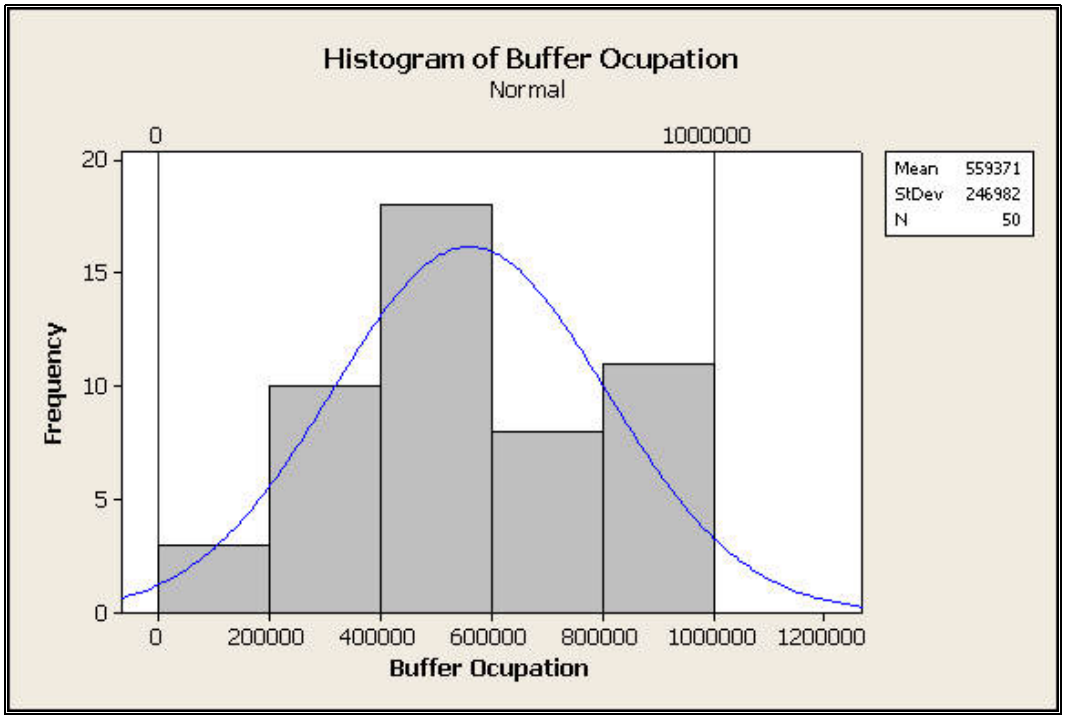

Figura 24 - Histograma da Memória Utilizada

O padrão normal foi adicionado ao histograma da figura 23. Após a comparação com diferentes quantidades de barras a normalidade pôde ser comprovada observando o gráfico da figura 24 que apresenta o teste de normalidade, onde encontramos um P-Value de 0,520.

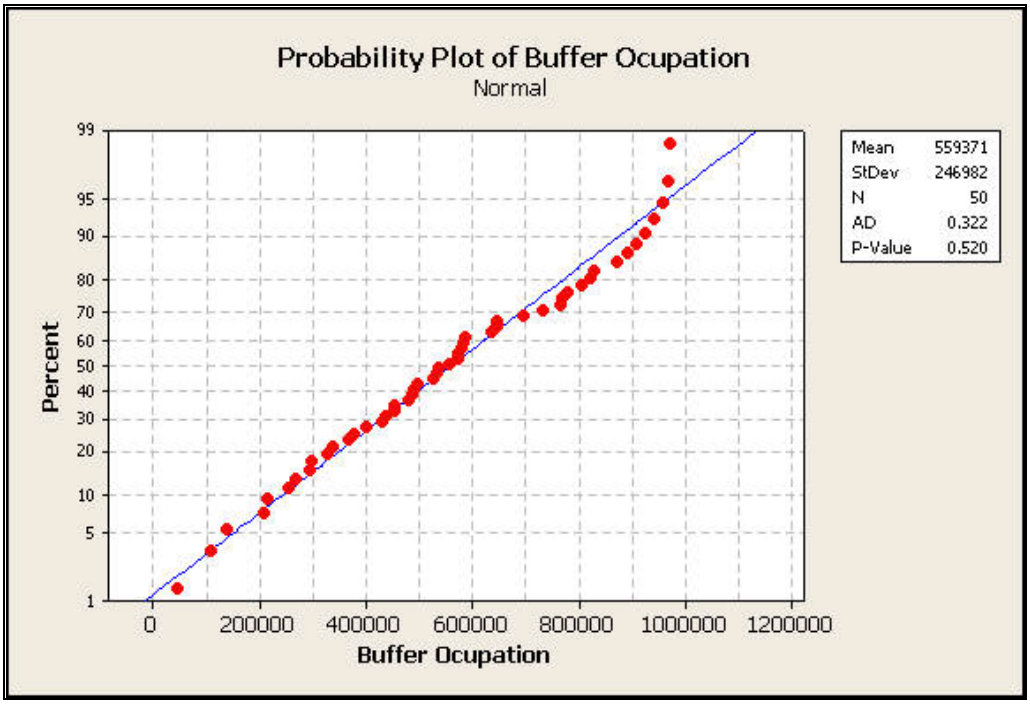

Figura 25 - Teste de Normalidade

A análise do gráfico de controle na figura 25 mostra que o processo de aquisição e reprodução dos dados está sob controle, ou seja, o valor máximo de 970.500 bytes está abaixo do limite de controle superior (UCL) de 1.372.110 bytes e o valor mínimo está acima do limite de controle inferior (LCL) de – 253.368 bytes.

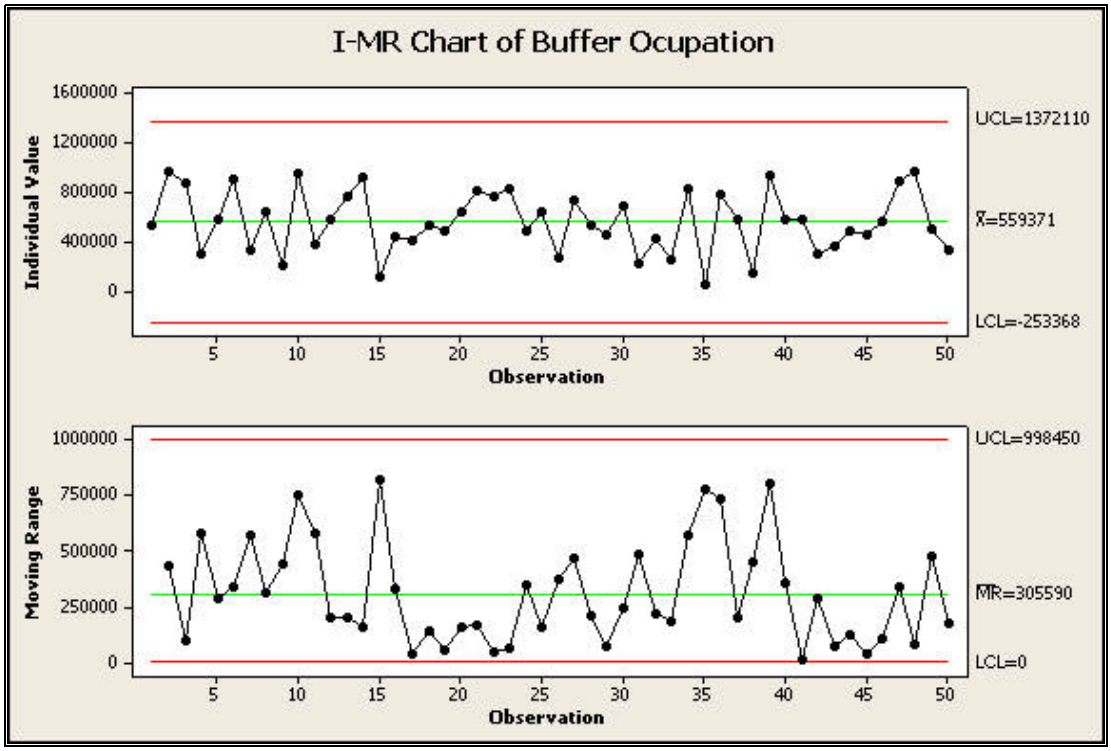

Figura 26 - Valores Individuais da Utilização da Memória

Os resultados da figura 26 mostram que apesar da amplitude apresentada ter sido de 924.622 bytes, a análise da capacidade indica que o espalhamento geral de controle é maior. Isso demonstra que apesar da utilização de um buffer de 1 Mbytes ter sido suficiente, uma alocação maior seria mais aconselhável para suportar eventuais mudanças bruscas.

Notamos a existência de picos, onde o valor máximo alcançado foi de 970500 bytes, e na seqüência quedas causadas principalmente pela ação da aceleração da reprodução acionada a uma taxa de 85% de ocupação, que visa impedir um buffer overflow.

Outra informação importante que podemos observar é o seu centro aparece deslocado em relação às linhas do limite de especificação inferior (LSL) e do limite de especificação superior (USL), o que também pode ser observado pelo valor negativo da linha LCL dos valores individuais da figura 25.

