

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMA DE INFORMAÇÃO**

**UM ESTUDO NA PLATAFORMA JAVA NA TRANSMISSÃO DE VÍDEO EM
TEMPO REAL**

**UM ESTUDO NA PLATAFORMA JAVA NA TRANSMISSÃO DE VÍDEO EM
TEMPO REAL**

Monografia apresentada ao Curso
de Graduação em Bacharelado em
Sistemas de Informação da
Faculdade Natalense para o
Desenvolvimento do Rio Grande do
Norte como requisito parcial para a
obtenção do grau de Bacharel em

Ricardo Alessandro de Medeiros Valentim

Orientador

Alaide Ferreira da Rocha Pires

**NATAL/RN
2004**

RICARDO ALEXSANDRO DE MEDEIROS VALENTIM

FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMA DE INFORMAÇÃO

**UM ESTUDO NA PLATAFORMA JAVA NA TRANSMISSÃO DE VÍDEO EM
TEMPO REAL**

**UM ESTUDO NA PLATAFORMA JAVA NA TRANSMISSÃO DE VÍDEO EM
TEMPO REAL**

Ricardo Alexandro de Medeiros Valentim

Monografia apresentada e aprovada em 11 de

Dezembro de 2004 pela banca examinadora composta pelos
seguintes membros.

**Monografia apresentada ao Curso
de Graduação em Bacharelado em
Sistema de Informação da
Faculdade Natalense para o
Desenvolvimento do Rio Grande do
Norte como requisito parcial para a
obtenção do grau em Bacharel em
Sistema de Informação.**

Orientador:

Aluizio Ferreira da Rocha Neto

**Natal/RN
2004**

UM ESTUDO NA PLATAFORMA JAVA NA TRANSMISSÃO DE VÍDEO EM
TEMPO REAL

Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Valentim, Ricardo Alexsandro de Medeiros

Um estudo na Plataforma Java na transmissão de vídeo em tempo real / Ricardo
Alexsandro de Medeiros Valentim. ____ Natal: FARN, 2004

p. 113

Orientador: Aluizio Ferreira da Rocha Neto (MSc)

Monografia (Bacharelado em sistemas de Informação) – Faculdade Natalense para o
Desenvolvimento do Rio Grande do Norte.

1. Linguagem de programação de computadores – Monografia
2. Java Media Frameworks – Monografia
3. Hipermídia – Monografia
- I. Título

RN/FARN/BC

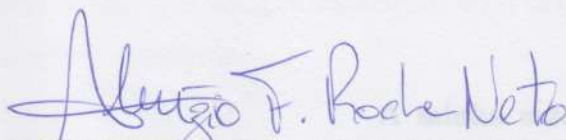
CDU 004.43

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMA DE INFORMAÇÃO**

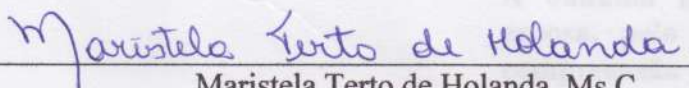
**UM ESTUDO NA PLATAFORMA JAVA NA TRANSMISSÃO DE VÍDEO EM
TEMPO REAL**

Ricardo Alessandro de Medeiros Valentim

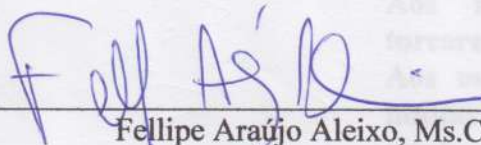
Monografia apresentada e aprovada em 11 de
DEZEMBRO de 2004 pela banca examinadora composta pelos
seguintes membros.



Aluizio Ferreira da Rocha Neto, Ms.C
Orientador

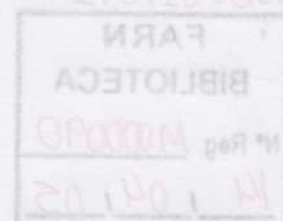


Maristela Terto de Holanda, Ms.C
Examinador



Felipe Araújo Aleixo, Ms.C
Examinador

Natal, 11 de DEZEMBRO de 2004



FARN

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
CURSO DE BACHARELADO EM SISTEMA DE INFORMAÇÃO

UM ESTUDO NA PLATAFORMA JAVA NA TRANSMISSÃO DE VÍDEO EM
TEMPO REAL

Ricardo Alexandre de Medeiros Valentinim

Monografia apresentada e aprovada em 11 de
de 2004 pela banca examinadora composta pelos
seguintes membros.

[Assinatura]

Aluizio Ferreira da Rocha Neto, M.Sc.
Orientador

[Assinatura]

Maristela Terto de Holanda, M.Sc.
Examinador

[Assinatura]

Felipe Araújo Aleixo, M.Sc.
Examinador

RB-21.371

FARN
BIBLIOTECA
Nº Reg. M000090
14 / 04 / 05

M

004.43

V251e

2004

Natal, 11 de Dezembro de 2004

AGRADECIMENTOS

Agradeço em primeiro lugar a minha família, pois em todos os momentos da minha vida sempre me apoiaram.

A minha mãe o agradecimento por reconhecer o meu esforço pessoal na formação de meu caráter humano e também pelo amor que a mesma tem por mim.

Aos meus irmãos, Adriano, César e Talmira o agradecimento, pois estes sempre torceram pelo meu sucesso pessoal e intelectual.

A minha esposa Janaína agradeço especialmente, pois reconheço nela uma excelente companheira que sempre tem confiado no meu potencial, prestando a minha vida com muito amor e compreensão principalmente nos instantes em que se fizeram necessários, sendo este um dos fortes motivos da realização deste trabalho.

Aos meus amigos, João Paulo, Vaincy Holanda e Valfrido Batista, agradeço pela motivação que vocês me deram durante toda a graduação e pelas várias noites de sono sendo nós encontrávamos estudando.

Aos mestres agradeço pelos ensinamentos que nós foram transmitidos, e ao meu orientador prof. Aluizio Ferreira da Rocha Neto por encontrar nestas não apenas um professor mais um grande amigo que sempre contribuiu com orientações e incentivos na execução dos meus trabalhos acadêmicos.

A minha mãe dedico este trabalho, como resultado de meu esforço e também do apoio e incentivo que dela tenho sempre recebido.

A Janaína minha linda e amada esposa, pelo estímulo, paciência e compreensão em suportar as minhas várias horas de ausência.

Aos meus irmãos por sempre torcerem pelo meu sucesso.

Aos meus amigos graduandos que muitas vezes foram estudar na minha residência me ajudando desta forma a consolidar os meus conhecimentos na graduação.

AGRADECIMENTOS

Agradeço em primeiro lugar a minha família, pois em todos os momentos da minha vida sempre me apoiaram.

A minha mãe o agradecimento por reconhecer o seu esforço pleno na formação de meu caráter humano e também pelo amor que a mesma tem por mim.

Aos meus irmãos, Adriano, César e Talmira o agradecimento, pois estes sempre torcem pelo meu sucesso pessoal e intelectual.

A minha esposa Janaína agradeço especialmente, pois reconheço nela uma excelente companheira que sempre tem confiado no meu potencial, preenchendo a minha vida com muito amor e compreensão principalmente nos instantes em que se fizeram necessários, sendo este um dos fortes motivos da realização deste trabalho.

Aos meus amigos, João Paulo, Volney Holanda e Valfredo Batista, agradeço pela motivação que vocês me deram durante toda a graduação e pelas várias noites de sono aonde nós encontrávamos estudando.

Aos mestres agradeço pelos ensinamentos que nós foram transmitidos, e ao meu orientador prof. Aluízio Ferreira da Rocha Neto por encontrar neste não apenas um professor mais um grande amigo que sempre contribuiu com orientações e incentivos na execução dos meus trabalhos acadêmicos.

RESUMO

Este trabalho realiza um estudo sobre a Plataforma Java na transmissão de vídeo em tempo real. Toda a pesquisa delineada neste trabalho tem como foco à produção de um modelo que represente uma aplicação hipermídia que seja capaz de reproduzir vídeo em tempo real. A fim de atender estes requisitos e de realizar as validações das teorias abordadas é criado um estudo de caso que efetivamente é a parte empírica deste trabalho. Nesta fase é amplamente discutida a implementação de dois modelos de aplicação. Cada um dos modelos codificados tem as suas funcionalidades específicas onde um tem o papel de reproduzir vídeo em tempo real e outro o de transmitir o vídeo capturado a partir de uma *webcam*.

Key words: Platform Java; JMF - Java Media Frameworks; Video in Real Time;

Palavras-chaves: Plataforma Java; JMF - Java Media Frameworks; Video em Tempo Real; Aplicação Hipemídia.

ABSTRACT

This work accomplishes a study on Java platform for video transmission in real time. The research has focus to the production of a model that represents a hypermedia application that is capable to reproduce video in real time. In order to assist these requirements and of accomplishing the validations of the approached theories it is created a case study that indeed is the empiric part of this work. In this phase it is discussed the implementation of two application models. Each one of the codified models has your specific functionalities, where one has the paper of reproducing video in real time the other one transmitting the video captured from a webcam.

Key words: Platform Java; JMF - Java Media Frameworks; Video in Real Time; hypermedia application.

SUMÁRIO

1	INTRODUÇÃO.....	13
1.1	OBJETIVOS.....	14
1.1.1	Objetivo Geral	14
1.1.2	Objetivos Específicos	14
1.2	JUSTIFICATIVA	15
1.3	METODOLOGIA.....	17
1.4	ORGANIZAÇÃO DO TRABALHO	18
2	REFERENCIAL TEÓRICO.....	19
2.1	SISTEMAS DE MONITORAMENTO REMOTO	19
2.2	SISTEMAS DE TEMPO REAL.....	21
2.3	MULTIMÍDIA.....	22
2.3.1	Multimídia associada ao Monitoramento Remoto	24
2.3.2	Multimídia: Sincronismo dos Quadros e Dependência Temporal	25
2.4	HIPERMÍDIA.....	28
2.4.1	Hipermídia e sua estrutura	29
2.4.2	Hipermídia: stream de vídeo.....	30
2.5	PROJETO ORIENTADO A OBJETOS.....	31
2.6	PLATAFORMA JAVA	34
2.6.1	Portabilidade	36
2.6.2	Segurança, distributividade e Applets.....	38
2.6.3	Assinatura Digital	41
2.7	JMF – JAVA MEDIA FRAMEWORK.....	42
2.7.1	Reproduzindo mídias com JMF	46
2.8	RTP – REAL-TIME TRANSPORT PROTOCOL	48
3	MODELAGENS, CODIFICAÇÃO E RESULTADOS	52
3.1	MODELAGEM DA APLICAÇÃO CLIENTE	52
3.1.1	Diagrama de Classes da Aplicação Cliente.....	55
3.1.2	Especificação das Classes de Domínio	57
3.1.3	Codificação do Modelo da Aplicação Cliente	62
3.1.3	Resultados obtidos com a codificação do modelo da Aplicação Cliente	71
3.2	MODELAGEM DO SERVIDOR DE <i>STREAM</i> DE VÍDEO	73
3.2.1	Diagrama de Caso de Uso.....	74
3.2.2	Diagrama de Seqüência.....	75

3.2.3 Diagrama de Classes	76
3.2.4 Especificação das Classes do Domínio	78
3.2.5 Codificação e Resultados	80
3.2.6 Resultados obtidos com a codificação do modelo da Aplicação Servidor	83
4 VERIFICAÇÃO DOS RESULTADOS	86
5 CONSIDERAÇÕES FINAIS	89
REFERÊNCIA BIBLIOGRÁFICA	90
ANEXOS	92
ANEXO 1 – CÓDIGO DA APLICAÇÃO CLIENTE	93
ANEXO 2 – CÓDIGO DA APLICAÇÃO SERVIDORA	100
ANEXO 3 – CÓDIGO DO COMPONENTE DA SUN	104
FIGURA 11 - DIAGRAMA DE CASO DE USO DA APLICAÇÃO CLIENTE	53
FIGURA 13 - FLUXO DE ATIVIDADES NA REPRODUÇÃO DE VÍDEO	54
FIGURA 14 - DOMÍNIO DA APLICAÇÃO CLIENTE	56
FIGURA 15 - ESTADOS DE UM PLAYER (FONTE: SUN, 2004)	60
FIGURA 16 - DIAGRAMA DE ESTADOS DE MANIPULADOR DE EVENTOS	61
FIGURA 17 - TELA DE ENTRADA	71
FIGURA 18 - INFORMANDO ALOCAÇÃO DA MÍDIA	72
FIGURA 19 - REPRODUZINDO VÍDEO E EXIBINDO UM DOS COMPONENTES DE CONTROLE	72
FIGURA 20 - DIAGRAMA DE CASO DE USO DA APLICAÇÃO SERVIDOR	74
FIGURA 21 - DIAGRAMA DE SEQUÊNCIA DA APLICAÇÃO SERVIDOR	75
FIGURA 22 - DOMÍNIO DA APLICAÇÃO SERVIDOR	77
FIGURA 23 - CONSOLA DO SERVIDOR DE STREAM DE VÍDEO	83
FIGURA 24 - APLICAÇÃO CLIENTE CAPTURANDO VÍDEO EM TEMPO REAL	85
FIGURA 25 - DADOS A RESPEITO DA TRANSMISSÃO	86

LISTA DE FIGURAS

FIGURA 1 - ASSISTINDO EM TEMPO REAL OS EVENTOS OCORRIDOS EM DIFERENTES AMBIENTES	22
FIGURA 2 - FUNÇÃO DA TAXA DE TRANSMISSÃO	26
FIGURA 3 - GRÁFICO ILUSTRANDO A AS TAXAS DE TRANSMISSÃO	27
FIGURA 4 - LEITOR E ESCRITOR DE <i>STREAM</i>	30
FIGURA 5 - ATIVIDADES DE DESENVOLVIMENTO (LARMAN 2002, p. 30)	32
FIGURA 6 - ESQUEMA DE PORTABILIDADE DA PLATAFORMA JAVA	38
FIGURA 7 - TAG HTML PARA ADICIONAR UM APPLET A UMA A UMA PÁGINA DA INTERNET	40
FIGURA 8 - MODELO DE REPRODUÇÃO DE MÍDIAS BASEADAS EM TEMPO.....	45
FIGURA 9 - POSICIONAMENTO DA JMF NA CONSTRUÇÃO DE APLICAÇÕES	46
FIGURA 10 - ARQUITETURA DO PROTOCOLO RTP.....	49
FIGURA 11 - CABEÇALHO DO PROTOCOLO RTP.....	50
FIGURA 12 - DIAGRAMA DE CASO DE USO DA APLICAÇÃO CLIENTE	53
FIGURA 13 FLUXO DE ATIVIDADES NA REPRODUÇÃO DE VÍDEO.....	54
FIGURA 14 - DOMÍNIO DA APLICAÇÃO CLIENTE	56
FIGURA 15 - ESTADOS DE UM <i>PLAYER</i> (FONTE: SUN, 2004)	60
FIGURA 16 - DIAGRAMA DE ESTADOS DE MANIPULADOREVENTOS	61
FIGURA 17 - TELA DE ENTRADA	71
FIGURA 18 - INFORMANDO A LOCALIZAÇÃO DA MÍDIA	72
FIGURA 19 - REPRODUZINDO VÍDEO E EXIBINDO UM DOS COMPONENTES DE CONTROLE	72
FIGURA 20 - DIAGRAMA DE CASO DE USO DA APLICAÇÃO SERVIDOR	74
FIGURA 21 - DIAGRAMA DE SEQÜÊNCIA DA APLICAÇÃO SERVIDOR.....	75
FIGURA 22 - DOMÍNIO DA APLICAÇÃO SERVIDOR.....	77
FIGURA 23 - CONSOLE DO SERVIDOR DE <i>STREAM</i> DE VÍDEO	83
FIGURA 24 - APLICAÇÃO CLIENTE CAPTURANDO VÍDEO EM TEMPO REAL	85
FIGURA 25 - DADOS A RESPEITO DA TRANSMISSÃO	86