Essa informação demonstra que apesar do processo ser efetivo ele ainda pode ser ajustado para um melhor desempenho em casos com maiores variações.

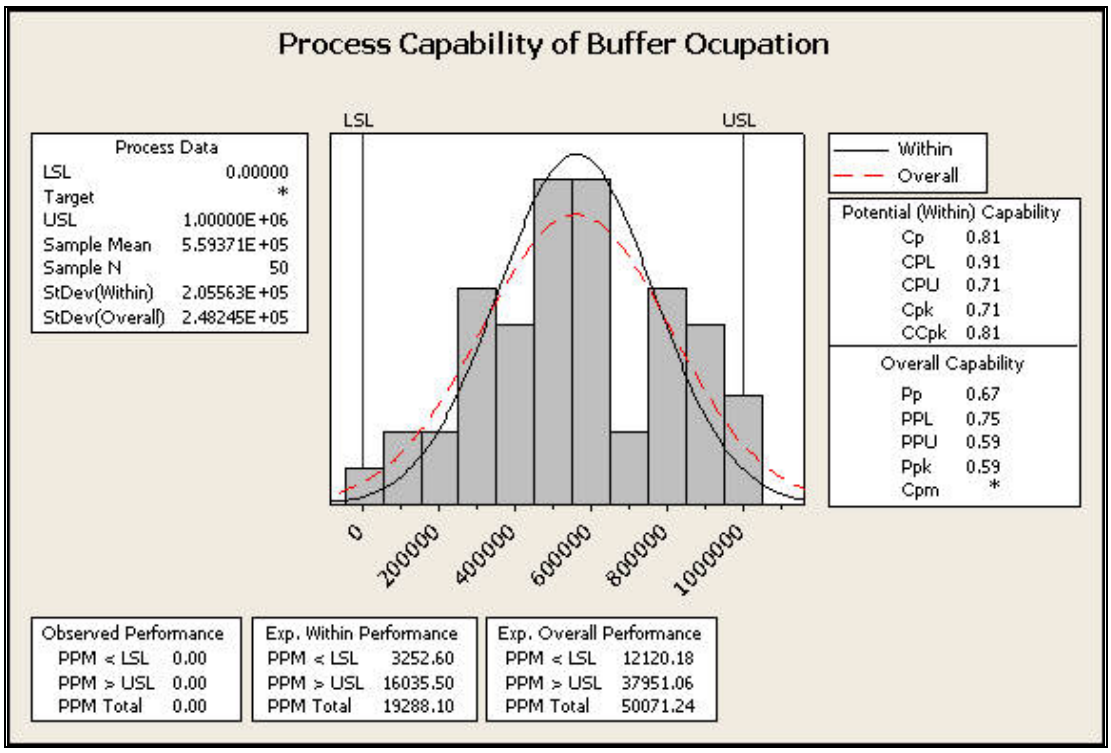

Figura 27 - Análise de Capacitação

A seguir, um diagrama Boxplot foi criado. O Boxplot permite uma boa visualização do “espalhamento” dos valores, média, mediana, quartis e “anomalias” (não encontradas durante os testes). Notamos mais facilmente que o centro de valores está deslocado. Apesar de não se tratar de um problema, o controle do buffer pode ser ajustado utilizando uma melhor definição para sua adaptação automática.

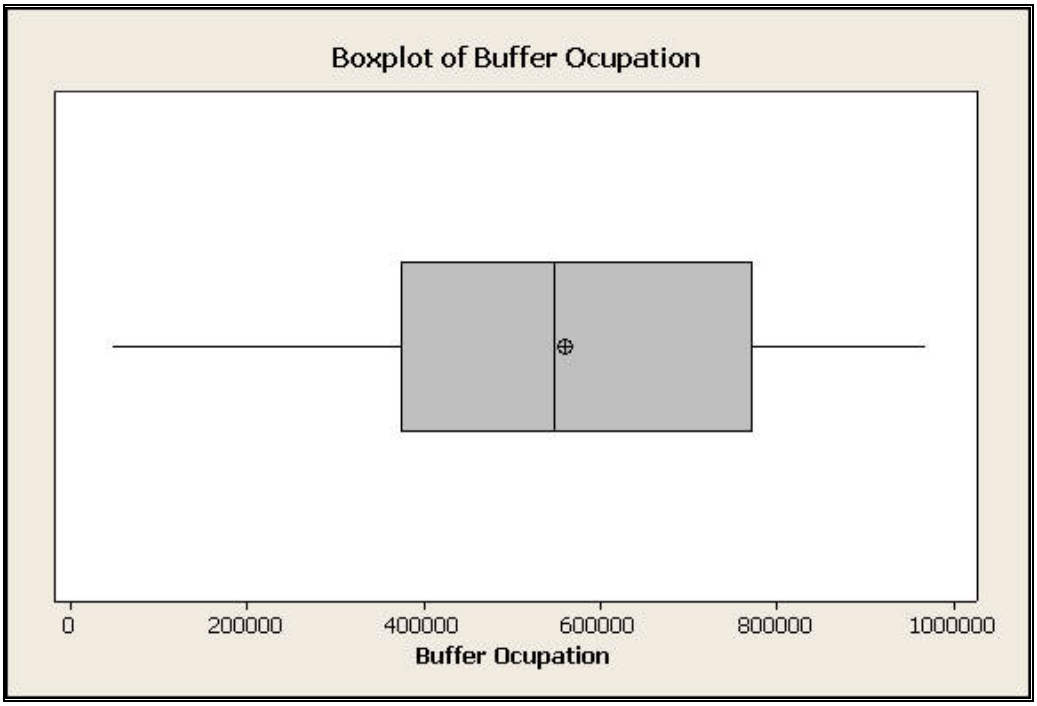

Figura 28 - Boxplot da Memória Utilizada

A figura 28 mostra a análise de capacitação. A análise indica que o espalhamento geral de controle é maior que a amplitude encontrada durante os testes e seu centro aparece deslocado, o que também pode ser observado nas demais figuras. Isso demonstra que apesar da utilização de um buffer de 1 Mbytes ter sido suficiente, pode não ser suficiente em alguns casos.

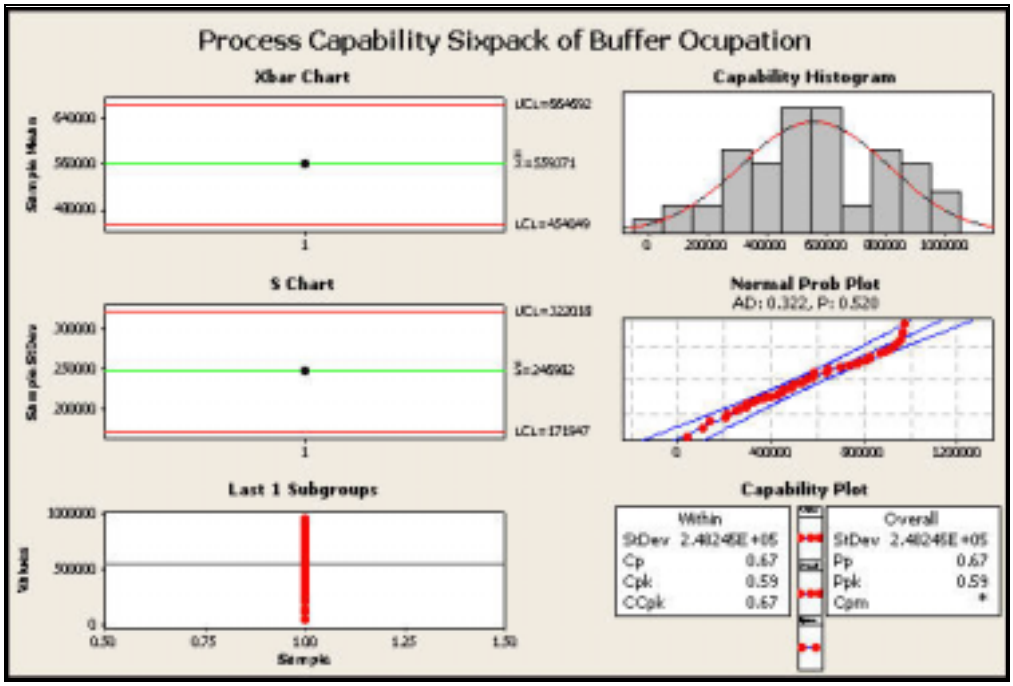

Figura 29 - Análise de Capacitação "Sixpack"

Toda a análise realizada veio a comprovar o bom desempenho da aplicação criada, demonstrando que já é possível uso de aplicações multimídia de tempo real em dispositivos móveis com recursos limitados. Além disso, a análise permitiu também a detecção de possibilidade de melhorias e trabalhos futuros que serão apresentados a seguir.

Capítulo 6

Conclusões

Este Trabalho apresentou um Framework que possibilita a criação de aplicações multimídia de tempo real para serem utilizadas em dispositivos móveis com poder computacional limitado, notadamente celulares. Estes dispositivos estão capacitados atualmente a reproduzir informações multimídia previamente armazenadas e a acessar dados através de redes de comunicação sem fio, porém não realizando essas funcionalidades em conjunto e em tempo real. O uso em conjunto possibilita a criação de uma maior variedade de aplicações, além de possibilitar a visualização de uma quantidade maior de dados do que seria possível armazenar de uma só vez em memória, porém exige um gerenciamento mais complexo dos recursos computacionais envolvidos.