LISTA DE TABELAS

TABELA 1 - TIPOS DE MÍDIA SUPOSTAS PELA JMF	43
TABELA 2 - TIPOS DE DISPOSITIVOS SUPORTADO PELA JMF	44
TABELA 3 - ESPECIFICAÇÃO DA CLASSE CLIENTEVIEW	58
TABELA 4 - ESPECIFICAÇÃO DA CLASSE MANIPULADOREVENTO	61
TABELA 5 - ESPECIFICAÇÃO DA CLASSE VIDEOTRANSMIT	78
TABELA 6 - ESPECIFICAÇÃO DA CLASSE GERAVIDEOWebCAM	79
QUADRO 7 - APLICAÇÃO CLIENTE: CRIAÇÃO DO PLAYER	66
QUADRO 8 - APLICAÇÃO CLIENTE: OBTENDO OS COMPONENTES VISUAIS DO PLAYER	67
QUADRO 9 - APLICAÇÃO CLIENTE: LIGANDO O PLAYER	68
QUADRO 10 - APLICAÇÃO CLIENTE: SUPER CLASS	69
QUADRO 11 - APLICAÇÃO SERVIDORA: CÓDIGO PARA TRANSMISSÃO	79
QUADRO 12 - APLICAÇÃO SERVIDORA: MÉTODO CONSTRUCTOR	81
QUADRO 13 - APLICAÇÃO SERVIDORA: MÉTODO OUVERIR	82
QUADRO 14 - CÓDIGO FONTE: CLIENTEVIEW.JAVA	93
QUADRO 15 - CÓDIGO FONTE: GERAVIDEOWebCAM.JAVA	100
QUADRO 16 - CÓDIGO FONTE: VIDEOTRANSMIT.JAVA	104

LISTA DE QUADROS

QUADRO 1 – APLICATIVO PARA REPRODUÇÃO DE VÍDEO LOCALIZADO POR UMA URL	47
QUADRO 2 - APLICAÇÃO CLIENTE: IMPORTANDO CLASSES.....	62
QUADRO 3 - APLICAÇÃO CLIENTE: DECLARAÇÃO DA CLASSE E SEUS ATRIBUTOS	63
QUADRO 4 - APLICAÇÃO CLIENTE: MÉTODO DE INICIALIZAÇÃO	63
QUADRO 5 - APLICAÇÃO CLIENTE: MÉTODO DE RETORNO DA MÍDIA	65
QUADRO 6 - APLICAÇÃO CLIENTE: MÉTODO PARA EXIBIÇÃO DE ERRO	65
QUADRO 7 - APLICAÇÃO CLIENTE: CRIAÇÃO DO PLAYER.....	66
QUADRO 8 - APLICAÇÃO CLIENTE: OBTENDO OS COMPONENTES VISUAIS DO PLAYER	67
QUADRO 9 - APLICAÇÃO CLIENTE: LIBERANDO O PLAYER.....	68
QUADRO 10 - APLICAÇÃO CLIENTE: <i>INNER CLASS</i>	69
QUADRO 11 - APLICAÇÃO SERVIDORA: CÓDIGO PARA TRANSMISSÃO.....	79
QUADRO 12 - APLICAÇÃO SERVIDORA: MÉTODO CONSTRUTOR.....	81
QUADRO 13 - APLICAÇÃO SERVIDORA: MÉTODO OUVIDOR	82
QUADRO 14 - CÓDIGO FONTE: CLIENTEVIEW.JAVA	93
QUADRO 15 - CÓDIGO FONTE: GERAVIDEOWebCAM.JAVA.....	100
QUADRO 16 - CÓDIGO FONTE: VIDEOTRANSMIT.JAVA.....	104

LISTA DE ACRÔNIMOS

API – *Application Programming Interface* – Interface para Programação de Aplicações

HTML – *Hyper Text Markup Language* – Linguagem de Marcação de Hiper Texto

IP – *Internet Protocol* – Protocolo de Internet

JMF – *Java Media Frameworks*

JMP – *Java Media Player*

JVM – *Java Virtual Machine* – Máquina Virtual Java

RGB – Red, Green e Blue – Vermelho, Verde e Azul

RTCP – *Real-Time Control Protocol* – Protocolo de Controle em Tempo Real

RTP – *Real-Time Transport Protocol* – Protocolo de Transmissão em Tempo Real

PB – Preto e Branco

S.O. – Sistema Operacional

UDP – *User Datagram Protocol*

UML – *Unified Modeling Language* – Linguagem de Modelagem Unificada

URL – *Uniform Resource Locator* – Endereço Único na Internet

1. INTRODUÇÃO

Atualmente, a Internet provê um excelente meio de interação entre pessoas, empresas, governo e tudo que possa estar inserido neste contexto de mídia interativa. Esta interação é uma ótima ferramenta de comunicação, pois facilita o acesso, uma vez que a sua disponibilidade é integral e sem limite geográfico, ou seja, pode ser acessada de qualquer lugar e a qualquer tempo, bastando para tanto, estar conectado a esta grande rede mundial de computadores.

Com os avanços tecnológicos, as aplicações multimídia¹ também passaram a sofrer deste meio de comunicação. Com isso, já é possível disponibilizar som, animações gráficas, vídeos, vídeos em tempo real e outros recursos que envolvam a comunicação digital na Web.

Permeando o contexto de aplicações multimídia na Internet, o presente trabalho desenvolve um estudo sobre os recursos disponibilizados na Plataforma Java sobre transmissão de vídeo em tempo real. Neste sentido, este trabalho monográfico especifica estes recursos, bem como, realiza uma verificação desta base com um modelo desenvolvido, no qual subsidiou a construção de uma aplicação hipermídia². Para tanto, a pesquisa tem como enfoque principal a transmissão de fluxos (*streams*) de vídeo usando como meio de comunicação a Internet.

O presente trabalho delinea toda a problemática de pesquisa – transmissão de vídeo em tempo real – não apenas em torno dos recursos que a Internet disponibiliza para tais fins, mas também, como estes recursos são utilizados, tais como: retardo na transmissão de imagens, transmissão de formatos de mídia, entre outros. Sendo assim, é comum encontrar publicações que discutem a transmissão de vídeo em tempo real, tais como:

**“Antes mundo era pequeno
Porque Terra era grande
Hoje mundo é muito grande
Porque Terra é pequena
Do tamanho da antena
parabolicamará”. (Gilberto
Gil, 1996)**

Desenvolvendo a problemática em torno do objeto de estudo, este trabalho, seguindo uma abordagem orientada a objetos, o qual é utilizado na construção de uma aplicação a fim de subsidiar os estudos desenvolvidos nesta pesquisa.

¹ Aplicações que fazem uso de sons, imagens e vídeos.

² Os fluxos que utilizam recursos multimídia concatenados às páginas da Internet.

1 INTRODUÇÃO

Atualmente, a Internet provê um excelente meio de interação entre pessoas, empresas, governo e tudo que possa estar inserido neste contexto de mídia interativa. Esta interação é uma ótima ferramenta de comunicação, pois facilita o acesso, uma vez que a sua disponibilidade é integral e sem limite geográfico, ou seja, pode ser acessada de qualquer lugar e a qualquer tempo, bastando para tanto, estar conectado a esta grande rede mundial de computadores.

Com os avanços tecnológicos, as aplicações multimídia¹ também passaram a usufruir deste meio de comunicação. Com isso, já é possível disponibilizar som, animações gráficas, vídeos, vídeos em tempo real e outros recursos que envolvem a comunicação digital na Web.

Permeando o contexto de aplicações multimídia na Internet, o presente trabalho desenvolve um estudo sobre os recursos disponibilizados na Plataforma Java sobre transmissão de vídeo em tempo real. Neste sentido, este trabalho monográfico especifica estes recursos, bem como, realiza uma verificação com base em um modelo desenvolvido, no qual subsidiou a construção de uma aplicação hipermídia². Para tanto, a pesquisa tem como enfoque principal à transmissão de fluxos (*streams*) de vídeo usando como meio de comunicação a Internet.

O presente trabalho delinea toda a problemática de pesquisa – transmissão de vídeo em tempo real – não apenas em torno dos recursos que a linguagem de programação Java disponibiliza para tais fins, mas também, como estes recursos contribuem na solução de problemas, tais como: retardo na transmissão de imagens, redução das taxas de transmissão, e conversão de formatos de mídia, entre outros. Sendo este, um fator relevante, uma vez que não é comum encontrar publicações que discutem estes problemas de forma centralizada.

Permeando a problemática em torno do objeto de estudo, é desenvolvido um modelo de sistema orientado a objetos, o qual é utilizado na construção de uma aplicação a fim de implementar os estudos desenvolvidos nesta pesquisa.

¹ As aplicações que fazem uso de som, imagens e vídeos.

² São sistemas que utilizam recursos multimídia concatenados às páginas da Internet

A aplicação construída através do modelo é um sistema que pode ser utilizado para o monitoramento remoto de ambientes. Deste modo, servindo de estudo de caso, pois tornou possível observar os resultados obtidos com a tecnologia pesquisada.

A construção da aplicação na Plataforma Java serviu de base para a consolidação dos estudos realizados em torno da temática abordada (transmissão de vídeo em tempo real utilizando como meio de transmissão a Internet).

Igualmente, através desta aplicação, são analisados fatores como eficácia do ambiente de programação na construção de aplicações hipermídia voltadas para o envio de *streams* de vídeo em tempo real. Com isso, sendo possível realizar um estudo de viabilidade em se utilizar a Plataforma Java na construção de aplicações dotadas deste tipo de características. Neste sentido, agregando uma excelente contribuição ao trabalho, pois poderá servir de base para outras implementações semelhantes.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo deste trabalho é realizar um estudo da Plataforma Java na transmissão de vídeo em tempo real, tendo em vista a criação de um modelo de sistema hipermídia orientado a objetos, no qual será desenvolvida uma aplicação a fim de se observar os resultados.

1.1.2 Objetivos Específicos

Os objetivos específicos são estudar as temáticas que estão inseridas no escopo deste projeto de pesquisa, a fim de delinear o referencial teórico de forma consistente, para tanto serão abordadas as seguintes temáticas:

- Multimídia;
- Hipermídia;
- Protocolo RTP (*Real-Time Transport Protocol* – Protocolo de Transmissão em Tempo Real);
- Sistemas de Tempo Real;

- Sistemas de Monitoramento Remoto;
- Modelagem e Sistemas Orientados Objetos

Também serão objetivos específicos desta monografia:

- Analisar as possibilidades da transmissão de vídeo em tempo real, verificando as formas pela qual a Plataforma Java dá suporte a construção efetiva deste tipo de aplicação;
- Construir um modelo conceitual que permita um fácil entendimento do domínio do problema;
- Viabilizar a construção de um sistema, aonde será possível avaliar a fundamentação teórica do trabalho, bem como, prover a coleta de resultados. Sendo assim evidenciando a concatenação do ambiente Java as técnicas de multimídia e hipermídia.

1.2 JUSTIFICATIVA

Em anos recentes, o segmento de multimídia digital do setor de computadores experimentou um crescimento enorme, como demonstra a grande quantidade de conteúdo multimídia disponível na Internet.

Com esta crescente demanda de sites que utilizam multimídia, abre-se uma infinidade de possibilidades para desenvolvimento de aplicações na web. Desta forma, também cresce a necessidade de dominar tecnologias que possibilitem manipular e desenvolver soluções que se enquadrem neste contexto.

Observando a importância que os sistemas multimídia têm como forma de comunicação para com o usuário final, este trabalho propõe um estudo que abrange significativamente a transmissão de vídeo acoplado as simples estruturas HTML.

Este estudo terá uma contribuição relevante, pois o seu principal objeto de pesquisa esta fortemente ligada a uma tecnologia que disponibiliza recursos para o desenvolvimento pleno de aplicações hipermídia.

A fundamentação teórica e empírica subsidiada por este trabalho permitirá uma contribuição diante da temática abordada, pois irá proporcionar um nível de detalhamento e descrição que poderá servir como instrumento para outras pesquisas.

Questões como protocolos de transmissão de vídeo em tempo real, captura de imagens através de dispositivos de videoconferência, armazenamento de imagens, serão amplamente abordados e detalhados neste trabalho.

A principal abordagem da pesquisa será com relação ao uso da API JMF (*Java Media Frameworks*)³ que dá suporte amplo ao protocolo RTP (*Real-Time Transport Protocol* – Protocolo de Transmissão de em Tempo Real). Por isso, o trabalho agrega uma importante contribuição dentro da sua área de pesquisa, pois irá tratar a forma pela qual a linguagem Java se relaciona com as aplicações multimídia e como ocorre o processo de transmissão de vídeo.

Os sites da web foram transformados em páginas HTML (*Hyper Text Markup Language* – Linguagem de Marcação de Hiper Texto) que utilizam textos combinados aos mais diversos recursos de multimídia. Os avanços tecnológicos de hardware e de software permitem que os desenvolvedores de aplicações multimídia integrem recursos mais sofisticados a sites da Internet. Desta forma, aumentando a demanda por sistemas Web que utilizam recursos de vídeo, som e imagem.

Observando esta crescente demanda por aplicações que possam utilizar vídeo na Internet é de fundamental importância que existam pesquisas nas quais possam favorecer um endimento mais amplo a esta temática.

Norteados por este contexto, a pesquisa consolidará várias abordagens, nas quais agrega informações que não são triviais em outros trabalhos com um enfoque similar, pois é raro encontrar textos que tratem de forma centralizada as temáticas que serão estudadas nesta pesquisa. Portanto, sendo este um dos fatores que justificam a sua viabilidade, haja vista a permeabilidade em torno dos assuntos fundamentais da Plataforma Java na construção de aplicações hipermídia que tenha como objetivo transmitir vídeo em tempo real.

³ Esta API é um pacote da linguagem Java utilizada em aplicações multimídia.

1.3 METODOLOGIA

Este trabalho divide-se basicamente em dois pontos fundamentais: a construção do referencial teórico metodológico da pesquisa, que para tanto, utiliza como base o seu objeto de estudo; o outro ponto é a criação de uma situação prática que se fundamenta num modelo conceitual, no qual, fornece subsídios para a construção de um sistema. Sendo este, o foco de observações dos resultados acerca do conteúdo teórico empregado neste trabalho.