O Framework apresentado é formado pelos seguintes componentes: um Player para reprodução dos dados, Controles que permitem maior manipulação da informação, Datasource para criar e controlar as conexões, Listener para tratamento de eventos assíncronos, Buffer de Equalização de atraso e um Buffer de Armazenamento Temporário para evitar perda de dados.

O desenvolvimento foi realizado utilizando a tecnologia J2ME, o que permite a utilização do Framework em praticamente todos os dispositivos “high end” atuais.

Com a popularização do “Open Source”, grandes empresas de desenvolvimento iniciaram pesquisas para a criação de dispositivos compatíveis com as demais tecnologias existentes, possibilitando ao usuário final a capacidade de personalização de seus equipamentos. A plataforma J2ME especifica uma configuração (CLDC) e um perfil (MIDP) que quando

combinados definem uma plataforma uniforme para que qualquer fabricante de dispositivos possa dar suporte e qualquer aplicação desenvolvida possa ser executada em uma grande variedade de dispositivos móveis.

Apesar de possibilitar a criação de aplicações que ocupem uma pequena porção do espaço de armazenamento do dispositivo móvel, disponibilizando uma grande variedade de recursos, pouco foi desenvolvido em relação a aplicações multimídia devido à falta de suporte para esse tipo de informação.

Com a recente criação da versão MIDP2.0 e da API multimídia, esses dados podem atualmente ser apresentados em dispositivos com grandes restrições de recursos. Porem, técnicas de QoS ainda precisam ser estudadas e incorporadas a essa plataforma para possibilitar a aquisição de dados por aplicações de tempo-real.

Modificações foram realizadas na API multimídia disponível, a JSR-135, de forma a possibilitar o uso de aplicações de tempo real, conforme mostrado nos capítulos anteriores. Em resumo, elas permitem que o gerenciamento de requisição de dados possa ser realizado tendo como base à ocupação atual do buffer. Cada nova requisição inicia um novo ciclo de aquisição (realize), prefetch e reprodução.

Uma aplicação de vídeo foi desenvolvida com os componentes do framework e testes foram executados através de um ambiente que emula as condições da rede e as características de um dispositivo móvel. As condições do sistema de buffer foram monitoradas e os resultados foram analisados com o uso de softwares estatísticos, demonstrando um desempenho suficiente para sua execução.

Assim, este trabalho apresentou o desenvolvimento de componentes que auxiliam o acesso a informações multimídia a qualquer momento e de qualquer lugar, facilitando o desenvolvimento de aplicações multimídia e permitindo o gerenciamento de políticas que governam o comportamento das aplicações.

Trabalhos Futuros

Acreditamos que as pesquisas visando uma melhor utilização de aplicações multimídia em dispositivos móveis serão ainda mais elaboradas culminando em uma completa difusão desse tipo de tecnologia. Nesse sentido, o framework desenvolvido é apenas um primeiro passo para esse avanço.

Outro passo fundamental está relacionado à evolução dos dispositivos móveis. A melhoria e aperfeiçoamento de suas capacidades computacionais tornaria viável o uso de outras técnicas de melhoria de QoS, principalmente no tocante ao gerenciamento e à adaptação de parâmetros. Juntamente com essa nova funcionalidade é possível adicionar uma maior interação com usuários, oferecendo uma interface de configuração de qualidade para usuários avançados.

Os trabalhos nessa área podem prosseguir com o estudo do uso do framework em conjunto com os mecanismos de QoS citados em 3.8, principalmente com enfoque em melhorias no meio de comunicação, servidores inteligentes ou ainda o uso de *proxies*.

Com a chegada de novos dispositivos e o atraso do desenvolvimento da TV Digital, cujo sistema oficial será decidido apenas em 2005 e implantação em 2007, a pesquisa da TV pelo celular se tornou o centro das atenções atualmente, mesmo com os fatores limitantes do uso de conexão fim-a-fim e transmissão paga pelo usuário.

Referências Bibliográficas

[1] Aurrecoechea, C.; Campbell, A.T. and Hauw L. **A Survey of QoS Architectures**, Multimedia Systems Journal , Special Issue on QoS Architecture, 1997.

[2] Campbell, A . T. and Coulson G., **QoS Adaptive Transports: Delivering scalable Media to the Desktop**, IEEE Network, March/April 1997.

[3] Guedes L. A e Cardozo E., **Especificação de um Protocolo para Negociação de Qualidade de Serviço em Sistemas Multimídia**, Unicamp e Universidade Federal do Pará, 1997.

[4] IFIP Fifth International Workshop on Quality of Service (IWQOS '97) - **Building QoS into Distributed Systems** - May 21-23, 1997. Disponível em: <http://www.ctr.columbia.edu/iwqos/>. Acesso em 15 ago 2001.

[5] Special Interest Group on QoS. Disponível em: <http://www.fokus.gmd.de/step/employees/jdm/jdmQoSIG.html>. Acesso em 23 set 2002.

[6] IETF. **Slides from IETF meeting 31**, Integrated Service Working Group, Disponível em: <ftp://mercury.lcs.mit.edu/pub/intserv>. Acesso em mar 2002.

[7] Nahrsted K. e Smith J., **The Broker QOS**, IEEE Multimedia, Spring 1995.

[8] Vogel A., Kerhervé B., Bocchmann G., Gecsei J., **Distributed Multimidia and QOS: A Survey**, IEEE Multimedia Summer 1995.

[9] Clark, D., Braden R.,and Shenker S., **Integrated Services in the Internet Architecture: an Overview**, Request for Comments, RFC-1633, 1994. Disponível em: <http://ds.internic.net/rfc/rfc1633.txt>. Acesso em fev 2001.

[10] Kerhervé B. et al.,**On Distributed Multimedia Presentacional Applications: Functional and Computational Architecture and QOS Negotiation**, Proc. 4th Int'l Workshop on protocols for High-Speed Networks, Chapman & Hall, London, 1994, pp. 1-17.

[11] Vogel A., Kerhervé B., Bocchmann G., Gecsei J., **On QoS Negociation in Distributed Multimidia Application**, Proc. Protocol for High Speed Networks, April 1994.

[12] Delgrossi L. et al.,**Media Scalling for Audiovisual Communication with the Heidelberg Transport System**, Proc. ACM Multimedia 93, ACM Press, New York, 1993, pp. 99-104.

[13] Elliot, C. **High-Quality Multimedia Conferencing Through a Long-Haul Packet Network**, Proc. 1st ACM Int'l Conf. On Multimedia, ACM Press, New York, 1993, pp. 91-98.

[14] Herrtwich R. G., **The Role of Performance, Scheduling, and Resource Reservation in Multimedia Systems**, Operating Systems of the 90s and Beyond, A Karshmer and J. Nehmer, eds. Springer-Verlag, Berlin, 1991, pp. 279-284.

[15] Bechler, M; Ritter, H; Schiller, J. H. **Quality of Service in Mobile and Wireless Networks : The Need for Proactive and Adaptive Applications.** Reprint 33nd. Annual IEEE Hawaii International Conference on System Sciences, HICSS, 2000.

[16] Cheong, F.; Lai, R. **QoS specification and mapping for distributed multimedia systems : A survey of issues**. Elsevier – The Journal of Systems and Software*,* 45, 1999, pp. 127-139.

[17] Haf, A; Bochmann, G v. **Quality-of-service adaptation in distributed multimedia applications**. Springer/Verlag – Multimedia Systems, 6,1998, pp. 299-315.

[18] STARDUST.COM. **QoS protocols & architectures**. *White Pape*r. Disponível em: <www.qosforum.com>. Acesso em 8 set1999.

[19] SUN MICROSYSTEMS, **Mobile Information Device Profile**, JCP Specification. Disponível em: <http://java.sun.com/aboutJava/communityprocess/final/jsr037/MIDPSpec_1_0.zip>., 2000

[20] SUN MICROSYSTEMS, **MIDP APIs for Wireless Applications**, 2001 Disponível em: <http://java.sun.com/products/midp/midp-wirelessapps-wp.pdf>. Acesso em 10 out 2001.

[21] SUN MICROSYSTEMS, **Applications for Mobile Information Devices**, 2000 Disponível em: <http://java.sun.com/products/midp/midpwp.pdf> Acesso em 10 mar 2000.

[22] IBM, **J2ME grows up**, 2001. Disponível em: <http://www-106.ibm.com/developerworks/java/library/j-j2me/index.html>. Acesso em 10 mar 2001.

[23] Paal, P., **Java 2 Micro Edition, M.Sc. thesis**, HUT, 2001. Disponível em: <http://www.hut.fi/~opaal/netsec/j2me.html>. Acesso em 10 mar 2001.

[24] IETF. **Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types**, Disponível em: <http://www.ietf.org/rfc/rfc2045.txt>. Acesso em 10 mar 2001.

[25] Ott M., Michelitsch G., Reininger D. and Welling G., **An Architecture for Adaptive QoS nd its Application to Multimedia Systems Design**, Special Issue of Computer Communications on Building Quality of Service into Distributed Systems, 1997.

[26] Guojun L, **Communication and Computing for Distributed Multimedia Systems**, 1996, pp.117-121.

[27] Neureiter G., Burness L., Kassler A., Khengar P., Kovacs E., Mandato D., Manner J., Robles T., Velayos H.. **The Brain Quality of Service Architecture for Adaptable Services with Mobility Support**, 2000.

[28] Noble B., Satyanarayanan, M. A. **Programming Interface for Application-Ware Adaptation in Mobile Computing**. Computing Systems 8, Fall, 1995.

[29] Noble B., **System suport for mobile, adaptive applications**, IEEE Personal Communications 7(1) , 2000, pp. 44-49.

[30] Globus Web Page. Disponível em: <http://www.globus.org>. Acesso em 10 jun 2003.

[31] Legion Web Page. Disponível em: <http://legion.virginia.edu>. Acesso em 10 jun 2003.

[32] MSHN Web Page. Disponível em: <http://www.mshn.org>. Acesso em 10 jun 2003.

[33] Casner S. and Deering S. **First IETF Internet Audiocast**. ACM Computer Communication Review, 22:92-97, July 1992.

[34] McCanne S. and Jacobsen V. **vic: A Flexible Framework for Packet Video**. In Proc. of ACM Multimidia, pages 511-522, San Francisco CA, Nov. 1995.

[35] Perkins C., Hardman V., Kouvelas I., and Sasse A. **Multicast Audio: The Next Generation**. In Proc. INET 97, Kuala Lumpur, Malaysia, June 1997.

[36] Campbell A., Coulson G., Garcia F., Hutchinson D., and Leopold H. **Integrated Quality of Service for Multimedia Communications**. In Proc. IEEE INFOCOM`93, San Francisco, Mar. 1993

[37] Nahrsted K. and Smith J. **A Service Kernel for Multimedia End points**. In Proc. IWACA `94, Sept. 1994.

[38] Vogt C., Wolf L. C., Herriwich R. G., and Wittig H. **HeiRAT** – **Quality-of-Service Management for Distributed Multimedia Systems**. Special Issue on QoS Systems of ACM Multimedia Systems Journal, 6(3):152-166, May 1998.

[39] Fry M., Seneviratne A., Vogel A., and Witana V. **Delivering QoS Controlled Continuous Media on the World Wide Web**. In Proc. IWQoS`96, pages 115-124, 1996.

[40] Lakshman K. and Yavatkar R. **AQUA: An Adaptive End-System Quality of Service Architecture. High-Speed Networking for Multimedia Applications**, Nov. 1996.

[41] Campbell A., Coulson G. and Hutchison D., **Supporting Adaptive Flows in a Quality of ervice Architecture**, ACM Multimedia Systems Journal 1996.

[42] Wittmann R. and Zitterbart M., **AMnet: Active Multicasting Networ*k***, Proc. of IEEE Intern. Conf. on Communications (ICC'98), Atlanta, GA, USA, June 1998.

[43] Hafid A., Fischer S.: **A Multi-Agent Architecture for Cooperative Quality of Service Management**, in Management of Multimedia Networks and Services Proceedings of MMNS'97), pages 41-54, Chapman & Hall, London, 1998.

[44] ISO, **Quality of Service Framework**, ISO/IEC JTC1/SC21/WG1 N9680, International Standards Organisation, UK, 1995.

[45] Ferrari, D., **The Tenet Experience and he Design of Protocols for Integrated Services Internetworks**, Multimedia Systems Journal, November 1995.

[46] Benerjea, A. and Mah B., **The Real-Time Channel Administration Protocol**, Second International Workshop on Network and Operating System Support for Digital Audio and Video", Heidelberg, November 1991.

[47] Wolfinger, B. and Moran M., **A Continuous Media Data Transport Service and Protocol for Real-time Communication in High Speed Networks**. Second International Workshop on Network and Operating System Support for Digital Audio and Video, IBM ENC, Heidelberg, Germany, 1991.

[48] SUN MICROSYSTEMS. **Mobile Media API (JSR-135) Specification**: Disponível em: <http://jcp.org/jsr/detail/135.jsp>. Acesso em 10 ago 2003.

[49] SUN MICROSYSTEMS. **MIDP 2.0 (JSR-118) Specification**: Disponível em: <http://jcp.org/jsr/detail/118.jsp>. Acesso em 10 ago 2003

[50] Tanembaum A. S., **Computer Networks**, 3a. Edição, 1996.

[51] ISO/IEC IS 13818 – **MPEG-2: Generic coding of moving pictures and audio information**. 2003

[52] Tourad Ebrehimi, **MPEG-4 Video Verification Model: A video encoding/decoding based on content representation**, Instituto Federal de Tecnologia da Suiça, Lausanne. 2002.

[53] Hess C.K., Raila D., Campbell R.H. and Mickunas D., **Design and performance of MPEG video streaming to palmtop computers**, November 2002.

[54] Lloyd-Evans, Robert, **QoS in integrated 3G networks**, Artech House mobile communications series, 2002, pp. 261-277.

[55] TELEFONAR.INFO Web Page, Disponível em: <http://www.telefonar.info>. Acesso em 14 abr 2004.

Anexo A – Ferramentas de Suporte

A plataforma Eclipse foi projetada para oferecer ambientes de desenvolvimento integrado (IDEs) que possam ser utilizados para a criação de aplicações diversas, como web sites, programas Java *embedded*, programas C++ e Enterprise JavaBeans.

A plataforma é uma proposta de consórcio de empresas que apóiam o uso de uma arquitetura aberta para a criação de ambientes integrados de desenvolvimento (IDEs), onde a indústria de software possa desenvolver diversos programas, aplicativos e ferramentas, de forma otimizada e padronizada, baseando-se nas iniciativas de software livre.

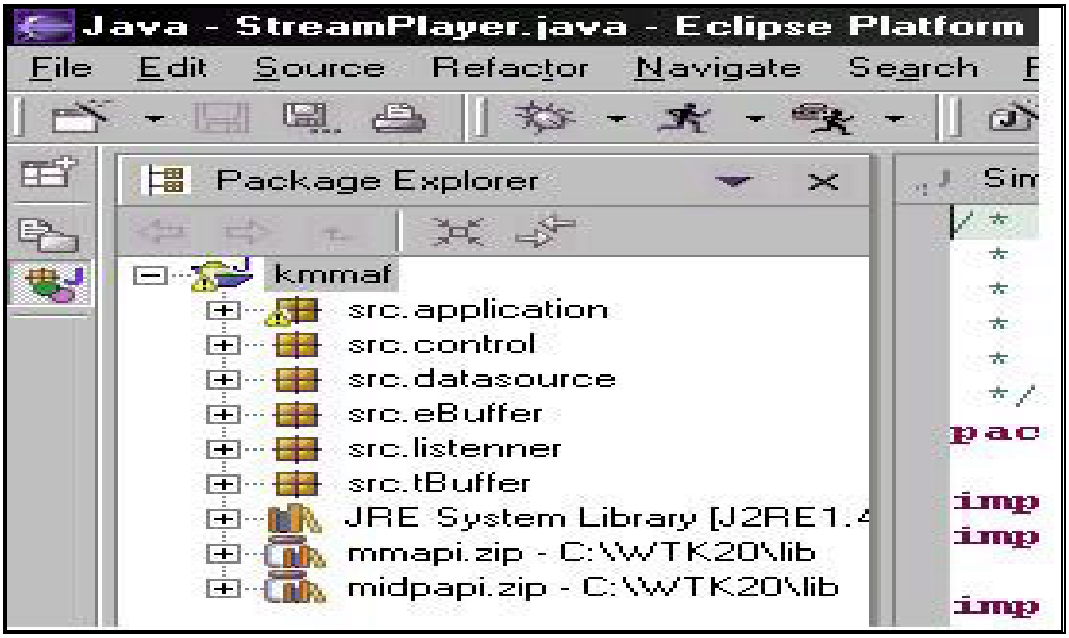

Figura 30 - Plataforma Eclipse de Suporte a Desenvolvimento

Embora a plataforma apresente muitas funcionalidades, a maioria é genérica de modo a suportar uma maior variedade de aplicações. A plataforma utiliza ferramentas adicionais de modo a ser estendida através de módulos executáveis chamados *plug-ins.* Dentre eles podemos destacar *plug-ins* para gerenciamento de versões como ClearCase e CVS e ainda modelagem UML.
www.eclipse.org

O Wireless Toolkit da plataforma Java 2 Micro Edition, (J2ME) é um conjunto de ferramentas que oferecem aos desenvolvedores de aplicações mecanismos para emulação de ambientes, documentação e exemplos necessários para o desenvolvimento de aplicações na tecnologia Java destinadas a telefones móveis que fazem uso da tecnologia CLDC/MIDP.