Diante deste contexto, o referido trabalho utiliza um tipo de pesquisa exploratória, com vistas a provê um levantamento bibliográfico consistente. Igualmente, a pesquisa realizada através deste tipo metodológico promove observações de resultados provenientes da problemática abordada, ora devidamente pesquisada.

Para tanto, são aplicados as seguintes técnicas e procedimentos de pesquisa: monográfico, estudo de caso e funcionalista. Sendo estas, substanciais no desenvolvimento da pesquisa, uma vez que são adequadas ao contexto.

A metodologia de pesquisa monográfica é aplicada na construção do referencial teórico deste trabalho, pois esta trata de especificar os conceitos que envolvem a temática do objeto de estudo da pesquisa. Desta forma, a revisão bibliográfica se norteia em livros, artigos e sites da Internet que abordam direta ou indiretamente o objeto pesquisado. Portanto, na efetivação desta fase, a técnica metodológica aplicada é a observação indireta, pois permiti uma investigação bibliográfica do conteúdo preterido.

A metodologia de estudo de caso é aplicada na construção de um modelo conceitual da problemática abordada. Esta modelagem é balizada na teoria da orientação a objetos. Na qual, prover um embasamento efetivo ao estudo de caso que é desenvolvido por este trabalho monográfico. Para tanto, é utilizado no desenvolvimento deste arquétipo, as técnicas da UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada) a fim de propôr um modelo conceitual orientado a objetos do estudo de caso.

A metodologia funcionalista é aplicada na construção do sistema, sendo este norteado pelo modelo proposto na UML. Nesta etapa, ocorrem os testes e as validações efetivas de toda a pesquisa. Sendo assim, o desenvolvimento do sistema, envolve toda a parte empírica do trabalho, ou seja, nesta fase será efetivamente desenvolvido um aplicativo que dá ênfase as teorias abordadas pela pesquisa. Igualmente, através desta metodologia é

possível observar as técnicas de multimídia inter-relacionadas à linguagem Java, pois se fundamentará nos fatores empíricos deste projeto de pesquisa.

Portanto, na fase de estudo de caso e da implementação do sistema, é aplicada à técnica metodológica de observação direta extensiva, pois esta possibilita o desenvolvimento do modelo baseado na UML, bem como simular e testar toda a aplicação verificando desta forma, os resultados obtidos.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho encontra-se dividido em 5 capítulos, os quais são antecipadamente descritos a seguir:

- Capítulo 1: este capítulo faz uma contextualização sobre o objeto de estudo abordado nesta monografia. Para tanto, fazendo uma introdução na temática que será abordada. Neste sentido, abordando os objetivos do trabalho, a justificativa pelo qual o mesmo foi envolvido e a metodologia utilizada em seu desenvolvimento;
- Capítulo 2: neste capítulo é desenvolvido todo o referencial teórico em torno da temática abordada;
- Capítulo 3: neste capítulo é realizado o desenvolvimento empírico do trabalho, abordando modelos e citando os resultados alcançados com as suas implementações;
- Capítulo 4: este capítulo aborda os resultados não discutidos no capítulo 3, neste sentido, discutindo os resultados não funcionais obtidos com a aplicação;
- Capítulo 5: são as considerações finais sobre o trabalho pesquisado nesta monografia;

2 REFERENCIAL TEÓRICO

O referencial teórico deste trabalho visa definir e/ou classificar as principais tecnologias abordadas na monografia. Desta forma, permitindo uma coesão consistente entre os pontos de discussão e a implementação efetiva deste trabalho de pesquisa, na qual consistem a problemática do objeto de estudo. Para tanto, serão abordadas neste referencial teórico as seguintes tecnologias:

- Sistemas de Monitoramento Remoto;
- Sistemas de Tempo Real;
- Multimídia e Hiperlinks;
- Orientação a Objetos e Projeto Orientado a Objetos;
- Plataforma Java;
- Java Media Framework;
- Protocolo RTP.

2.1 SISTEMAS DE MONITORAMENTO REMOTO

[...] Os sistemas de monitoramento remoto além de poderem compor com outros subsistemas (segurança, por exemplo), são capazes de um raio de ação muito amplo e tem varias formas de atuação dependendo somente de uma configuração (LAR, 2004, p. 1).

Sistemas que permitem a um ator observar determinados comportamentos ou eventos remotamente são denominados de sistemas de monitoramento remoto. Estes atores do sistema podem ser: pessoas, dispositivos ou agentes (software) que de alguma forma utilizam a arquitetura do sistema para monitorar determinados eventos ou comportamentos de forma remota. Nesta sessão, serão citados alguns exemplos de utilização de sistemas de monitoramento remoto, a fim de prover uma melhor visão a respeito do assunto.

Os sistemas de monitoramento remoto têm varias aplicações, tais como: monitoramento de tráfego de rede, monitoramento de temperatura de caldeiras, monitoramento de tráfego aéreo, monitoramento de ambientes, telemetria⁴, entre outras. De fato, existem vários tipos de monitoramento remoto, os mais populares são os baseados em aplicações multimídia.

A violência infantil, tanto em casa quanto na escola, preocupa a sociedade. Como não podem abrir mão do trabalho para acompanhar de perto o crescimento dos filhos, pais buscam tranquilidade e segurança em tecnologias que viabilizam a monitoração à distância (PÁTIO VIRTUAL, 2002).

Atualmente, uma aplicação que vem se tornando comum são sistemas que estão sendo usados para que os pais possam observar os seus filhos no colégio enquanto estes se encontram em casa ou no trabalho. Desta forma, tornando mais cômodo acompanhar os filhos na escola.

De acordo com padre Abelardo Benito Rico [...] diretor da escola, com 900 alunos matriculados, o sistema permite que os pais consultem a Internet para saber se os filhos realmente entraram nas salas de aula. "Os jovens passam boa parte do dia aqui, cabe a nós garantir que nada de mal aconteça a eles" (PÁTIO VIRTUAL, 2004).

Um outro tipo de sistema de monitoramento é o de vigilância remota, estes possibilitam a vigilância de ambientes através de imagens ou vídeos que podem ou não ser transmitidos em tempo real. Estes sistemas estão sendo utilizados na área de segurança de lojas, condomínios, casas, entre outros patrimônios que possam ser monitorados através da Internet.

O sistema Geovision, baseado na transmissão de dados pelo computador, permite acessar todas as câmeras de qualquer lugar do mundo através do próprio programa ou simplesmente do navegador da Internet, visualizando em tempo real até 16 câmeras com qualidade digital (PROJETEC, 2004).

O estudo de caso deste trabalho se fundamenta na transmissão de video em tempo real para criar um sistema de vigilância remota, mas a pesquisa desenvolvida pode ser aplicada nas mais divedas áreas que utilizam a transmissão de video ao vivo, uma vez que são enumeras as suas aplicações.

⁴ Telemetria: são sistemas que utilizam transmissão unidirecional ou bidirecional em rádio freqüência para receber ou transmitir dados a um determinado dispositivo ou software. Um exemplo comum são os sistemas utilizados nos carros de fórmula 1.

2.2 SISTEMAS DE TEMPO REAL

Os sistemas de tempo real são projetados em uma base temporal. Desta forma é bastante complexo desenvolver sistemas com esta característica. Esta complexidade existe em razão do fator temporal, que para garantir a qualidade do sistema deve ser sempre levado em consideração.

[...] Todas essas interações computadorizadas – sejam elas úteis ou instrutivas – são exemplos de computação em tempo real. O computador está controlando algo que interage com a realidade numa base oportuna quanto ao tempo. De fato, o *timing* é a essência da interação... Um sistema de tempo real indiferente pode ser pior do que nenhum sistema em absoluto (GLASS apud PRESSMAN, 1995, p. 634).

Figura 1 - Assistência em tempo real às crises ocorridas em diferentes regiões

Neste trabalho, um dos objetos de estudo mais relevante é a transmissão de vídeo em tempo real. Para tanto, questões como o tempo em que os quadros do vídeo (cenas) são transmitidos para o usuário do sistema é uma preocupação pertinente, pois a aplicação deve garantir que as imagens exibidas de fato estão ocorrendo naquele instante.

Segundo Pressman (1995, p. 633):

O software de tempo real é altamente acoplado ao mundo externo. Ou seja, o software de tempo real deve responder ao domínio do problema (o mundo real) numa escala tempo dita por este. Uma vez que o software deve operar sob rigorosas restrições [...].

Este acoplamento ao mundo externo no qual Pressman faz alusão na citação acima é mapeado neste trabalho através da modelagem do sistema. Esta dependência do aplicativo ao mundo real ocorre em função da conexão com dispositivo de captura de vídeo (*webcam*), uma vez que este é fundamental para que exista a transmissão de vídeo em tempo real. A Figura 1 ilustra a citação de Pressman.

2.3 MULTIMÍDIA

As formas de comunicação vêm ao longo da história da humanidade evoluindo e cada vez mais se multiplicando. Antes tínhamos desenhos nas cavernas e escrituras rústicas em pedras. Com a evolução intelectual do homem, estes meios de comunicação também

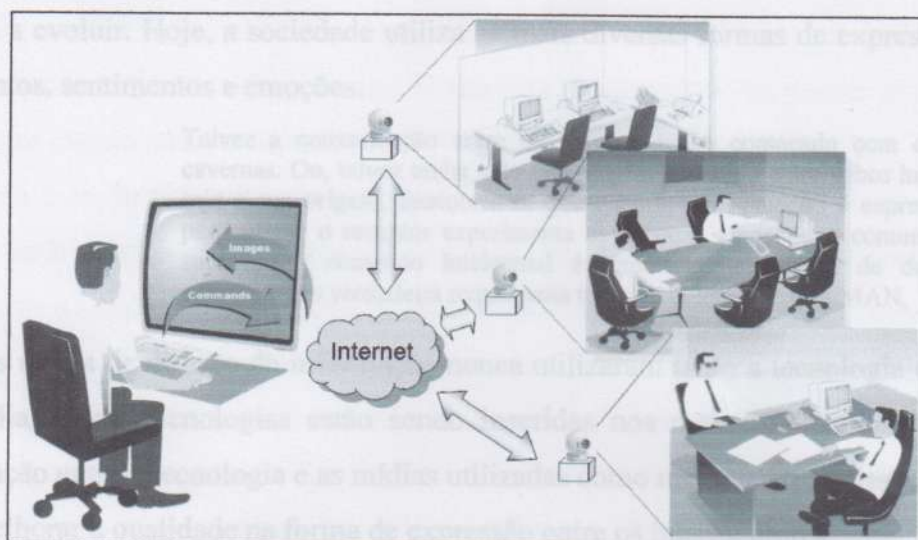


Figura 1 - Assistindo em tempo real os eventos ocorridos em diferentes ambientes

Segundo Pressman (1995, p. 635), “Os sistemas de tempo real geram certa ação e resposta a eventos externos. Para cumprir essa função, eles realizam controle e aquisição de dados em alta velocidade sob severas restrições de tempo e confiabilidade”.

Para que o monitoramento remoto possa ocorrer dentro deste contexto às imagens capturadas pelo dispositivo devem ser transmitidas de forma sequencial e muito rapidamente. A velocidade em que as imagens são transmitidas irá depender basicamente da largura de banda utilizada para transmissão e recepção destas imagens. Desta forma, o sistema fica restrito a infra-estrutura de rede para garantir uma transmissão de boa qualidade e conseqüentemente o vídeo.

Esta restrição em relação à infra-estrutura de rede deve ser levada em consideração, pois todo o desempenho do sistema estará relacionado diretamente às condições oferecidas para a transmissão de vídeo. Este trabalho não pretende estudar ou desenvolver um sistema de tempo real em si, mas tem uma forte associação, considerando, que o modelo proposto tem um alto acoplamento ao mundo real e também por estar bastante restrito ao fator temporal.

2.3 MULTIMÍDIA

As formas de comunicação vêm ao longo da história da humanidade evoluindo e cada vez mais se multiplicando. Antes tínhamos desenhos nas cavernas e escrituras rústicas em pedras. Com a evolução intelectual do homem, estes meios de comunicação também

passaram a evoluir. Hoje, a sociedade utiliza as mais diversas formas de expressar os seus pensamentos, sentimentos e emoções.

Talvez a comunicação entre os homens tenha começado com desenhos nas cavernas. Ou, talvez tenha sido um simples grunhido entre tribos humanas. Qual seja a sua origem, descobriu-se que quando uma emoção é expressa de modo particular e o receptor experimenta as mesmas emoções, a comunicação é um sucesso. O conteúdo intelectual é importante em troca de dados, mas a comunicação verdadeira requer uma troca emocional (WOLFMAN, 1994, p. 1).

Os meios de difusão da informação nunca utilizaram tanto a tecnologia como hoje. A cada dia, novas tecnologias estão sendo inseridas nos meios de comunicação. Esta concatenação entre a tecnologia e as mídias utilizadas como meio de transmissão ocorrem a fim de melhorar a qualidade na forma de expressão entre os interlocutores.

Um dos paradigmas das comunicações está nos computadores. Hoje, estes dispositivos são excelentes ferramentas de comunicação, pois diferente dos jornais, rádios e televisão eles promovem interação. Assim sendo, um usuário deste veículo de comunicação pode demonstrar e manifestar reações a uma determinada informação, ou seja, esta mídia permite interação entre o usuário e o conteúdo.

Atualmente, a multimídia é um excelente recurso na transmissão de informação, pois permite uma comunicação de forma mais ampla. Esta amplitude alcançada pelos aplicativos multimídia existe pelo fato da combinação possível de recursos utilizados na composição de uma mensagem. Uma aplicação multimídia pode ser composta de vários recursos, tais como: texto, imagens estáticas, áudio e vídeo.

Segundo Paulo Filho (2000, p.3):

Por multimídia entenderemos todos os programas e sistemas em que a comunicação entre homem e computador se dá através de múltiplos meios de representação da informação, como som e imagem animada, além da imagem estática já usada nos aplicativos gráficos.

Segundo Wolfman (1994, p. 9):

De certo modo, a multimídia é a culminação de todas as formas preliminares de comunicação. É a única tecnologia que engloba áudio, vídeo, imagens estáticas, texto e interação. É através do uso efetivo dessas tecnologias que o mais poderoso impacto pode ser conseguido em uma audiência. Isso é possível porque essas tecnologias, quando combinadas, têm a capacidade de influenciar mais as emoções que qualquer outro subconjunto ou combinação.

“Se acreditarmos que a informação é a resposta emocional à análise de dados, então, até hoje, a multimídia é a mais poderosa ferramenta de apresentação, porque ela é capaz de provocar um grande número de emoções” (WOLFMAN, 1994, p. 9).

Esta emoção provocada pelos aplicativos multimídia é de suma relevância quando se deseja ter o máximo de realidade na apresentação, ou seja, a utilização destes recursos pode garantir a aplicação uma melhor qualidade do conteúdo que será transmitido.

Os recursos multimídia têm um enorme poder em nível de composição da informação, pois permite uma elaboração bem mais sofisticada do conteúdo e quando isso ocorre, torna a exibição da mensagem muito mais refinada para o usuário final.