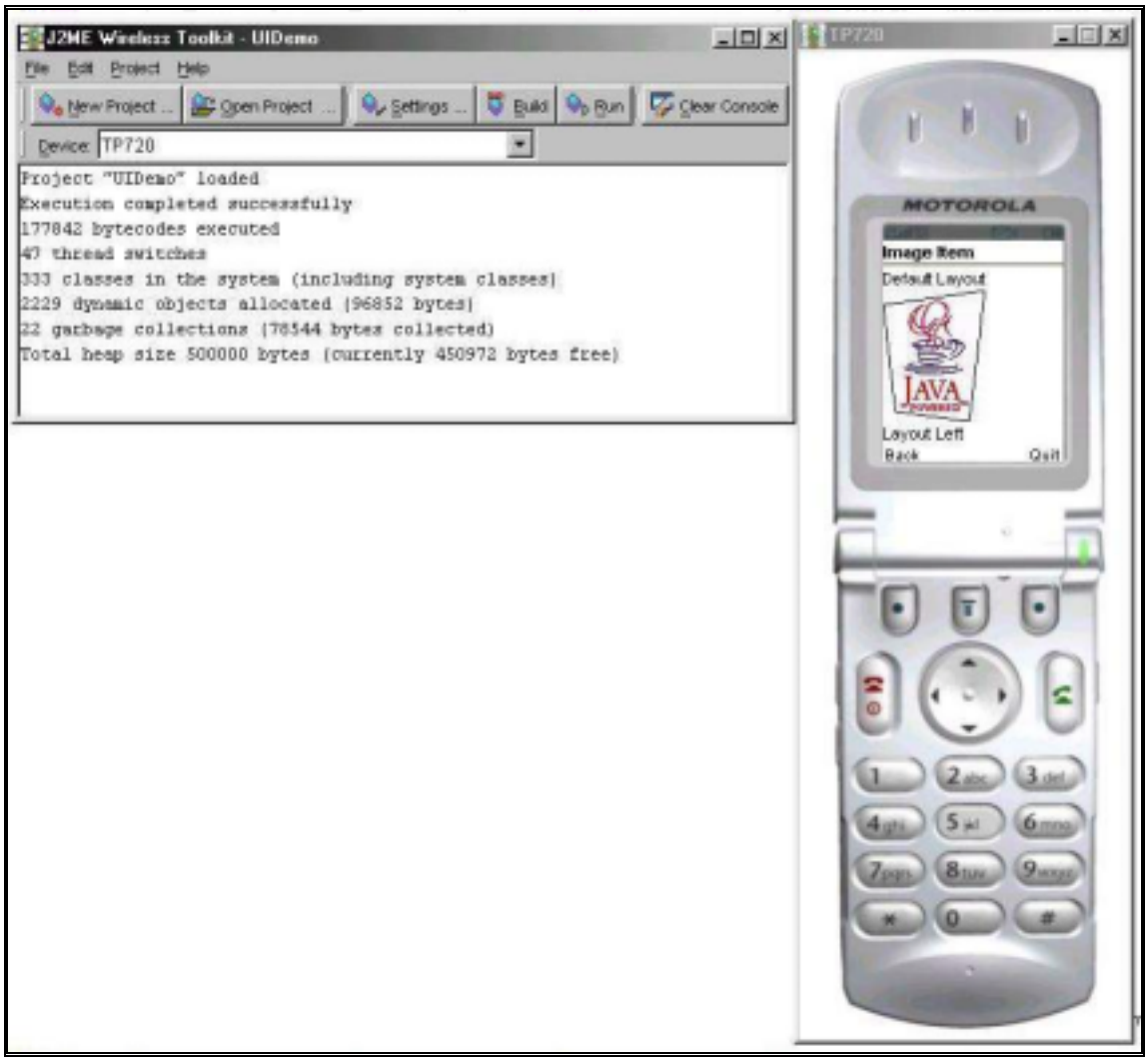

Figura 31 - Wireless Toolkit

A ferramenta permite a criação de pacotes dos arquivos de projeto ou ainda um pacote “ofuscado” ou codificado para proteger o código de uma possível decompilação. Outro benefício do uso desta função é que o processo reduz o tamanho do bytecode, resultando em um arquivo JAR de menor tamanho.

A ferramenta possui também medidores de desempenho, com resultados detalhados ou ainda gráficos.

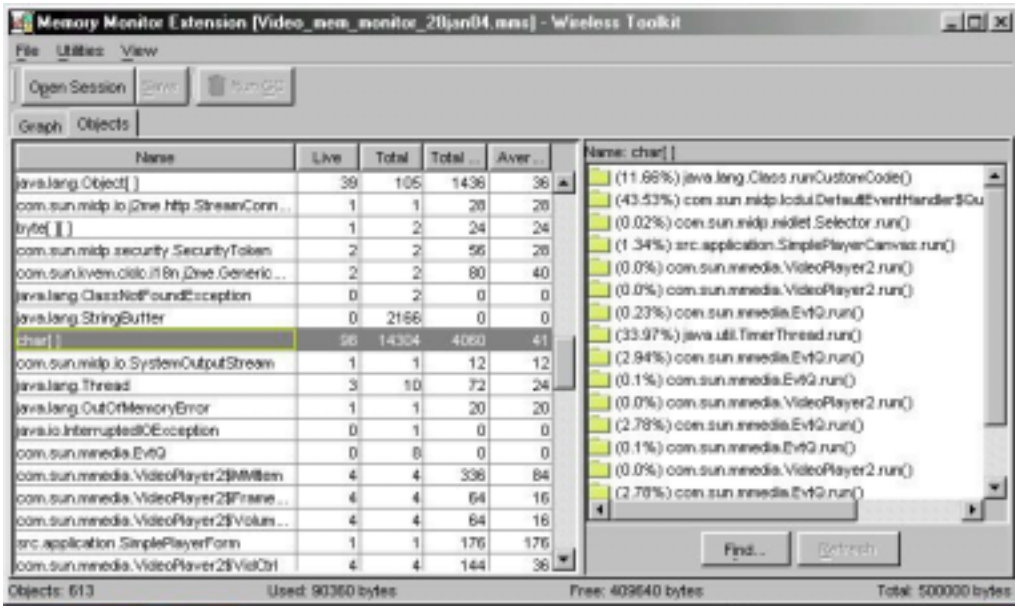

Figura 32 - Monitoramento de Memória

O emulador de rede é uma ferramenta de propósito geral para emulação do desempenho de redes IP. A ferramenta é projetada para permitir experimentos controlados com aplicações sensíveis/adaptativas e protocolos de controle para redes, em um simples laboratório. Operando no nível IP, é possível emular características fim-a-fim críticas para os diversos tipos de redes.

http://www.nist.gov

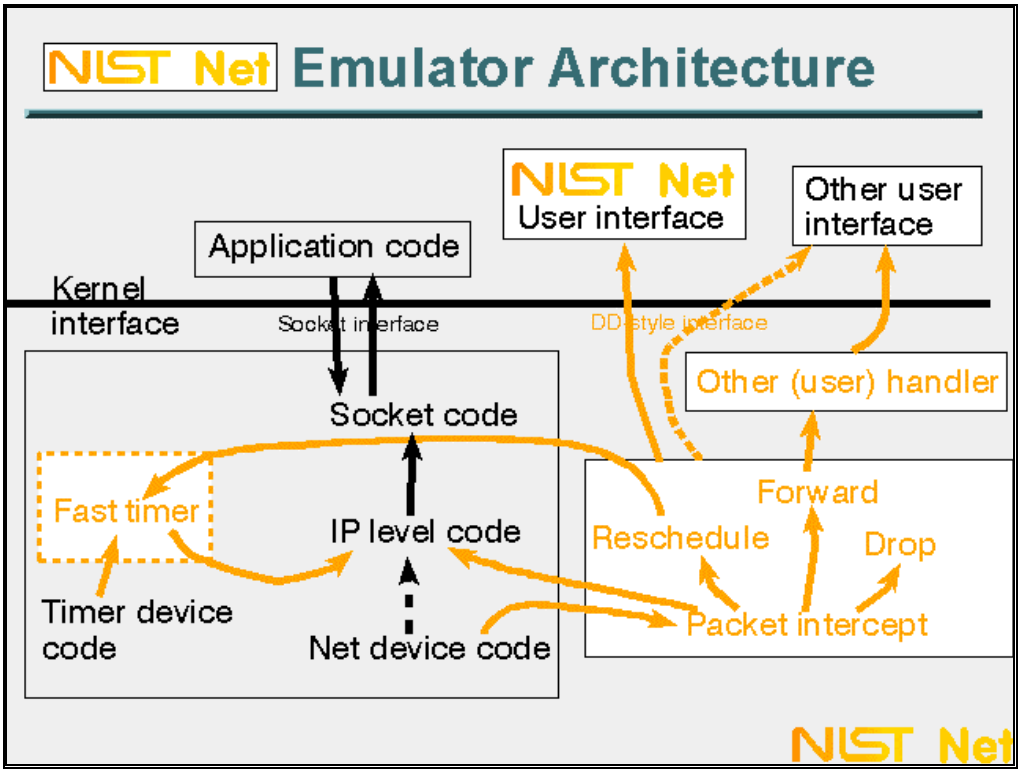

Figura 33 - Arquitetura do Emulador de Rede

O software estatístico MINITAB é o pacote ideal para o processo Six Sigma e os demais projetos da melhoria de qualidade. O software oferece os métodos necessários para a execução de cada fase do projeto da qualidade, desde o controle do processo estatístico à criação de experimentos.

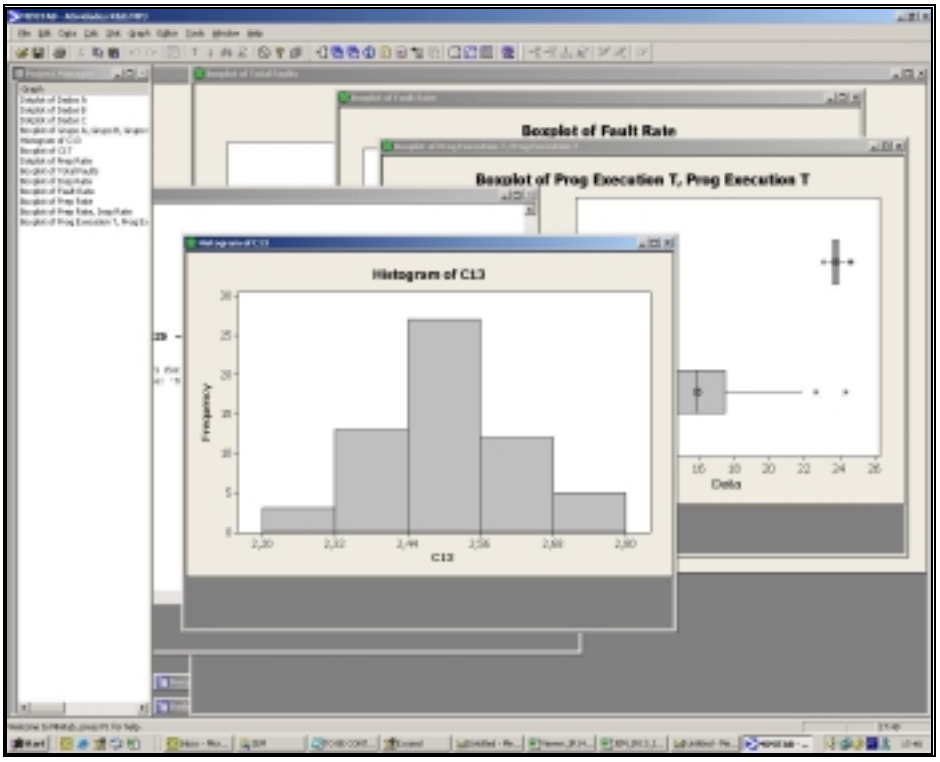

Figura 34 - Software Estatístico

Anexo B – QoS-KMMAF

Sistema de Buffers

Um PullSourceStream personalizado utiliza um buffer circular para melhoria da reprodução dos dados. A classe possui como atributos o input stream atual, uma flag para fim-de-arquivo, o tamanho da stream em bytes, a quantidade de bytes lidos até o momento, o tamanho padrão do buffer circular, o buffer circular, a posição atual no buffer, tamanho dos dados válidos no buffer, posição da cauda do buffer (dado a ser reproduzido), e um objeto auxiliar para sincronização.

```
private class XPullSourceStream implements PullSourceStream, Seekable {
 private InputStream m_in;
 private boolean  m_bEndOfStream;
 private long  m_nContentLength, m_nTotalRead;
 public static final int DEFAULT_WINDOW_SIZE = 524288;
 private final byte m_vWnd[];
 private int   m_nPos, m_nBuffer, m_nTail;
 private Object  LOCK = new Object ();
...
}
```

Um construtor para o objeto PullSourceStream com o tamanho padrão:

```
public XPullSourceStream () throws IOException
{
        this (DEFAULT_WINDOW_SIZE);
}
```

Um construtor para o objeto PullSourceStream com o tamanho desejado:

```
public XPullSourceStream (int nWindowSize) throws IOException
{
  m_in = m_isf.resetStream (null);
  m_nContentLength = m_isf.getContentLengthOfStream(null);
  m_nTotalRead = 0L;
  m_nPos = m_nBuffer = m_nTail = 0;
  m_vWnd = new byte[DEFAULT_WINDOW_SIZE];
}
```

Método para fechar o input stream.

```
public void close() throws IOException
{
        m_in.close();
}
```

Retorna “true” se e somente se não há dados disponíveis no input stream e a posição atual no buffer circular não é um dado válido.

```
public boolean willReadBlock()
{
        synchronized (LOCK)
        {
           try
           {
                return (m_nPos == m_nTail) && (m_in.available() == 0);
           } catch (IOException ex)
           {
                return true;
           }
        }
}
```

Retorna a diferença lógica entre a posição atual no buffer circular e a posição da cauda (dados válidos) no buffer circular.

```
private int posDistanceToTail ()
{
	if (m_nTail < m_nPos)
		return m_nTail + m_vWnd.length - m_nPos;
	else
		return m_nTail - m_nPos;
}
```

```
// Thread-safe
public boolean endOfStream() {
	boolean bRet;

	synchronized (LOCK) { bRet = m_bEndOfStream; }

	return bRet;
}
```

Adiciona ao buffer nLen bytes de dados e retorna o número de bytes que foram realmente adicionados.

```
private int fill (int nLen) throws IOException
{
	int	nResult = 0, nOrig, nPosDistanceToTail;

	nPosDistanceToTail = posDistanceToTail();
	nLen = Math.min (nLen, m_vWnd.length - nPosDistanceToTail);
	nOrig = nLen;

	while ((nLen > 0) && (-1 != (nResult = m_in.read (m_vWnd, m_nTail, nLen + m_nTail >
m_vWnd.length ? m_vWnd.length - m_nTail : nLen))))
	{
		nLen -= nResult;
		m_nTail = (m_nTail + nResult) % m_vWnd.length;
		m_nTotalRead += nResult;
	}

	if (nResult == -1)
		m_bEndOfStream = true;

	m_nBuffer = Math.min (m_nBuffer + (nOrig - nLen), m_vWnd.length);

	return nOrig - nLen;
}
```

```
public int read (byte[] vBuf, int nOffs, int nLen) throws IOException
{
   int     nOrig, nDist, nResult, nCopyLeft, nRet;

   synchronized (LOCK)
   {
          nOrig = nLen;

          while (nLen > 0)
          {
                    nDist = posDistanceToTail ();
                    if (nDist == 0)
                              nResult = fill (nLen);
                    else
                              nResult = Math.min (nDist, nLen);

                    if (vBuf != null)
                    {
                       int nCopied = (nOrig - nLen);

                       if (m_nPos + nResult > m_vWnd.length)
                       {
                              int                   nRemaining = m_vWnd.length - m_nPos;

                              System.arraycopy(m_vWnd, m_nPos, vBuf, nOffs + nCopied,
nRemaining);
                              System.arraycopy(m_vWnd, 0, vBuf, nOffs + nCopied + nRemaining,
nResult - nRemaining);
                       } else
                              System.arraycopy(m_vWnd, m_nPos, vBuf, nOffs + nCopied, nResult);
                       }
                       nLen -= nResult;
                       m_nPos = (m_nPos + nResult) % m_vWnd.length;
                    }

                    nRet = m_bEndOfStream && (m_nPos == m_nTail) ? -1 : nOrig - nLen;
                    }

                    return nRet;
          }
```

```
public long seek (long nWhere)
{
   long            nSkip, nActual = 0L, nRead, nRet;

   synchronized (LOCK)
   {
         try
         {
               if (nWhere > m_nTotalRead)
                     nSkip = nWhere - m_nTotalRead;
               else if (m_nTotalRead - nWhere <= m_nBuffer)
               {
                     int nDelta = m_nTail - (int)(m_nTotalRead - nWhere);

                     if (nDelta < 0)
                           m_nPos = m_vWnd.length + nDelta;
                     else
                           m_nPos = nDelta;
                     return nWhere;
               }
               else
               {
                     for (int c = 0; c < nWhere % 8; c++)
                           m_in.read();

                     m_in = m_isf.resetStream(m_in);
                     m_nTotalRead = 0L;
                     m_bEndOfStream = false;
                     m_nTail = m_nPos = m_nBuffer = 0;
                     nSkip = nWhere;
               }

               for (long nIter = (nSkip - nActual); nIter > 0; nIter -= nRead)
               {
                     nRead = read (null, 0, nIter < Integer.MAX_VALUE ? (int)nIter :
Integer.MAX_VALUE);
                     if (nRead == -1)
                           break;
               }

         } catch (IOException ex)
         {
               System.err.println ("Seek to " + nWhere + " failed, " + ex.getMessage());
   }

   m_nTotalRead += nActual;
   nRet = m_nTotalRead;
   }
   return nRet;
}
```

```
// Thread-safe
public long getContentLength() {
        long nRet;

        synchronized (LOCK) { nRet = m_nContentLength; }

        return nRet;
}
```

```
public ContentDescriptor getContentDescriptor() { return m_cd; }

// Our MyPullSourceStream class does not have any controls
public Object[] getControls() { return new Object[0]; }
public Object getControl(String parm1) { return null; }
}
```

```
// Contains the content-type returned by the InputStreamFactory
   protected ContentDescriptor                m_cd;
// Contains the PullSourceStream object
protected MyPullSourceStream[]                        m_vSources;
// Flag determining whether or not this DataSource is connected
protected boolean                                                m_bConnected;
// The InputStreamFactory passed into the constructor
protected final InputStreamFactory         m_isf;
```