Nas aplicações multimídia a possibilidade deste binômio (sofisticação e refinamento) existe justamente por:

- Diversidade de combinação recursos (vídeos, som ou outros tipos de arquivos multimídia);
- Possibilidade de poder manipular todos os recursos envolvidos na aplicação;
- Provê interações;

Estes pontos tornam a aplicação multimídia, um recurso ímpar na construção de sistemas que desejam estabelecer uma forma de comunicação eficiente com o usuário final.

2.3.1 Multimídia associada ao Monitoramento Remoto

A transmissão de vídeo em tempo real utiliza bastante os recursos de multimídia no que diz respeito à exibição das imagens. As técnicas de multimídia devem ser aplicadas a fim de provê aos quadros de imagens transmitidas o máximo de clareza com relação às cenas apresentadas. Sendo assim, as técnicas aplicadas devem levar em consideração o sincronismo das imagens, ou seja, a sequência lógica em que as imagens serão apresentadas.

O sincronismo lógico das imagens em um sistema de monitoramento remoto (transmissão de vídeo em tempo real), dá ao usuário a sensação de estar presente no ambiente ao qual se estar observando, pois permitirá ao observador ter uma percepção real dos movimentos ou das ações ocorridas no ambiente. Para garantir um monitoramento

remoto em tempo real, devem-se levar em consideração duas particularidades dos aplicativos multimídia: grande volume de dados e dependência temporal.

Segundo Conceição (2004, p. 2):

(...) Uma aplicação multimídia clássica consiste na transmissão de vídeo e som entre um servidor e uma aplicação cliente. Estas aplicações possuem certas peculiaridades que as diferenciam da maioria das aplicações, dentre as principais estão: grande volume de informações e dependência Temporal.

2.3.2 Multimídia: Sincronismo dos Quadros e Dependência Temporal

O dispositivo de vídeo utilizado neste trabalho permite gerar até 30 quadros por segundo, os quais serão transmitidos via Internet e vinculados a uma página HTML, este acoplamento dos quadros a um hipertexto caracteriza uma aplicação hipermídia que será abordada em outra seção.

Um ponto relevante numa aplicação onde o foco principal é a transmissão em tempo real de imagens é garantir ao usuário do sistema uma nítida percepção das ações ocorridas. Para tanto, uma aplicação que pretende transmitir vídeo em tempo real deve possibilitar o envio e a recepção de um número mínimo de quadros (cenas).

Segundo Conceição (2004, p. 3):

O conteúdo de vídeo exibido por uma aplicação multimídia baseia-se na exibição de imagens diversas vezes por segundo, gerando a impressão de movimento ao observador. Para exibir uma imagem em movimento é necessário fazer uma atualização da imagem pelo menos 20 vezes por segundo para se criar a sensação de movimento no observador.

Este contexto justifica a necessidade de um grande fluxo de dados, o que implica diretamente em uma taxa de transmissão razoável, a fim de garantir que o vídeo seja exibido de forma eficiente.

Segundo Paulo Filho (2000, p.284):

Uma imagem típica de vídeo, com qualidade de televisão, ocupa cerca de 512 X 480 pixels, ou seja, 240⁵ K pixels⁶. Em sistema de cor verdadeira, isto significa 3 bytes por pixels, dando um total 720 KB. Um segundo de vídeo compreende 30 quadros, dando um total de 21.600 KB.

⁵ 512 pixels X 480 pixels = 245.760 pixels / 1024 = 240 K pixels

⁶ Pixels são pontos em matriz de coordenada X e Y.

Segundo Conceição (2004, p.3):

Para se mostrar uma imagem preta e branca (onde 1 bit de informação define a cor de cada ponto), é necessário mostrar uma imagem completa a cada 1/20 segundos. Se esta imagem possui 640 X 480 pixels, então o volume de informação necessária para apresentar um segundo de filme é 20 X 640 X 480 bits, ou seja, é necessária uma taxa de transmissão de pelo menos 6 Mbps (milhões de bits por segundo).

Segundo Soares, Lemos e Colcher (1995, p. 499):

A vazão média gerada por uma fonte de vídeo varia com a qualidade do sinal, e os algoritmos de codificação, compactação e compressão empregados. Sistemas de vídeo apresentam informações como uma seqüência de quadros, sendo cada quadro composto de linhas. Um dos sistemas de distribuição de televisão mais usados usa 486 linhas por quadro a uma taxa de 30 quadros / segundo (padrão M). Nesses sistemas, se uma linha tiver 720 pontos (pixel – pictorial elements), sendo cada pixel codificado por 24 bits (como por exemplo, no padrão RGB⁷ 8-8-8) obteremos uma taxa de: 525 linhas X 30 quadros/seg. X 720 pixel 24 bits/pixel = 252 Mbps.

O dado referente aos números de pixels em relação ao número de bits que um determinado número de quadros/segundo pode gerar é demonstrado através da função ilustrada na Figura 2:

$$f: \rightarrow T_x = T_p * N_q$$

Figura 2 - Função da taxa de transmissão

Aonde T_x é a taxa de transmissão, T_p é o total dos pixels e N_q é o número de quadros que devem ser exibidos por segundo. Sendo que T_p é a soma dos pontos na matriz de pixels, vezes o número de bytes para cada pixel. Desta forma, T_p é igual a $x * y$ da matriz de pixels, vezes o número de bits determinado pela paleta de cores.

Na Figura 3 pode-se observar no gráfico da função que a taxa de transmissão ideal varia em função dos valores de $T_p * N_q$. Onde, a taxa de transmissão A é determinada por $T_x = 640 * 480 * 1(PB) * 20$, então $T_x = 6$ Mbps; e a taxa de transmissão B é determinada por: $T_x = 525 * 720 * 24(RGB) * 30$, então $T_x = 252$ Mbps.

⁷ RGB – Padrão de cores. A partir da mistura destas cores (Red – Vermelho, Green – Verde, Blue – Azul) básicas pode-se encontrar outras cores.

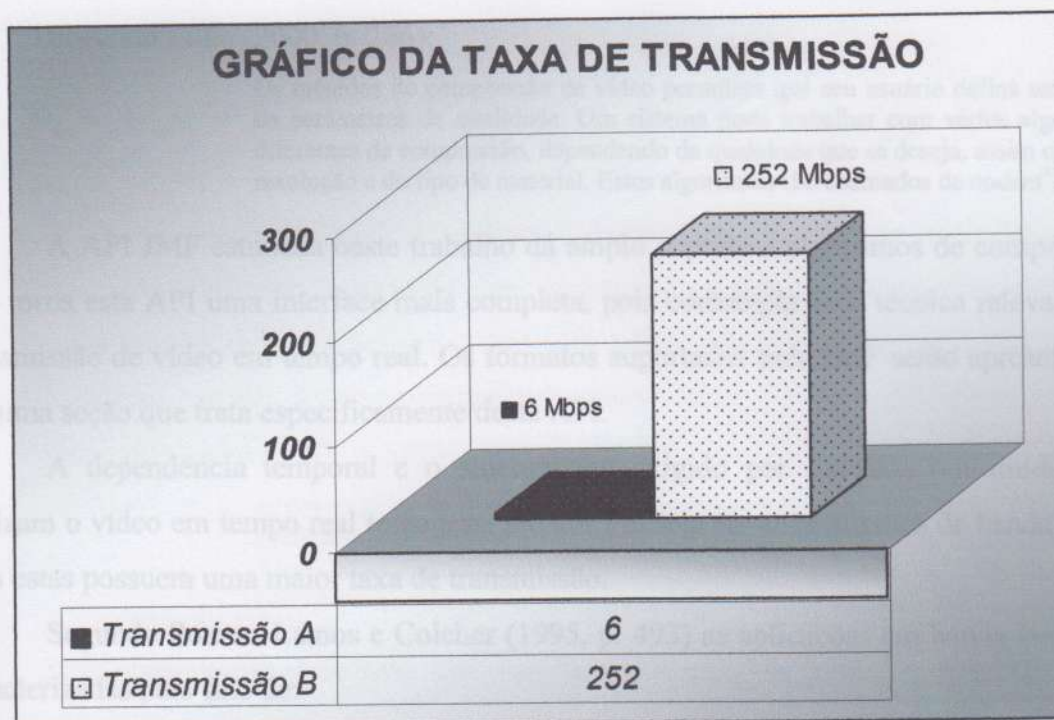


Figura 3 - Gráfico ilustrando a as taxas de transmissão

Os dados indicados pelo gráfico demonstram também que a razão da taxa de transmissão cresce tanto em função do número de quadros como em função do número de pixels e sua paleta de cores. Desta forma, devem ser levadas em consideração estas variáveis em relação à taxa de transmissão, pois estas podem elevar consideravelmente o tamanho de um vídeo.

Um outro problema abordado no exemplo “B” do gráfico é que para transmitir os 30 quadros por segundo é necessário 252 Mbps, o que pode tornar inviável a transmissão do vídeo, pois exige uma alta taxa de transmissão. Para resolver este problema na transmissão de vídeo em tempo real, devem ser utilizados algoritmos de compressão, pois estes contribuem efetivamente na redução do número de bytes para cada quadro gerado.

Segundo Conceição (2004, p. 3):

O conteúdo multimídia facilmente ultrapassa a ordem dos megabytes. Contudo, a natureza deste conteúdo é altamente redundante, o que possibilita a aplicação de algoritmos de compressão e grandes economias de espaço e de armazenamento e banda de transmissão.

Segundo Filho (2000, p. 286):

Os métodos de compressão de vídeo permitem que seu usuário defina uma série de parâmetros de qualidade. Um sistema pode trabalhar com vários algoritmos diferentes de compressão, dependendo da qualidade que se deseja, assim como da resolução e do tipo de material. Estes algoritmos são chamados de *codecs*⁸.

A API JMF estudada neste trabalho dá amplo suporte a algoritmos de compressão. Isto torna esta API uma interface mais completa, pois contempla uma técnica relevante na transmissão de vídeo em tempo real. Os formatos suportados pela JMF serão apresentados em uma seção que trata especificamente desta API.

A dependência temporal e o sincronismo exigido por sistemas multimídia que utilizam o vídeo em tempo real terão uma melhor eficácia em transmissões de banda larga, pois estas possuem uma maior taxa de transmissão.

Segundo Soares, Lemos e Colcher (1995, p. 493) as aplicações em banda larga são caracterizadas pelo fato de:

[...] lidar com objetos não convencionais (áudio e vídeo etc.). Não convencionais no sentido de serem objetos logos (1 minuto de vídeo não comprimido, qualidade TV, contém 1,8 Gbytes de dados), de exigirem transferência contínua de dados a altas taxas (252 Mbps para sinal de TV não comprimido) e, além de tudo, de exigirem acesso sincronizado aos dados.

A citação acima torna explícito o limitante das aplicações multimídia com relação à transmissão de vídeo em tempo real, pois estas se caracterizam como aplicações de banda larga. Isso é um forte indicador de que o fator temporal e a sincronia dos quadros de uma transmissão de vídeo em *real-time* serão garantidos por um meio adequado de transmissão, ou seja, por uma infra-estrutura que suporte aplicações de banda larga.

2.4 HIPERMÍDIA

Um hipertexto tem idéias contrárias ao texto normal, ou seja, não possui a idéia de seqüencialidade. O modo de leitura não é padronizado, sendo que o leitor pode escolher seu rumo dentro do texto. Normalmente, hipertexto se compara às páginas de Internet ou manuais eletrônicos de programas (BAIRON apud MACHADO, 2001, p. 5).

Um sistema de hipermídia segue a contextualização de um hipertexto, isso porque, todo conteúdo hipermídia está contido em uma página HTML. Sendo assim, os conteúdos de "multimídia web" (hipermídia) seguem a idéia de leitura não linear (não seqüencial).

⁸Codecs: Codificadores ou Decodificadores. Este termo foi tomado emprestado da telefonia (FILHO, 2000).

Hipermídia se baseia na integração de gráficos, som, vídeos ou qualquer combinação desses elementos em um sistema associativo, permitindo ao usuário fazer associações entre assuntos de forma não seqüencial (XIMENES apud MACHADO, 2001, p. 6).

A forma não seqüencial de leitura é basicamente uma intersecção entre um hipertexto e uma aplicação hipermídia, isso ocorre porque toda aplicação hipermídia está efetivamente contida em um hipertexto. Portanto, a diferença essencial entre ambos está no conteúdo exibido, ou seja, na forma de apresentação.

Segundo Comer (2001, p. 382):

A diferença entre hipertexto e hipermídia é fruto do conteúdo do documento; documentos de hipertexto contêm apenas informações textuais, enquanto que documentos de hipermídia podem conter representações adicionais de informações, incluindo imagens fotográficas digitalizadas ou gráficas.

Um outro ponto relevante das aplicações hipermídia é que assim como os aplicativos multimídia, estes podem ser aplicados em quatro contextos diferentes, tais como: vídeo conferência, *stream media* (fluxo de dados), *media player* (reprodutor de arquivos de mídia) e vídeos sob demanda. Isso ocorre porque toda aplicação hipermídia é efetivamente uma aplicação multimídia inserida no ambiente web.

2.4.1 Hipermídia e sua estrutura

A estrutura do arquivo hipermídia consiste na mesma idéia de estrutura de um hipertexto, aonde a nós e ligações. A ligação entre nós se torna um pouco diferente em relação ao hipertexto, pois no arquivo hipermídia a referência pode ser tanto para vídeo como para sons e imagens, sendo que essas ligações levam o usuário a acessar uma reprodução de trechos de vídeos, ou função multimídia (MACHADO, 2001, p. 6).

Um arquivo hipermídia para transmissão de vídeo em tempo real pode ser composto de vários nós e ligações. Neste trabalho, os principais “pontos de multimídias” (nó de ligação) serão objetos do tipo Applet⁹ Java, responsável por requisitar e exibir um fluxo de vídeo (*stream*) transmitido por um servidor que disponibilizará tal recurso.

⁹ Applet Java é um programa carregado a partir de um arquivo HTML, isto é, visualizado dentro de um navegador de Internet, que é executado do lado cliente, como se fosse uma aplicação *Desktop*.

2.4.2 Hipermídia: stream de vídeo

Um *stream* de vídeo¹⁰ é de fato uma cadeia de *bytes* que após serem lidos por um aplicativo irão gerar vídeo. Para que ocorra este envio dos *bytes* de vídeo é necessário um escritor e um leitor de bytes. O escritor é o servidor de *stream* de vídeo que ficará transmitindo os bytes gerados a partir de um dispositivo de vídeo (*webcam*) ou de um arquivo. O leitor é um aplicativo que captura estes fluxos de dados (bytes) gerados, para partir destes, constituir o vídeo apresentando-o a um usuário, ou simplesmente gravando-os em disco.

“Um stream representa um fluxo de dados, ou um canal de comunicação com (pelo menos em conceito) um escritor em uma ponta e um leitor em outra” (NIEMEYER; KNUDSEN, 2000, p. 265).

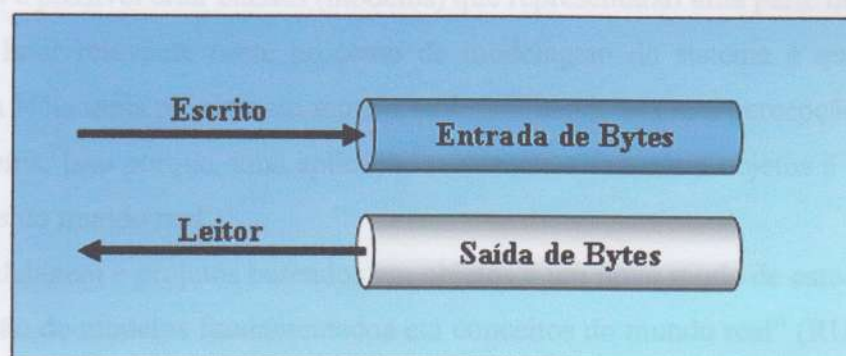


Figura 4 - Leitor e escritor de *stream*

“No caso do *stream* o conteúdo é executado sem que haja necessidade de trazê-lo por inteiro para o lado do cliente. Assim ocorre a transmissão por meio de fluxo contínuo de pacotes e reprodução simultânea” (MACHADO, 2001, p. 9).

A transmissão de fluxo contínuo e a reprodução simultânea em sistemas de *stream* media podem garantir a uma aplicação a transmissão de eventos em tempo real, o que é desejável para aplicações cuja a função é a transmissão de vídeo ao vivo via Internet.

A aplicação desenvolvida por este trabalho é um sistema onde a transmissão é ao vivo, ou seja, o usuário recebe todo o *stream* de vídeo de forma passiva, de tal maneira que a aplicação cliente não poderá controlar as informações. Como por exemplo: avançar ou voltar o vídeo, pelo fato das imagens serem enviadas no instante em que são criadas.

¹⁰ Um *stream* de vídeo pode ser visto como uma cadeia ou sequência de *bytes*.

2.5 PROJETO ORIENTADO A OBJETOS

Um sistema orientado a objetos é uma coleção de classes utilizadas como base para criação de seus respectivos objetos que uma vez instanciados cooperam entre si, implementado desta forma as funcionalidades da aplicação.

[...] Cada objeto tem sua própria identidade, que lhe é inerente. Em outras palavras, dois objetos são distintos mesmo que todos os valores de seus atributos (como nome e tamanho) sejam idênticos.

No mundo Real um objeto limita-se a existir, mas no que se refere à linguagem de programação, cada objeto dispõe de um único indicador, pelo qual ele pode ser referenciado inequivocamente (RUMBAUGH et al., 1994, p. 2).

A orientação a objetos permite aos projetistas de sistemas, mapear de forma classificatória as principais entidades envolvidas no domínio do problema. A partir desta classificação é possível criar classes (modelos) que representarão uma parte do problema.

Um fator relevante neste processo de modelagem do sistema é que os modelos seguem uma idéia mais próxima do mundo real, facilitando assim a percepção do projetista sobre o sistema. Isso porque, uma aplicação realmente orientada a objetos é fundamentada em conceitos do mundo real.

“Modelagem e projetos baseados em objetos é um novo modo de estudar problemas com utilização de modelos fundamentados em conceitos do mundo real” (RUMBAUGH et al. 1994, p. 1).

O paradigma da programação orientada a objetos incorpora em sua estrutura básica um conceito muito forte de modularidade. Igualmente, outro fator relevante na programação orientada a objetos é a possibilidade de criar componentes de software independentes. Desta forma, provendo uma maior reusabilidade de código e ampliando a escalabilidade do sistema.

“As qualidades¹¹ mais freqüentes observadas em sistemas construídos no modo orientado a objetos são: reutilização, confiabilidade, robustez, extensibilidade, distributividade e armazenabilidade” (PAGE-JONES, 2001, p. 68).

¹¹ As definições mais específicas sobre reutilização, confiabilidade, robustez, extensibilidade, distributividade e armazenabilidade de software podem ser encontradas na obra de PAGE-JONES, 2001, pág. 68.

A modelagem do sistema proposto nesta monografia se fundamenta na proposta de Larman (2002), no qual cita três passos importantes, estes são: análise, projeto e implementação.

- **Análise:** É toda a investigação do problema em torno da transmissão de vídeo em tempo real (conhecimento do domínio do problema);
- **Projeto:** É a solução lógica para o sistema proposto. Na qual está baseada no uso de diagramas fundamentados na teoria da orientação a objetos.
- **Implementação:** É efetivamente a codificação do sistema que toma por base o “desenho do software” (modelagem) proposto no projeto.

Na Figura 5 é possível observar a proposta de Larman (2002, p. 30) para o desenvolvimento de sistemas orientados a objetos:

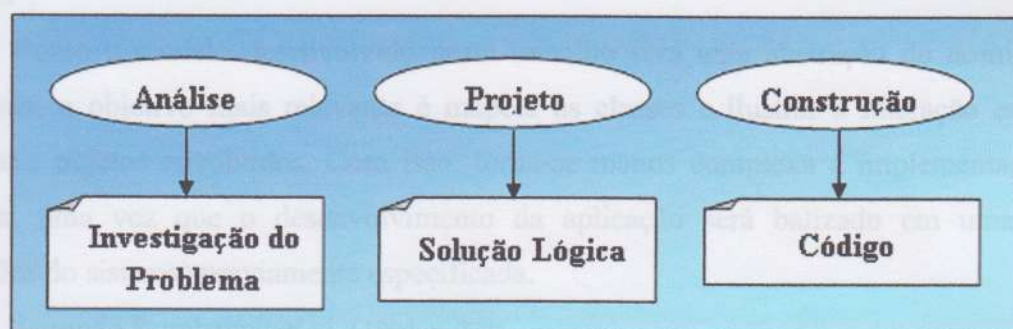


Figura 5 - Atividades de desenvolvimento (LARMAN 2002, p. 30)

“A essência da análise e do projeto orientados a objetos é enfatizar a consideração de um domínio de problema e uma solução lógica, segundo a perspectiva de objetos” [...] (LARMAN, 2002, p. 30).

No processo de modelagem orientada a objetos é importante observar que o modelo concebido é uma proposta conceitual que provêm de uma análise do problema proposto. Portanto este modelo é independente de uma linguagem de programação. Desta forma, a modelagem de sistema orientado a objetos não especifica nenhuma técnica de programação e sim uma nova forma de ver o problema.

Rumbaugh, et al, (1994, p.5), observa que:

O desenvolvimento baseado em objetos é um processo conceitual independente de uma linguagem de programação até as etapas finais. O desenvolvimento baseado em objetos é fundamentalmente uma nova maneira de pensar e não uma técnica de programação. Seus maiores benefícios provêm do auxílio prestado aos especificadores, desenvolvedores e clientes em expressarem com clareza conceitos abstratos. Ele pode servir como um meio para especificação, análise, documentação, interface e para programação.

Objetivando modelar o sistema proposto em torno do objeto de estudo optou-se por utilizar a UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada) como linguagem de modelagem, pois esta permite especificar, visualizar e construir os artefatos de software, usando os conceitos da orientação a objetos.

Segundo Rumbaugh, et al. (1994, p. 23), “Um modelo é uma abstração de alguma coisa, cujo propósito é permitir que se conheça essa coisa antes de se construí-la. Como um modelo omite detalhes não-essenciais, sua manipulação é mais fácil do que a entidade original”.

Como o modelo desenvolvido neste trabalho será uma abstração do domínio do problema, o objetivo mais relevante é mapear as classes e ilustrar a interação entre os principais objetos envolvidos. Com isso, torna-se menos complexa a implementação do sistema, uma vez que o desenvolvimento da aplicação será balizado em uma visão (notações do sistema) previamente especificada.

Segundo Rumbaugh et al. (1994, p. 23):

Para se construir sistemas complexos, o desenvolvedor deve abstrair diferentes visões do sistema, construir modelos com utilização de uma notação precisa, verificar se os modelos satisfazem os requisitos do sistema e acrescentar detalhes gradativamente para transformar os modelos em uma implementação.

Mapear as diferentes visões do sistema é uma metodologia que irá prover uma compreensão mais ampla e detalhada da aplicação proposta e simultaneamente é uma forma de documentação mais consistente. Neste trabalho, a modelagem é verificada com a implementação. Isto garante a documentação uma melhor precisão, pois à medida que novos detalhes surgem estes também serão adicionados ao modelo. Sendo assim, toda a modelagem construída da aplicação será finalizada com as verificações finais do sistema.

2.6 PLATAFORMA JAVA

Em maio de 1995, a Sun anunciou Java formalmente em uma conferência importante. Normalmente, um evento como esse não teria gerado muita atenção. Entretanto, Java gerou interesse imediato na comunidade comercial por causa do fenomenal interesse pela Word Wide Web. Java é agora utilizada para criar páginas da Web com conteúdos interativos e dinâmicos, para desenvolver aplicativos corporativos de larga escala, para aprimorar a funcionalidade de servidores da Word Wide Web (os computadores que fornecem o conteúdo que vemos em nossos navegadores da Web), fornecer aplicativos para dispositivos destinados ao consumidor final (como telefones celulares, *paggers* e PDAs) e assim por diante (DEITEL, 2001, p. 64).

Permitir uma coesão consistente entre o projeto (modelo) e a linguagem Java é uma das contribuições deste trabalho, pois poderá servir de base para outros estudos que tenham uma finalidade semelhante. Para tanto, será realizado um estudo que se fundamenta em detalhar os recursos desta linguagem tendo como foco o objeto de estudo da pesquisa.

O embasamento teórico da linguagem Java irá contribuir para um entendimento mais amplo do trabalho no que diz respeito à implementação do sistema, pois será à base de toda a pesquisa.

Um fator importante em modelar um sistema baseado nos recursos da linguagem Java é a independência que o sistema pode assumir em relação às mais diversas plataformas de hardware¹² e de sistemas operacionais.

A linguagem de programação Java, desenvolvida pela Sun Microsystems sob a orientação dos astros da Net, James Gosling e Bill Joy, foi projetada para ser uma linguagem de programação independente de máquina, que seja segura o suficiente para atravessar redes e poderosa o bastante para substituir o código executável nativo (NIEMEYER; KNUDSEN, 2000, p. 2).

Poder desenvolver um sistema independente de hardware e de uma plataforma de sistema operacional é de fato um ponto substancial desta linguagem de programação. Esta liberdade proposta pela linguagem Java possibilita um maior grau de reusabilidade. Isso porque, a possibilidade ampliada de reutilização de códigos permite aos sistemas migrarem facilmente para outro ambiente sem alterar uma única linha de código.

O nível de reutilização de código que a linguagem de programação Java possibilita é desejável quando se está desenvolvendo sistemas, pelo fato de aumentar a produtividade e

¹² Hardware no sentido de que a linguagem Java possibilita ao programador desenvolver para os mais diversos tipos de dispositivos eletrônicos. Para obter maiores informações, consulte o site da SUN (www.java.sun.com).

também por dar uma maior independência ao desenvolvedor quanto ao ambiente aonde irá executar a aplicação, ou seja, o programador codifica os seus sistemas sem se preocupar com o sistema operacional aonde irá implantá-lo.

Desenvolver um sistema que funciona em qualquer ambiente é bastante confortável para quem está programando; e Java dá esta possibilidade, pois para o programador deixa de existir a preocupação com código executável nativo uma vez que o código não fica preso às restrições de uma plataforma específica.

A linguagem de programação Java tem várias características importantes, as quais foram de suma relevância para escolha desta enquanto objeto de estudo, há exemplo podem ser citadas:

- Linguagem totalmente orientada a objetos;
- Vasto número de APIs;
- Portabilidade;
- Segurança;
- Distributividade.

Nesta seção, são utilizadas abordagens de alguns autores sobre esta linguagem de programação, a fim de fundamentar teoricamente estes argumentos.

Deitel (2001, p. 65), na citação abaixo, faz alusão algumas das características da linguagem Java, tais como: orientação a objetos e as bibliotecas de classes Java (APIs):

Programas Java consistem em partes chamadas classes. Estas consistem em partes chamadas métodos que realizam tarefas e retornam as informações ao complementarem suas tarefas. Você pode programar cada pedaço que talvez você precise para formar um programa Java. No entanto, mais programadores Java tiram proveito de ricas coleções de classes existentes em bibliotecas de classes Java. As bibliotecas de Java também conhecidas como APIs. Portanto, há realmente duas partes para aprender o "mundo" de Java [...].

Uma das principais características de uma linguagem orientada a objetos é prover herança, ou seja, permitir que uma classe possa herdar características pertinentes de uma outra classe.

Segundo Cesta e Rubira (2002, p. 8):

Java suporta herança, mas não herança múltipla. A ausência de herança múltipla pode ser compensada pelo uso de herança e interfaces, onde uma classe herda comportamento de sua superclasse além de oferecer uma implementação para uma ou mais interfaces.

Utilizar a Plataforma Java no processo de construção de software é pertinente por dois motivos:

- Fidelidade entre o modelo e a implementação. Neste sentido, mantendo uma ligação lógica natural entre os arquétipos¹³ e a codificação;
- Maior produtividade, pois a linguagem provê recursos nativos, como as APIs. Exemplo de uma API relevante para este trabalho é a *Java Media Framework*, que será discutida em uma seção específica.

2.6.1 Portabilidade

A linguagem de programação Java foi projetada para ser uma linguagem portátil. A portabilidade que esta linguagem provê ocorre em função da JVM (*Java Virtual Machine* – Máquina Virtual Java) que simula um processador real.

De fato, o que JVM faz é interpretar *byte-codes*¹⁴. Hoje, já existem interpretadores (máquina virtual) Java para vários ambientes operacionais, tais como, Unix, Linux, Solaris, Macintosh, Windows, entre outros. Para que a JVM interprete os *byte-codes* é necessário que ocorra primeiro a compilação do código Java, ou seja, os *byte-codes* serão gerados pelo compilador Java e só depois poderão ser executados por uma JVM.

O processo de interpretação dos *byte-codes* pela máquina virtual, que ocorre em runtime (tempo de execução), deixa a linguagem um pouco mais lenta em relação às outras linguagens de programação que geram programas executáveis, como por exemplo, C e C++.

“Historicamente, os interpretadores têm sido considerados mais lentos, mas como o interpretador Java executa o código de bytes compilado, Java é uma linguagem interpretada de forma relativamente rápida” (NIEMEYER; KNUDSEN, 2000, p. 6).

Obviamente o retardo gerado pelo processo de interpretação pode ser compensado pelo alto grau de portabilidade que Java oferece, pois isso reduz significativamente a necessidade de reescrever código. A Figura 6 ilustra o esquema de portabilidade da linguagem Java.

¹³ São considerado arquétipo os artefatos provenientes de uma modelagem, ou seja, o modelo do sistema.

¹⁴ Byte-codes – Código em bytes.

Cesta e Rubira (2002, p. 9) acreditam que a linguagem Java irá otimizar sensivelmente o tempo de interpretação e execução:

Como Java foi criada para ser usada em computadores pequenos, ela exige pouco espaço, pouca memória. Java é muito mais eficiente que grande parte das linguagens de "scripting" existentes, embora seja cerca de 20 vezes mais lentas que C, o que não é um marco definitivo. Com a evolução da linguagem, serão criados geradores de "byte-codes" cada vez mais otimizados que trarão as marcas de performance das linguagens mais próximas das C++ e C.