O objeto InputStreamFactory não deve ser null, é usado primeiramente para reiniciar o input stream se uma busca é requisitada antes do buffer circular

```
   public EB_DataSource (InputStreamFactory isf)
   {
        m_isf = isf;
        }
```

Este método faz o armazenamento do content-type do objeto InputStreamFactory usado para iniciar este objeto e construir um objeto PullSourceStream.

```
public void connect() throws IOException
{
        if (!m_bConnected)
        {
                        String sMimeType;

                        sMimeType = m_isf.getContentTypeOfStream (null);
                        if (sMimeType == null)
                           sMimeType = "UnknownContent";
                        m_cd = new ContentDescriptor (

        ContentDescriptor.mimeTypeToPackageName(sMimeType)
                                );

                        m_vSources = new MyPullSourceStream[1];
                        m_vSources[0] = new MyPullSourceStream ();
                        m_bConnected = true;
                }
}
```

Retorna o content-type devolvido pelo InputStreamFactory usado para iniciar este objeto.

```
public String getContentType()
{
                if (!m_bConnected)
                throw new Error("Source is unconnected.");
                return m_cd.getContentType();
}
```

```
public void disconnect()
{
                if (m_bConnected)
                {
                   try {
                                m_vSources[0].close();
                   } catch (IOException ex) {
                        ex.printStackTrace();
                   }
                   m_bConnected = false;
                }
}
```

Retorna o objeto PullSourceStream pré-construído.

```
public PullSourceStream[] getStreams()
{
        if (!m_bConnected)
            throw new Error ("Unconnected source.");

        return m_vSources;
}
```

```
public void start() throws IOException {}
public void stop() throws IOException {}
public Time getDuration() { return Duration.DURATION_UNKNOWN; }

// This DataSource doesn't have any controls
public Object[] getControls() { return new Object[0]; }
public Object getControl(String controlName) { return null; }
```

```
public src.eBuffer.EB_DataSource createClone ()
{
        EB_DataSource                   ds = new EB_DataSource(m_isf);
        MediaLocator            ml = getLocator ();

        ds.setLocator(ml);

        if (m_bConnected) {
        try {
                        ds.connect();
            } catch (IOException e) {
                        return null;
            }
        }
        return ds;
}
```

```
public long tell ()
{
        long    nRet;
        synchronized (LOCK)
        {
                nRet = m_nTotalRead - posDistanceToTail ();
        }
        return nRet;
}
```

Aplicação

```
public class SimplePlayerGUI implements CommandListener, PlayerListener,
ItemStateListener, Utils.QueryListener {
...
}
```

```
    // Commands
    private Command backCommand = new Command("Back", Command.BACK, 1);
    private Command playCommand = new Command("Play", Command.ITEM, 1);
    private Command stopCommand = new Command("Stop", Command.ITEM, 1);
    private Command metaCommand = new Command("META data", Command.ITEM, 3);
    private Command volCommand = new Command("Volume", Command.ITEM, 2);
    private Command muteCommand = new Command("Mute", Command.ITEM, 1);
    private Command unmuteCommand = new Command("Unmute", Command.ITEM, 1);
    private Command loopCommand = new Command("Loopmode", Command.ITEM, 4);
    private Command stcCommand = new Command("Stop in 5 sec", Command.ITEM, 4);
    private Command selectCommand = new Command("Select", Command.ITEM, 1);
    private Command skipFwCommand = new Command("Skip Forward", Command.ITEM, 5);
    private Command skipBwCommand = new Command("Skip Backward", Command.ITEM, 5);
    private Command rewindCommand = new Command("Rewind", Command.ITEM, 5);
    private Command rateCommand = new Command("Rate", Command.ITEM, 5);
    private Command tempoCommand = new Command("Tempo", Command.ITEM, 5);
    private Command pitchCommand = new Command("Pitch", Command.ITEM, 5);
    private Command fullScreenCommand = new Command("Full Screen", Command.ITEM, 5);
    private Command normalScreenCommand = new Command("Full Screen", Command.ITEM, 5);
    private Command startRecordCommand = new Command("Start Recording", Command.ITEM, 5);
    private Command stopRecordCommand = new Command("Stop Recording", Command.ITEM, 5);
```

```
    private void assertPlayer() throws Throwable {
    String state="";
    try {
        debugOut("assertPlayer");
        // make sure we can go back, even if failed
        display.removeCommand(backCommand);
        display.addCommand(backCommand);
        if (player == null) {
            if (songInputStream == null) {
                state="Opening "+songLocator; setStatus(state+"...");
                player = Manager.createPlayer(songLocator);
      System.err.println("Manager.createPlayer(songLocator)");
            } else {
                state="Opening "+songName; setStatus(state+"...");
                player = Manager.createPlayer(songInputStream,
songContentType);
      System.err.println("Manager.createPlayer(songInputStream,
songContentType)");
            }
        }
        player.addPlayerListener(this);
        //System.out.println("player = " + player);
        state="Realizing"; setStatus(state+"...");
        player.realize();
        state="Prefetching"; setStatus(state+"...");
        player.prefetch();
        state="Prefetched"; setStatus(state);
        setPlayerCommands();
        // initiate redisplay
        parent.fullScreen(fullScreen);
        durationUpdated();
      } catch (Throwable t) {
        player=null;
        setStatus("");
        throw new MediaException(Utils.friendlyException(t)+" at "+state);
      }
  }
```

```
    public void startPlayer() {
        try {
       debugOut("startPlayer");

            if (targetState >= Player.STARTED)
                return;
            targetState = Player.STARTED;
            if (player == null || player.getState() < Player.PREFETCHED) {
                assertPlayer();
            }
            if (player == null || player.getState() >= Player.STARTED) {
                return;
            }
            updateLoop();
            // auto-rewind
            try {
                long duration = player.getDuration();
                if (duration != Player.TIME_UNKNOWN && player.getMediaTime()
>= duration) {
                    player.setMediaTime(0);
                }
            } catch (MediaException e) {}
            setStatus("Starting...");
            player.start();
            setStatus("Playing");
            setFeedback("");
            // tempo may have changed due to new position
            updateTempo(null);
        } catch (Throwable t) {
            error(t);
        }
    }
```

```
    public void closePlayer() {
      targetState = Player.CLOSED;
      if (player != null) {
         setStatus("Stopping...");
         try {
               player.stop();
         } catch (MediaException e) {}
         setStatus("Closing...");
         player.close();
         setStatus("Closed");
         player = null;
         initialize();
      }
  }
```

```
  public void pausePlayer() {
      if (player != null) {
         int state = player.getState();
         if (state >= Player.PREFETCHED)
               targetState = Player.PREFETCHED;
         else if (state >= Player.REALIZED)
               targetState = Player.REALIZED;
         else
               targetState = Player.UNREALIZED;
         setStatus("Stopping...");
         try {
               player.stop();
         } catch (MediaException e) {}
         clearKaraoke();
         setStatus("Stopped");
         setFeedback("");
      }
  }
```

```
    /** fast forward or fast backward */
public void skip(boolean backwards) {
        if (player != null) {
            long mTime = player.getMediaTime();
            // default 10 sek
            long jump = 10000000;
            long duration = player.getDuration();
            if (duration >= 0) {
                    // skip 5%
                    jump = duration / 20;
            }
            if (backwards) {
                    // Jump backward
                    setMediaTime(mTime-jump);
            } else {
                    // Jump forward
                    setMediaTime(mTime+jump);
            }
        }
}
```

```
public void changeRate(boolean slowdown) {
        int diff = 10000; // 10%
        if (slowdown) {
            diff = -diff;
        }
        RateControl rc = getRateControl();
        if (rc != null) {
            int ocr = rc.getRate();
            int ncr = ocr + diff;
            if (ncr >= rc.getMinRate() && ncr <= rc.getMaxRate()) {
                    int ecr = rc.setRate(ncr);
                    setFeedback("New rate: "+toFloatString(ecr, 3)+"%");
                    updateRate(rc);
                    // rate changes effective tempo.
                    updateTempo(null);
            }
        } else {
                    setFeedback("No RateControl!");
        }
}
```