A portabilidade da linguagem Java é um instrumento fundamental na construção de aplicações que serão distribuídas na Web, pois diminuem consideravelmente o número de "restrições". Neste trabalho, o fator portabilidade é relevante, uma vez que a aplicação que será apresentada ao usuário final é um sistema hipermídia, ou seja, rodará num browser da Internet e este poderá estar funcionando em qualquer ambiente operacional.

"A portabilidade é uma das características que se inclui nos objetivos almejados por uma linguagem orientada a objetos. Em Java ela foi obtida de maneira inovadora com relação ao grupo atual de linguagens orientadas a objetos" (CESTA; RUBIA, 2002, p. 8).

Também na Figura 6, é possível observar exatamente o que Cesta e Rubira (2002, p. 8) dizem a respeito da portabilidade dos sistemas desenvolvidos em Java o que é reforçado pelos autores nesta citação:

Java foi criada para ser portátil. O "byte-codes", que é gerado pelo compilador para sua aplicação específica pode ser transportado entre plataformas distintas que suportam Java (Solaris 2.3, Windows NT, Mac/Os, etc). Não é necessário recompilar um programa para que ele rode numa máquina e sistema diferente, ao contrário do que acontece, por exemplo, com programas escritos em C.

A portabilidade oferecida pela linguagem Java é utilizada na implementação deste trabalho, a fim de alcançar do aplicativo hipermídia construído o máximo de flexibilidade, evitando assim, conflitos com os sistemas operacionais. Essa portabilidade torna o sistema muito mais acessível, pois a aplicação terá mais flexibilidade no que se refere ao usuário final, pois o mesmo poderá acessar o sistema de "qualquer ambiente operacional".

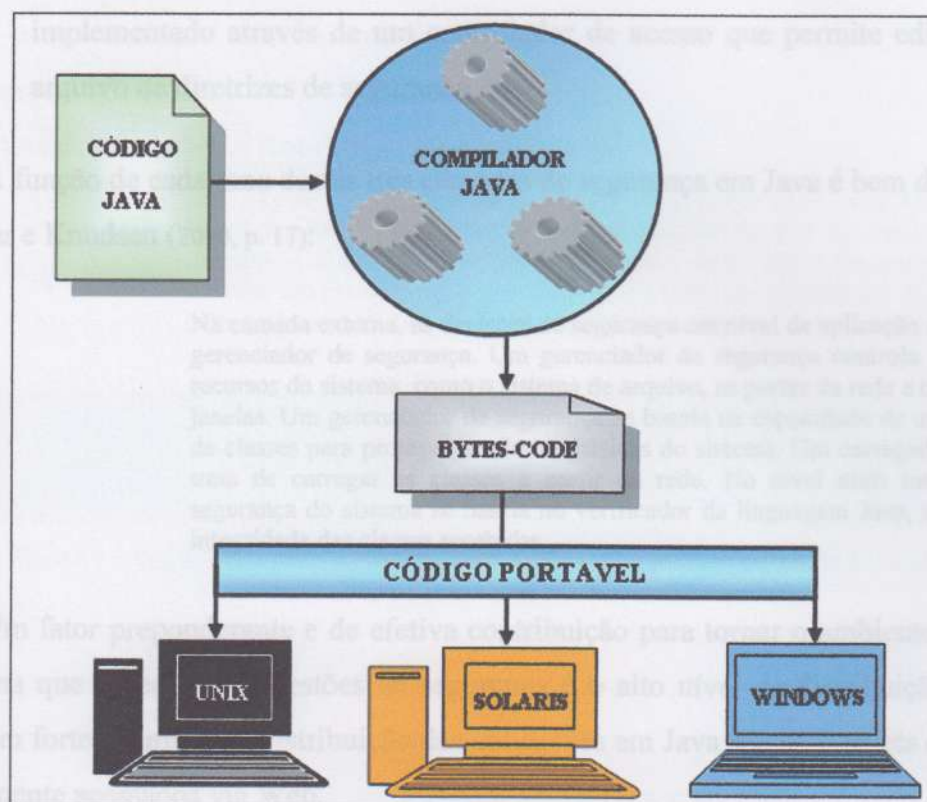


Figura 6 - Esquema de portabilidade da plataforma Java

2.6.2 Segurança, distributividade e Applets

A linguagem de programação Java possibilita o desenvolvimento de sistemas de grande porte e também permite a distribuição destes através da web. Desta forma, o ambiente Java prover meios para se construir aplicações seguras.

Em Java a segurança é tratada em vários níveis, tais como: verificador, carregador de classes e gerenciador de segurança. Estes podem ser definidos como:

- Verificador de byte-codes: Verifica os códigos binários antes de executá-los, caso não exista código malicioso a máquina virtual Java executa o código.
- Carregador de classes: Carrega classes de rede em memória e passa pelo verificador. Esta técnica permite associar as classes a um destino específico e desta forma, criar um ambiente restrito, ou seja, determinar que estas classes irão interagir com classes vindas apenas da mesma origem.
- Gerenciador de segurança: É responsável por analisar e tomar decisões de segurança em nível de aplicação. Este gerenciador de segurança no Java 2 é

implementado através de um controlador de acesso que permite editar em um arquivo de diretrizes de segurança.

A função de cada uma destas três camadas de segurança em Java é bem definida por Niemeyer e Knudsen (2000, p. 17):

Na camada externa, as decisões de segurança em nível de aplicação são feitas por gerenciador de segurança. Um gerenciador de segurança controla o acesso aos recursos do sistema, como o sistema de arquivo, as portas de rede e o ambiente de janelas. Um gerenciador de segurança se baseia na capacidade de um carregador de classes para proteger as classes básicas do sistema. Um carregador de classes trata de carregar as classes a partir da rede. No nível mais interno, toda a segurança do sistema se baseia no verificador da linguagem Java, que garante a integridade das classes recebidas.

Um fator preponderante e de efetiva contribuição para tornar o ambiente Java uma plataforma que especifique questões de segurança é o alto nível de distribuição que esta provê. Um forte exemplo da distribuição das aplicações em Java são os Applets que podem ser facilmente acessados via Web.

Um *Applet* em Java é uma aplicação que pode rodar em uma máquina cliente independente de sua localização geográfica, bastando para isso, estar conectado a Internet e ter uma máquina virtual Java instalada.

“O termo ‘*Applet*’ é usado para significar uma pequena aplicação subordinada ou incorporável. Com ‘incorporável’, queremos dizer que ela é projetada para ser executada dentro do contexto de um sistema maior” (NIEMEYER; KNUDSEN, 2000, p. 23).

Segundo Niemeyer e Knudsen (2000, p. 598):

Pragmaticamente, um *Applet* é um intruso no ambiente de alguma pessoa e, portanto, precisa ser tratado com suspeita. Os browsers da Web que executam Applets impõem restrições sobre o que ele pode fazer. As restrições são impostas por um gerenciador de segurança de Applet, sendo que os Applets não assinados não podem alterá-lo. O browser também oferece um “contexto de *Applet*”, que é o suporte adicional que ajuda o *Applet* a residir dentro de suas restrições. A não ser por essa estrutura de alto nível e as restrições de segurança, não existe diferença entre um *Applet* e uma aplicação.

Desenvolver uma aplicação em Java utilizando a tecnologia Applets tem duas implicações diretas: distributividade e restrições de segurança.

Uma aplicação baseada em *Applet* geralmente é projetada para ser de fácil distribuição, esta distributividade existe em função da sua vinculação a um navegador da

Internet, o que aumenta consideravelmente o seu poder de distribuição, haja vista a popularização da Internet.

Segundo Niemeyer e Knudsen (2000, p. 597):

Um Applet é parte de uma página da Web, assim como uma imagem ou um hiperlink. Ele “possui” alguma área retangular da tela do usuário. Ele pode desenhar o que quiser e responder a eventos do teclado e do mouse nessa área. Quando um browser da Web carrega uma página que contém um Applet Java, ele sabe como carregar as classes do Applet e executá-las.

```
<HTML>  
<APPLET code = "Simorew.class" width = 320 Height = 240 ></APPLET>  
</HTML>
```

Figura 7 - Tag HTML para adicionar um Applet a uma a uma página da Internet

Efetivamente um *Applet* Java é uma classe especial que pode voitar de uma máquina servidora a uma maquina cliente, e lá executar métodos que de alguma forma irão acionar algum tipo de interação com o usuário. Como esta tecnologia é dotada de um alto poder de distribuição, a sua arquitetura é fundamentada em restrições, a fim de garantir ao usuário um determinado nível de segurança e confiabilidade na aplicação.

Neste trabalho, a tecnologia *Applet* é à base da aplicação hipermídia, uma vez que possibilita a construção de aplicações multimídia agregada a um hipertexto. A opção por esta tecnologia foi determinada em função da possibilidade que a mesma disponibiliza para a construção de aplicações de fácil distribuição através da *Web*, tal qual são os *Applet* Java.

Um outro fator relevante na escolha da tecnologia *Applet* foi à possibilidade de se construir uma aplicações hipermídia mais sofisticada que podesse executar do lado cliente garantido um bom nível de interação em temos de multimídia.

Como a tecnologia *Applet* impõe fortes restrições relacionadas à segurança, questões como a execução devida da aplicação é pertinente ao estudo de trabalhos que pretendem utilizá-las. Neste sentido torna-se substancial uma abordagem nas técnicas de assinatura digital, a fim de resolver os problemas de restrições impostas pelas *Applet* Java. Neste trabalho, a questão de utilizar ou não um técnica de assinatura digital será verificada na implementação, o qual constará nos resultados deste trabalho monográfico.

2.6.3 Assinatura Digital

A assinatura digital é uma forma de resolver alguns problemas de segurança impostos pelos *browsers* da *web* aos *Applets* Java, tais como:

- Não poder ler ou gravar arquivos no *host* local;
- Não poder abrir estabelecer uma conexão de rede via soquete, a menos que a conexão seja com o servidor de origem;
- Não poder iniciar outro processo no *host* local;
- Não poder ter métodos nativos.

Segundo Niemeyer e Knudsen (2000, p. 600):

Os *applets* ficam presos dentro do *browser* por um *SecurityManager* de *applet*. O *SecurityManager* faz parte do *browser* da Web ou visualizador de *applet*(*AppletView*). Ele é instanciado antes que o *browser* carregue quaisquer *applets* e implementam as restrições básicas que permitem ao usuário executar *applets* não confiáveis com segurança.

Os sistemas que utilizam *applets* na construção de aplicações hipermídia para reprodução de vídeo podem sofrer restrições com relação à conectividade. A restrição poderá existir quando o *applet* Java tentar estabelecer uma conexão com servidor de *stream* de vídeo, caso este servidor de vídeo não seja o mesmo que hospeda o *applet*. Esta restrição ocorre porque um *applet* só pode estabelecer uma conexão com o seu servidor de origem a menos que este *applet* seja assinado, ou seja, possua uma assinatura digital.

Para Niemeyer e Knudsen (2000, p. 614), “uma assinatura digital, viabiliza um modo de autenticar documentos e outros dados. Sendo assim, é possível a um *Applet* garantir ao usuário a sua origem”.

Tomando por base as restrições que os *Applets* Java podem sofrer é importante realizar esta abordagem com relação assinatura digital, pois poderão se fazer necessários em ambientes aonde existem uma distinção entre a localização dos vídeos e dos *applets*, ou seja, o servidor de *stream* de vídeo não é o mesmo do *applet*¹⁵. Este tipo de situação determina que o *applet* abra uma conexão com servidor que não é de sua origem, o que

¹⁵ Para obter maiores informações sobre assinatura digital devesse ler a obra *Aprendendo Java* de Niemeyer e Knudsen, páginas de 614 a 625.

obriga o *applet* a ter uma assinatura digital a fim de garantir que mesmo possa reproduzir o vídeo desejado.

2.7 JMF – JAVA MEDIA FRAMEWORK

A JMF é uma API Java que foi projetada para dar suporte a aplicações multimídia. Para tanto, ela fornece subsídios para implementação de aplicações que utilizem recursos multimídia mais específicos, tais como: vídeo e som.

Segundo a SUN (2004):

“O Java Media Framework (JMF) fornece uma arquitetura unificada e um protocolo de mensagens para controlar a aquisição, processamento, e a exibição de dados mídia baseados em tempo. A JMF é projetada para suportar a maioria dos tipos de mídia padrão, tais como: o AIFF, o AU, o AVI, a G/M, o MIDI, o MPEG, o QuickTime, o RMF, e o WAV”.

Segundo Deitel (2002, p. 1113) a Sun:

Reconhecendo que os aplicativos Java deveriam suportar recursos de vídeo e áudio digitais, a Sun Microsystems, a Intel e a Silicon Graphics trabalharam juntas para produzir uma API para multimídia que é reconhecida como a Java Mídia Framework (JMF). A API JMF é uma das várias APIs para multimídia em Java. Usando a API JMF, os programadores podem criar aplicativos Java que reproduzem, editam e capturam muitos tipos populares de mídia.

Como a JMF foi criada para suportar os formatos de mídia mais conhecidos, os projetos que pretendem usá-la devem conhece-los previamente, pois desta forma pode-se minimizar os conflitos com relação aos tipos de mídia suportada pela API.

Uma particularidade da API JMF é que existem três versões para instalação, sendo estas: *Cross Platform Version*, *Solaris/Linux Performance Pack* e *Windows Performance Pack*. Um fator importante com relação às versões da JMF são tipos de mídias que estas suportam. Na Tabela 1, extraída da documentação da Sun (2004) é feita uma disposição de versão da JMF por tipo de mídia e o formato em que estas podem ser codificadas ou decodificadas, o que torna mais fácil a visualização.

Tabela 1 - Tipos de mídia suportadas pela JMF

Media Type	JMF 2.1.1 Cross Platform Version	JMF 2.1.1 Solaris/Linux Performance Pack	JMF 2.1.1 Windows Performance Pack
AIFF (.aiff)	read/write	read/write	read/write
8-bit mono/stereo linear	D,E	D,E	D,E
16-bit mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
A-law	D	D	D
IMA4 ADPCM	D,E	D,E	D,E
AVI (.avi)	read/write	read/write	read/write
Audio: 8-bit mono/stereo linear	D,E	D,E	D,E
Audio: 16-bit mono/stereo linear	D,E	D,E	D,E
Audio: DVI ADPCM compressed	D,E	D,E	D,E
Audio: G.711 (U-law)	D,E	D,E	D,E
Audio: A-law	D	D	D
Audio: GSM mono	D,E	D,E	D,E
Audio: ACM** ¹⁶	-	-	D,E
Video: Cinepak	D	D,E	D
Video: MJPEG (422)	D	D,E	D,E
Video: RGB	D,E	D,E	D,E
Video: YUV	D,E	D,E	D,E
Video: VCM** ¹⁷	-	-	D,E
GSM (.gsm)	read/write	read/write	read/write
GSM mono audio	D,E	D,E	D,E
HotMedia (.mvr)	read only	read only	read only
IBM HotMedia	D	D	D
MIDI (.mid)	read only	read only	read only
Type 1 & 2 MIDI	-	D	D
MPEG-1 Video (.mpg)	-	read only	read only
Multiplexed System stream	-	D	D
Video-only stream	-	D	D
MPEG Layer II Audio (.mp2)	read only	read/write	read/write
MPEG layer 1, 2 audio	D	D,E	D,E
QuickTime (.mov)	read/write	read/write	read/write
Audio: 8 bits mono/stereo linear	D,E	D,E	D,E
Audio: 16 bits mono/stereo linear	D,E	D,E	D,E
Audio: G.711 (U-law)	D,E	D,E	D,E
Audio: A-law	D	D	D
Audio: GSM mono	D,E	D,E	D,E

¹⁶ ACM** - Window's Audio Compression Manager support. Tested for these formats: A-law, GSM610, MSNAudio, MSADPCM, Truespeech, mp3, PCM, Voxware AC8, Voxware AC10.

¹⁷ VCM** - Window's Video Compression Manager support. Tested for these formats: IV41, IV51, VGPX, WINX, YV12, I263, CRAM, MPG4.

Audio: IMA4 ADPCM	D,E	D,E	D,E
Video: Cinepak	D	D,E	D
Video: H.261	-	D	D
Video: H.263	D	D,E	D,E
Video: JPEG (420, 422, 444)	D	D,E	D,E
Video: RGB	D,E	D,E	D,E
Sun Audio (.au)	read/write	read/write	read/write
8 bits mono/stereo linear	D,E	D,E	D,E
16 bits mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
Wave (.wav)	read/write	read/write	read/write
8-bit mono/stereo linear	D,E	D,E	D,E
A-law	D	D	D
16-bit mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
A-law	D	D	D
GSM mono	D,E	D,E	D,E
DVI ADPCM	D,E	D,E	D,E
MS ADPCM	D	D	D
ACM**	-	-	D,E

Fonte: Sun Microsystems, 2004.

- **D** indica que o formato pode ser decodificado e apresentado.
- **E** indica que o *stream* de mídia pode ser codificado no formato.
- **read** indica que o tipo de mídia pode ser usado como entrada (lê a partir de um arquivo).
- **write** indica que o tipo de mídia pode ser gerado como saída (escrita para um arquivo).

A API JMF também suporta os mais diversos tipos de dispositivos de captura de vídeo e som. Estes dispositivos podem ser perfeitamente acoplados a qualquer sistema multimídia desenvolvido em Java que tenha por finalidade capturar vídeo e/ou áudio. Na Tabela 2, é possível visualizar quais dispositivos são suportados pela API.

Tabela 2 - Tipos de dispositivos suportado pela JMF

Capturer	JMF 2.1.1 Cross Platform Version	JMF 2.1.1 Solaris Performance Pack	JMF 2.1.1 Windows Performance Pack
JavaSound (16-bit, 44100, 22050, 11025Hz, 8000Hz linear)	X (J2SE 1.3+)	X	X
SunVideo	-	X	-
SunVideoPlus	-	X	-
VFW	-	-	X
Intel Create & Share	-	-	Win9x
Diamond Supra Video Kit; Share	-	-	Win98
QuickCam VC (camera)	-	-	WinNT
e-cam (camera)	-	-	WinNT, 9X
Winnow Videum	-	-	WinNT, 9X
Creative Web Cam II	-	-	Win9X

Miro Video DC30	-	-	Win9X
lomega Buz	-	-	Win9X
QuickCam Home USB (Camera)	-	-	Win98
Smart Video Recorder III	-	-	Win9X
Fonte: Sun Microsystem, 2004.			

Segundo a SUN(2004):

“API JMF utiliza uma estrutura de alto nível parecido com os modelos convencionais de reprodução de mídia baseada em tempo, onde é necessário uma origem de dados (dispositivos de captura), um dados, um reproduutor de mídia e os dispositivos de saída, tais como caixas de som e / ou televisão.”

A Figura 8, ilustra através de um processo convencional, como funciona a estrutura de alto nível da API JMF.

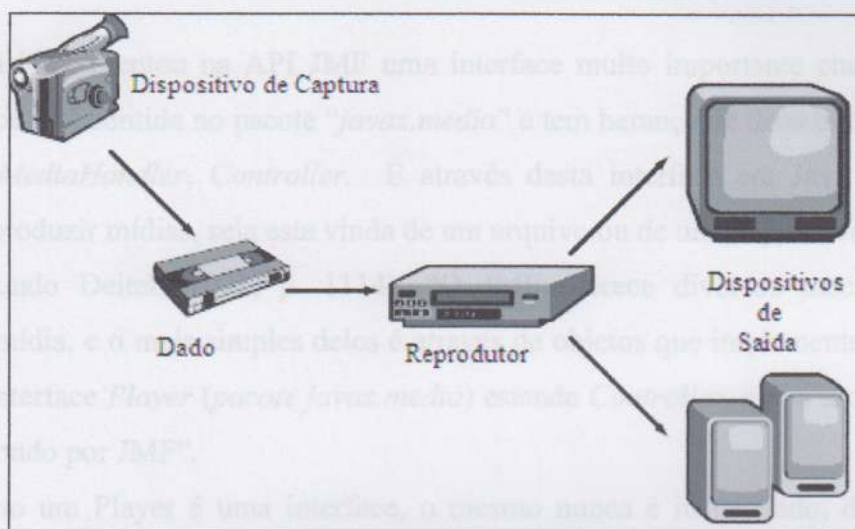


Figura 8 - Modelo de reprodução de mídias baseadas em tempo

As aplicações desenvolvidas com base na JMF devem ser construídas utilizando todo o nível de abstração que esta API fornece para o desenvolvimento de aplicações multimídia. Neste sentido, a JMF encapsula detalhes de baixo nível, tornando estes transparentes aos desenvolvedores. A Figura 9 retirada do guia de desenvolvimento da SUN (2004) ilustra o posicionamento da JMF na construção de aplicações multimídia que utilizam esta API.

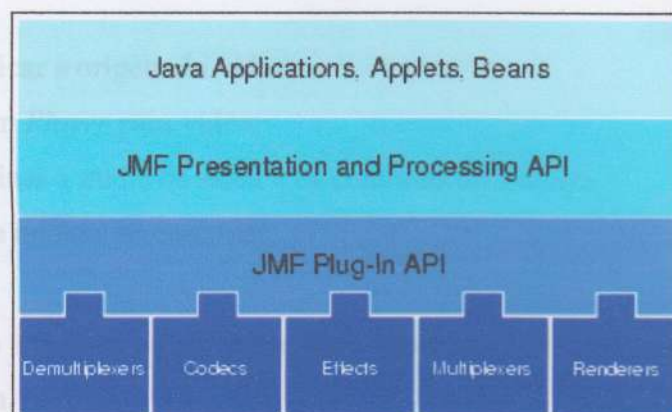


Figura 9 - Posicionamento da JMF na construção de aplicações, (Fonte: SUN, 2004).

2.7.1 Reproduzindo mídias com JMF

Java implementou na API JMF uma interface muito importante chamada *Player*, esta interface está contida no pacote "*javax.media*" e tem herança de duas outras interfaces, estas são: *MediaHandler*, *Controller*. É através desta interface em Java que se torna possível reproduzir mídias, seja esta vinda de um arquivo ou de um dispositivo de captura.

Segundo Deitel (2002, p. 1114), "O JMF oferece diversos mecanismos para reproduzir mídia, e o mais simples deles é através de objetos que implementam a interface *Player*. A interface *Player* (pacote *javax.media*) estende *Controller*, que é um tratador para mídia suportado por JMF".

Como um *Player* é uma interface, o mesmo nunca é instanciado, desta forma, a criação de um objeto do tipo *Player* é subsidiada por uma outra classe que permite realizar tal procedimento. "Os *Player* são distribuídos por uma classe de alto nível no JMF, chamada de *Manager*" (NIEMEYER E KNUDSEN, 2000, p. 557).

Um *Player* só pode ser criado se existir efetivamente uma fonte de mídia, ou seja, a mídia deve existir de fato, seja esta um arquivo ou um dispositivo de captura. Isso porque, um *Player* além da necessidade de confirmar a existência da fonte de mídia, precisa também verificar a compatibilidade da mídia com o JMP (*Java Media Player*).

"Uma forma de obter um *Player* é especificar o URL do Filme" (NIEMEYER; KNUDSEN, 2000, P. 557).

Com base em Deitel (2004, p. 1114), para reproduzir um vídeo são necessários efetivamente quatro passos:

1. Especificar a origem do vídeo;
2. Criar um *Player* para vídeo;
3. Determinar a mídia de saída e os controles do *Player*;
4. Exibir a mídia e os controles.

No Quadro 1 é possível observar estes quatro passos necessários para reprodução de uma fonte mídia.

Quadro 1 – Aplicativo para reprodução de vídeo localizado por uma URL

```

/*Arquivo - 1: RicardoPlayer.java - Criado em 03 de junho de 2004, 01:49 - Madrugada :) -
@author Ricardo A. de M. Valentim */

1. import java.awt.*; import java.awt.event.*; import java.net.URL; import javax.swing.*;
import javax.media.*; import java.io.*;

2. public class RicardoPlayer extends JComponent {
3.     public static void main(String[] args) throws Exception{
4.         final JFrame f = new JFrame("RicardoPlayer");
5.         JFileChooser fileChooser = new JFileChooser();
6.         fileChooser.showOpenDialog(f);
7.         File arquivo = fileChooser.getSelectedFile();
8.         f.addNotify( );
9.         f.setLocation(100, 100);
10.        f.addWindowListener(new WindowAdapter( ) {
11.            public void windowClosing(WindowEvent we) { System.exit(0); }
12.        });
13.        URL url = arquivo.toURL(); //1º Passo: Especificar origem do vídeo
14.        final Player player = Manager.createPlayer(new MediaLocator(url.toString()));
15.        // 2º Passo: Criar um Player para vídeo
16.        player.realize();
17.        player.addControllerListener(new ControllerListener( ) {
18.            // 3º Passo: Determinar a mídia saída e os controles de Player
19.            public void controllerUpdate(ControllerEvent ce) {
20.                if (ce instanceof RealizeCompleteEvent) {
21.                    Component c = player.getVisualComponent( );
22.                    Container content = f.getContentPane( );
23.                    content.setLayout(new BorderLayout( ));
24.                    content.add(c, BorderLayout.CENTER);
25.                    Insets i = f.getInsets( );
26.                    Dimension d = c.getSize( );
27.                    f.setSize(d.width + i.left + i.right, d.height + i.top + i.bottom);
28.                    f.setVisible(true);
29.                    player.start( ); // 4º Passo: Exibir a mídia e os controles
30.                }
31.            }
32.        });
33.    }

```

Neste código descrito no Quadro 1, existe uma série de outras preocupações, mas será comentado apenas o que efetivamente é relevante para a execução de uma aplicação que venha reproduzir uma mídia.

No 1º passo é criado um objeto do tipo URL para conter o endereço do arquivo no qual se deseja para a reprodução da mídia. No 2º passo é criado um *Player* através do método "*createPlayer()*" da classe *Manager*. O 3º passo, é efetivamente adicionar ao *Player* um objeto de controle (ouvidor) que é obtido através do método "*addControllerListene()*", onde recebe como parâmetro um objeto do tipo "*ConttrollerListene()*", este objeto de controle é também responsável pela saída de mídia através da obtenção do componente visual para exibição. Por fim, o 4º passo é realizado com a chamada do método *start()*.

A API JMF tem muitas classes e métodos que podem tratar vários aspectos sobre mídia digital, mais um ponto substancial desta API é a possibilidade de transmitir as mídias em *real-time*. Para tanto, a JMF utiliza um protocolo específico, o RTP. Este protocolo é suportado pela API, permitindo a construção de servidores de stream media, tanto para arquivos, como para transmissão de mídias capturadas a partir de dispositivos.

2.8 RTP – REAL-TIME TRANSPORT PROTOCOL

O RTP ou Protocolo de Transporte em Tempo Real (Real-Time Transport Protocol) foi apresentado formalmente em janeiro de 1996 pelo Grupo de trabalho de Redes (Networkig Working Group) do IETF (Internet Engineering Task Force) com objetivo de fornecer uma padronização de funcionalidades para os aplicativos de transmissão de dados em tempo-real como vídeo, áudio, tanto em redes unicast como nas multicast, sem, entretanto garantir a qualidade de serviço QoS ou reservar recursos de endereçamento. (MEDEIROS, 2003, p.2).

Segundo Machado (2000, p.32), "A API Java Media Framework proporciona o desenvolvimento de aplicações multimídia que apresentem conteúdos vindos de uma rede e enviem conteúdos multimídia pela rede para outros receptores utilizando o protocolo RTP."

O protocolo RTP não é orientado a conexão, pois o mesmo necessita apenas de serviço de transporte. Para efetivar uma comunicação fim a fim este protocolo roda sobre uma camada UDP/IP, a sua arquitetura pode ser observada na Figura 10.

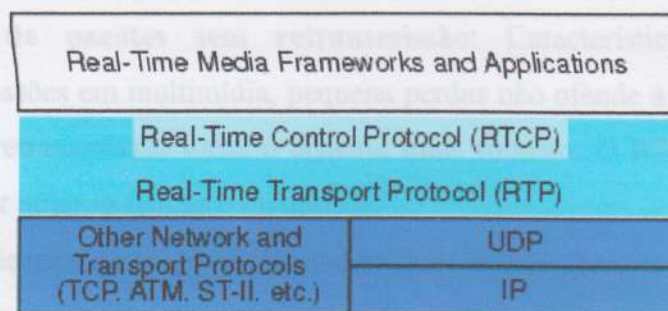


Figura 10 - Arquitetura do protocolo RTP, (Fonte: SUN, 2004).

Segundo Medeiros (2003, p.2-3):

O RTP roda sobre a camada UDP/IP utilizando os serviços de multiplexação e checksum do UDP estabelecendo uma comunicação fim a fim. As porções de áudio e vídeo produzidas pelo aplicativo remetente são encapsuladas em pacotes RTP que por sua vez são encapsulados em um segmento UDP. Entretanto, apesar de utilizar o UDP e o IP, o RTP pode ser implementado em outros ambientes já que necessita apenas de serviços de transporte não orientado à conexão.

O protocolo RTP norteia-se nas especificações de requisitos de tempo e conteúdo pertinentes à transmissão de multimídia, tanto no envio quanto da recepção. Com base em Medeiros (2003, p.3), ele utiliza os seguintes critérios:

- Numeração seqüenciada;
- Selo de temporização (stampilho);
- Envio de pacotes sem retransmissão;
- Identificação de origem;
- Identificação de conteúdo;
- Sincronismo.