...

```
public void setMediaTime(long time) {
        if (player == null) {
            return;
        }
        try {
            player.setMediaTime(time);
            setFeedback("Set MediaTime to "+timeDisplay(player.getMediaTime()));
            updateTime();
            clearKaraoke();
            updateTempo(null);
        } catch (MediaException me) {
            error(me);
        }
    }
```

```
    public void changeVolume(boolean decrease) {
        int diff = 10;
        if (decrease) {
            diff = -diff;
        }
        VolumeControl vc = getVolumeControl();
        if (vc != null) {
            int cv = vc.getLevel();
            cv += diff;
            vc.setLevel(cv);
            updateVolume(vc);
        } else {
                setFeedback("No VolumeControl!");
        }
    }
```

```
    public void toggleMute() {
        VolumeControl vc = getVolumeControl();
        if (vc != null) {
            vc.setMute(!vc.isMuted());
            updateVolume(vc);
        } else {
                setFeedback("No VolumeControl!");
        }
    }
```

```
    public void skipFrame(boolean back) {
        int diff = 1; // 1 frame
        if (back) {
            diff = -diff;
        }
        FramePositioningControl fpc = getFramePositioningControl();
        if (fpc != null) {
            int res = fpc.skip(diff);
            setFeedback("Skipped: "+res+" frames to
"+fpc.mapTimeToFrame(player.getMediaTime()));
        } else {
                setFeedback("No FramePositioningControl!");
        }
    }
```

```
    public void commandAction(Command c, Displayable s) {
        if (c == backCommand) {
            goBack();
        }
        else if (c == muteCommand || c == unmuteCommand) {
            mutePressed();
        } else if (c == volCommand) {
            volPressed();
        } else if (c == metaCommand) {
            metaPressed();
        } else if (c == loopCommand) {
            loopPressed();
        } else if (c == stcCommand) {
            stopAfterTime();
        } else if (c == playCommand || c == stopCommand) {
            togglePlayer();
        } else if (c == skipFwCommand) {
            skip(false);
```

```
        } else if (c == skipBwCommand) {
            skip(true);
        } else if (c == rewindCommand) {
            setMediaTime(0);
        } else if (c == rateCommand) {
            ratePressed();
        } else if (c == tempoCommand) {
            tempoPressed();
        } else if (c == pitchCommand) {
            pitchPressed();
        } else if (c == fullScreenCommand) {
            fullScreenPressed();
        } else if (c == startRecordCommand) {
            startRecording(null);
        } else if (c == stopRecordCommand) {
            stopRecording();
        } else if (s != display) {
            // e.g. when list item in MetaData display list is pressed
            goBack();
        }
    }
```

```
    public void playerUpdate(Player plyr, String evt, Object evtData) {
        System.err.println("SimplePlayerGUI.playerUpdate()");
        System.err.println(evt);
        System.err.println(songName);
            try {
        if ((evt == END_OF_MEDIA) && ((songName.startsWith("VodServer")) ||
(songName.startsWith("HTTPServer")))){
            //TODO implement
            //player.close();
            player = secPlayer;
            player.start();
```

```
        //fullScreenPressed();
        realizeSecondStep();
        return;
      }

            if ( (evt == END_OF_MEDIA) || (evt == STOPPED_AT_TIME) )
                  targetState = Player.PREFETCHED;

            // special case: end-of-media, but loop count>1 !
            if (evt == END_OF_MEDIA && plyr.getState() == Player.STARTED) {
                  setFeedback("Looping");
                  return;
            }
            if (evt == CLOSED || evt == END_OF_MEDIA  || evt == STARTED || evt ==
STOPPED_AT_TIME || evt == STOPPED)  {
                  if (timeDisplayTask!=null) {
                     timeDisplayTask.cancel();
                     timeDisplayTask=null;
                     updateTime();
                  }
            }
            if (evt == END_OF_MEDIA  || evt == STOPPED_AT_TIME || evt == STOPPED)  {
                  display.removeCommand(stopCommand);
                  display.addCommand(playCommand);
                  changeSongDisplayCounter = 0;
                  currSongDisplay = 0;
                  updateSongDisplay();
            }
            if (evt.equals("com.sun.midi.lyrics")) {
                  // META data
                  byte[] data=(byte[]) evtData;
                  if (data!=null && (evtData instanceof byte[]) && data.length>0) {
                     if (Utils.DEBUG) System.out.println("META event
0x"+Integer.toHexString(data[0] & 0xFF));
                     switch (data[0]) {
```

```
                    case 0x01:  // Text (commonly used for Karaoke, but not sent if Player is in
Karaoke mode)
                        // fall through
                    case 0x05:  // Lyrics (isn't this meant for Karaoke ??)
                        if (karaokeMode) {
                            break;
                        }
                        // fall through if not Karaoke
                    case 0x06: // marker
                        // fall through
                    case 0x07: // Cue point
                        String text = new String(data, 1, data.length-1);
                        setFeedback(text);
                        if (Utils.DEBUG) System.out.println("META event
0x"+Integer.toHexString(data[0] & 0xFF)+": "+text);
                        break;
                    case 0x51: // Tempo
                        updateTempo(null);
                        break;
                    case LYRICS_EVENT: // inofficial lyrics event: data 1-3 pos, 4-6 length
                        int kPos = (data[1] << 16) | (data[2] << 8) | (data[3] & 0xFF);
                        int kLen = (data[4] << 16) | (data[5] << 8) | (data[6] & 0xFF);
                        setupKaraokeLines(kPos, kLen);
                        updateKaraoke();
                        break;
                        //case 0x58: // Time Signature
                        //case 0x59: // Key Signature
                        //case 0x7F: // Proprietary
                    }
                }
            }

            if (evt == STARTED) {
                if (songDisplayNames.length > 1) {
                    changeSongDisplayCounter = SONG_DISPLAY_COUNTER * 2;
```

```
            }
            timeDisplayTask = new SPTimerTask();
            guiTimer.scheduleAtFixedRate(timeDisplayTask, 0, timerInterval);
            display.addCommand(stopCommand);
            display.removeCommand(playCommand);
        } else if (evt == DEVICE_UNAVAILABLE) {
            setFeedback("Audio device not available!");
        } else if (evt == BUFFERING_STARTED) {
            setFeedback("Buffering started");
        } else if (evt == BUFFERING_STOPPED) {
            setFeedback("Buffering stopped");
        } else if (evt == CLOSED) {
            setFeedback("Closed");
        } else if (evt == DURATION_UPDATED) {
            setFeedback("Duration updated");
            durationUpdated();
        } else if (evt == END_OF_MEDIA) {
            setStatus("End of media.");
            setFeedback("");
        } else if (evt == ERROR) {
            setFeedback("Error: "+((String) evtData));
        } else if (evt == RECORD_STARTED) {
            isRecording=true;
            display.addCommand(stopRecordCommand);
            display.removeCommand(startRecordCommand);
            setFeedback("Recording Started");
        } else if (evt == RECORD_STOPPED) {
            isRecording=false;
            display.addCommand(startRecordCommand);
            display.removeCommand(stopRecordCommand);
            setFeedback("Recording Stopped");
        } else if (evt == SIZE_CHANGED) {
            int[] newDim = (int[]) evtData;
            setFeedback("Resize to "+newDim[0]+"x"+newDim[1]);
        } else if (evt == STOPPED_AT_TIME) {
```

```
                setStatus("Stopped at time.");
                setFeedback("");
            } else if (evt == VOLUME_CHANGED) {
                VolumeControl vc = (VolumeControl) evtData;
                setFeedback("New volume: "+vc.getLevel());
                updateVolume(vc);
            }
        } catch (Throwable t) {
            if (Utils.DEBUG) System.out.println("Uncaught Exception in
SimplePlayerGUI.playerUpdate()");
            error(t);
        }
    }
```

```
public void itemStateChanged(Item item) {
        if (item!=null) {
           if (item == gauge) {
                switch (gaugeMode) {
                case GAUGE_VOLUME:
                   VolumeControl vc = getVolumeControl();
                   if (vc != null) {
                        vc.setLevel(gauge.getValue());
                        updateVolume(vc);
                   }
                   break;
                case GAUGE_RATE:
                   RateControl rc = getRateControl();
                   if (rc != null) {
                        rc.setRate((gauge.getValue()*1000)+rc.getMinRate());
                        updateRate(rc);
                   }
                   break;
                case GAUGE_TEMPO:
                   TempoControl tc = getTempoControl();
                   if (tc != null) {
                        tc.setTempo((gauge.getValue()+1)*1000);
                        updateTempo(tc);
                   }
                   break;
                case GAUGE_PITCH:
                   PitchControl pc = getPitchControl();
                   if (pc != null) {
                        pc.setPitch((gauge.getValue()*1000)+pc.getMinPitch());
                        updatePitch(pc);
                   }
                   break;
                } // switch
           }
        }
}
```

```
/**
 * The timer task that will be called every TIMER_INTERVAL
 * milliseconds
 */
private class SPTimerTask extends TimerTask {
        public void run() {
           updateTime();
           if (redisplayKaraokeCounter>0) {
                   redisplayKaraokeCounter--;
                   if (redisplayKaraokeCounter == 0) {
                      displayNextKaraokePhrase();
                   }
           }
           if (changeSongDisplayCounter > 0 && songDisplayNames.length>0) {
                   changeSongDisplayCounter--;
                   if (changeSongDisplayCounter == 0) {
                      currSongDisplay = (currSongDisplay + 1) % songDisplayNames.length;
                      updateSongDisplay();
                      changeSongDisplayCounter = SONG_DISPLAY_COUNTER;
                      if (currSongDisplay == 0) {
                             changeSongDisplayCounter *= 2;
                      }
                   }
           }
        }
}
```