De forma mais específica, com base em Medeiros (2003, p.3), estes critérios tem a seguinte semântica:

- **Numeração seqüenciada:** o RTP atribui número de ordem aos pacotes. Isto pode ser usado para verificação das perdas, seqüenciamento e possível redirecionamento;
- **Selo de temporização (stampilho):** Possibilita a temporização dos pacotes contendo áudio e/ou vídeo;

- **Envio de pacotes sem retransmissão:** Característica fundamental das transmissões em multimídia, pequena perdas não ofende a qualidade do envio e a não retransmissão torna o sistema mais robusto. O RTP apenas permite ao receptor notar as perdas e ou atrasos;
- **Identificação de origem:** Necessário para indicar quem enviou o pacote. Numa conferência *multicast*, um mesmo fluxo pode ter várias origens;
- **Identificação de conteúdo:** Permite a alteração dinâmica dos vocoders em redes sem garantia de QoS em função das perdas e do atraso com objetivo de melhorar a qualidade acústica;
- **Sincronismo:** Pacotes de uma mesma corrente podem sofrer diferentes atrasos. A variação deste atraso é prejudicial à reprodução da mídia. *Buffers* adicionais podem então ser utilizados para eliminar a diferença entre os atrasos (*jitter*). Esses mecanismos processam as informações de tempo de cada pacote.

O protocolo RTP utiliza vários recursos para garantir que os aplicativos multimídia possam de fato ser transmitidos aos seus destinos, esteja este em uma rede *unicast* ou *multicast*, tenha este vídeo ou áudio, ou os dois concomitantemente.

Essa possibilidade de transmissão de áudio e vídeo é permitida através do tratamento que o protocolo RTP provê, pois este permite uma renderização entre o áudio e vídeo, de tal forma que os quadros do vídeo entram em sincronia com o áudio. Desta forma, permitindo uma inteligibilidade entre os dois formatos apresentados ao usuário.

Na Figura 11 é possível verificar o cabeçalho do protocolo RTP e entender como os pacotes multimídia são enviados pela rede.

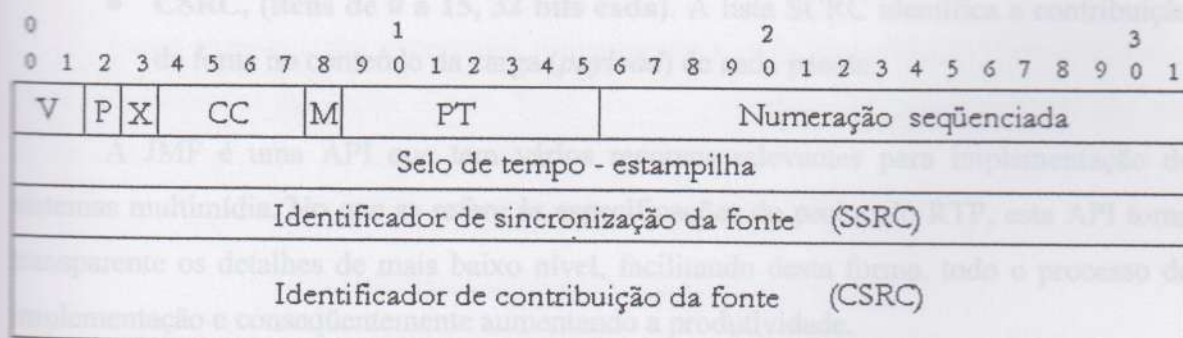


Figura 11 - Cabeçalho do protocolo RTP

Com base em Medeiros (2003, p.4), o cabeçalho pode ser especificado da seguinte forma:

- **P – Preenchimento/padding, (1 bit).** Sinaliza a adição de octetos de enchimento adicionais ao conteúdo da carga (*payload*) sem fazer parte da mesma.
- **X – Extensão/extension, (1 bit).** Com esse bit marcado, é acrescentada uma extensão ao cabeçalho original.
- **CC – Contador CSRC/CSRC count, (4 bits).** Este campo contém o número de identificadores CSRC.
- **M – marcador/marker, (1 bit).** Usado para identificar as fronteiras de um quadro numa corrente de pacotes.
- **PT – Tipo de carga/payload type, (7 bits).** Este campo identifica o formato da carga do pacote RTP como também a determinação de sua interpretação pela aplicação.
- **Numeração seqüenciada/sequence number, (16 bits) –** A numeração seqüenciada põe em ordem os diversos pacotes de RTP. A cada novo pacote, a numeração é incrementada de uma unidade.
- **Selo de temporização/timestamp, (16 bits).** Esse campo reflete o instante de amostragem do primeiro octeto no pacote RTP.
- **SSRC, (32 bits).** Esse campo identifica a fonte de sincronismo. Esta identificação foi escolhida aleatoriamente tencionando-se que duas fontes de sincronismo com a mesma sessão RTP não teriam o mesmo identificador SSRC.
- **CSRC, (itens de 0 a 15, 32 bits cada).** A lista CSRC identifica a contribuição da fonte no conteúdo da carga (*payload*) de cada pacote.

A JMF é uma API que tem vários recursos relevantes para implementação de sistemas multimídia. No que se refere às especificações do protocolo RTP, esta API torna transparente os detalhes de mais baixo nível, facilitando desta forma, todo o processo de implementação e conseqüentemente aumentando a produtividade.

3 MODELAGENS, CODIFICAÇÃO E RESULTADOS

Neste capítulo são desenvolvidos os modelos orientados a objetos, norteados efetivamente pelo objeto de estudo deste trabalho. Igualmente, especificando e comentado os resultados alcançados com a modelagem. Portanto, os resultados obtidos como os modelos projetados são utilizados para fundamentar todo o estudo desenvolvido em torno da temática abordada. Neste sentido, validando de forma mais consistente o modelo, no qual optou-se por dividi-lo em duas partes:

- Modelo da aplicação cliente: este modelo trata de ilustrar a análise de domínio de negócio da aplicação que é executada do lado cliente, ou seja, este modelo especifica os relacionamentos entre as classes que compõe a parte da aplicação que será utilizada pelo usuário;
- Modelo da aplicação servidor: Este modelo trata de ilustrar a análise de domínio de negócio da aplicação responsável pela transmissão dos fluxos de vídeo. Neste sentido são abordadas as classes que compõe este modelo, bem como, as suas interações e outros artefatos da modelagem que se fazem necessários no processo de análise.

Em cada um dos modelos é observado não apenas o design, mas também trechos relevantes da codificação que são substanciais para o entendimento do funcionamento do artefato, assim representando a validação efetiva do mesmo. Neste íterim, são citados os resultados alcançados com os modelos propostos e suas respectivas implementações.

3.1 MODELAGEM DA APLICAÇÃO CLIENTE

Neste modelo serão observadas primeiramente as funcionalidades disponibilizadas para os usuários na aplicação cliente. Desta forma, o artefato utilizado é caso de uso que exhibe as funcionalidades do sistema.

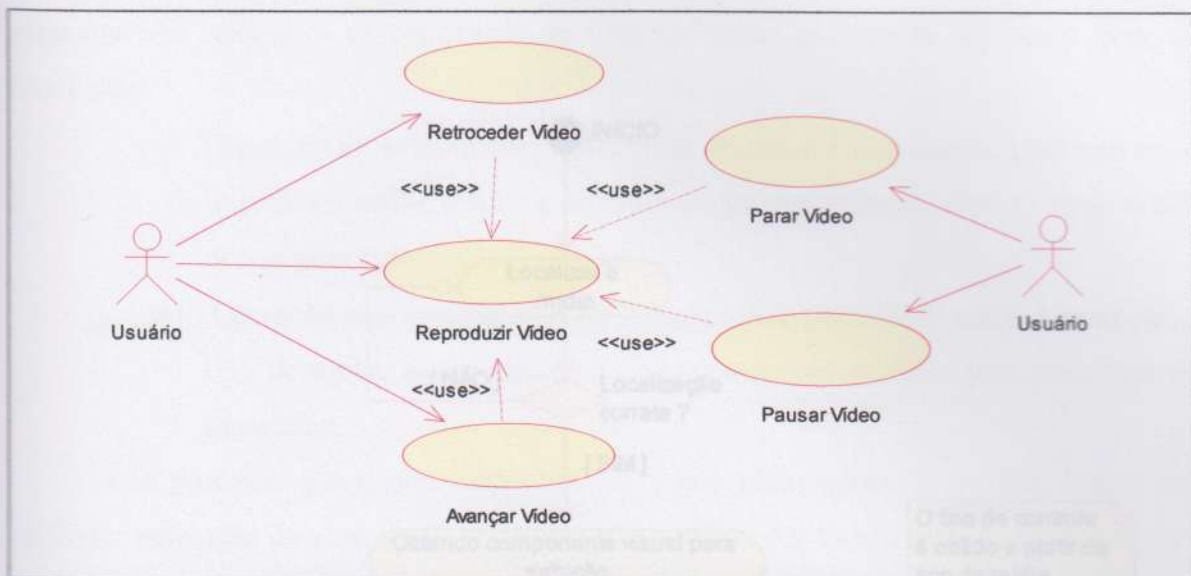


Figura 12 - Diagrama de caso de uso da aplicação cliente

Na Figura 12 é possível observar um caso de uso (Reproduzir vídeo) que é utilizado pelos demais casos de uso, desta forma criando uma forte dependência. De fato, o que ocorre é que esta dependência é imposta pelo tipo de mídia, uma vez que o tipo pode determinar quais recursos estarão disponíveis para o usuário, por exemplo, em uma mídia ao vivo só é possível reproduzir o vídeo e nunca retrocedê-lo ou avançá-lo.

Machado (2001, p. 9), afirma que quando o conteúdo é gerado ao vivo: “O usuário recebe o *stream* de forma passiva. Durante uma transmissão o cliente recebe a informação, mas não pode controlar. Por exemplo, ele não pode avançar ou voltar o conteúdo que recebe”.

A API JMF traz uma contribuição relevante neste sentido, pois a mesma permite identificar o tipo de mídia em tempo de execução de forma totalmente transparente, desta forma, otimizando eficientemente o trabalho dos desenvolvedores que não se preocuparam em criar sistemas que tenham que fazer este tipo de verificação. A Figura 13 é um diagrama de atividades que ilustra bem o fluxo que o caso de uso pode gerar.

Figura 13 Fluxo de atividades na reprodução de vídeo

O diagrama de atividades ilustra o fluxo no qual configura uma aplicação de reprodução de mídias, mostrando as operações realizadas. De forma expressiva, existem

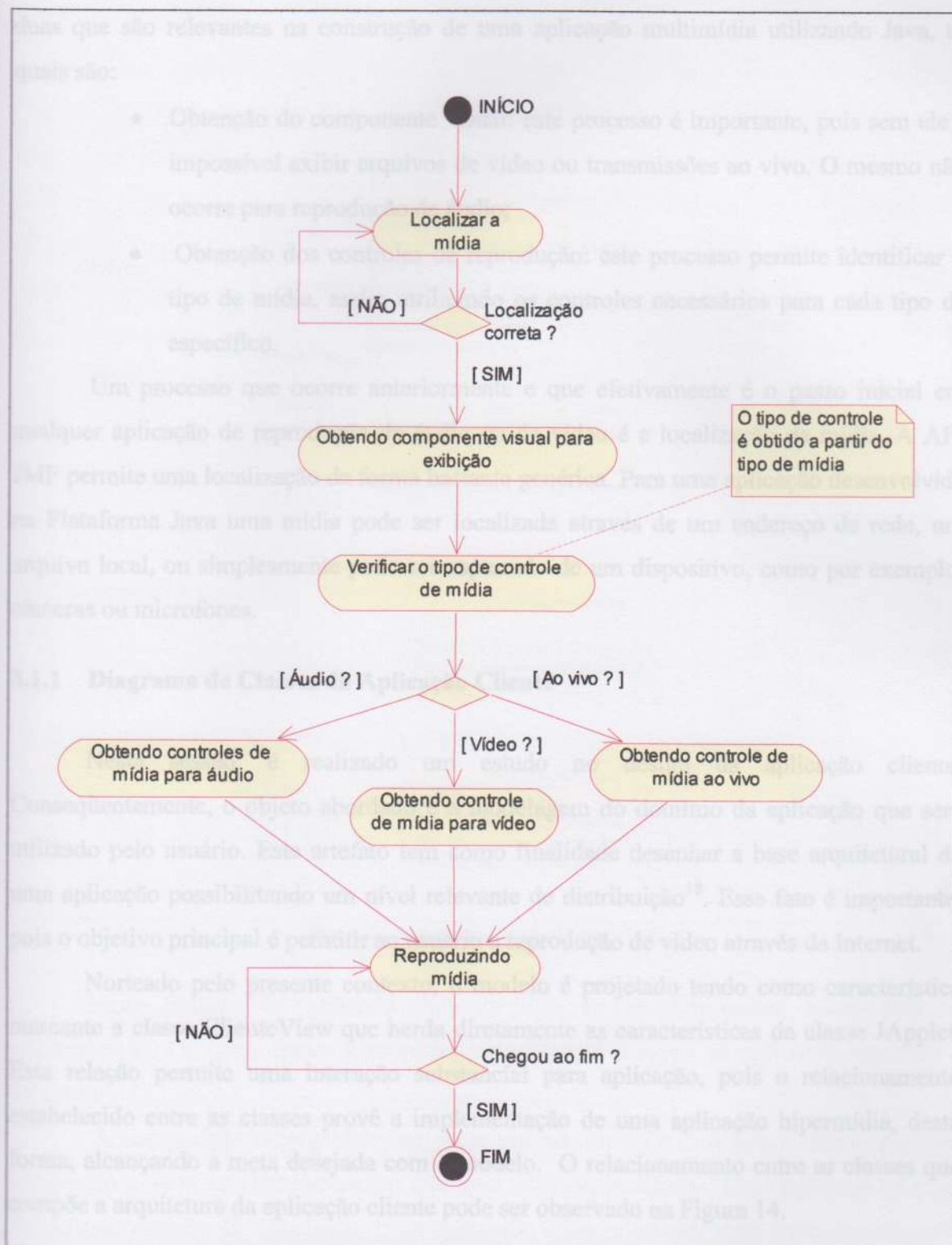


Figura 13 Fluxo de atividades na reprodução de vídeo

O diagrama de atividades ilustra o fluxo no qual configura uma aplicação de reprodução de mídias, mostrando as operações realizadas. De forma expressiva, existem

duas que são relevantes na construção de uma aplicação multimídia utilizando Java, as quais são:

- Obtenção do componente visual: este processo é importante, pois sem ele é impossível exibir arquivos de vídeo ou transmissões ao vivo. O mesmo não ocorre para reprodução de áudio;
- Obtenção dos controles de reprodução: este processo permite identificar o tipo de mídia, assim atribuindo os controles necessários para cada tipo de específico.

Um processo que ocorre anteriormente e que efetivamente é o passo inicial em qualquer aplicação de reprodução de áudio ou de vídeo é a localização da mídia. A API JMF permite uma localização de forma bastante genérica. Para uma aplicação desenvolvida na Plataforma Java uma mídia pode ser localizada através de um endereço de rede, um arquivo local, ou simplesmente pode ser capturado de um dispositivo, como por exemplo, câmeras ou microfones.

3.1.1 Diagrama de Classes da Aplicação Cliente

Nesta sessão, é realizado um estudo no design da aplicação cliente. Conseqüentemente, o objeto abordado é a modelagem do domínio da aplicação que será utilizado pelo usuário. Este artefato tem como finalidade desenhar a base arquitetural de uma aplicação possibilitando um nível relevante de distribuição¹⁸. Esse fato é importante, pois o objetivo principal é permitir ao usuário a reprodução de vídeo através da Internet.

Norteadado pelo presente contexto, o modelo é projetado tendo como característica marcante a classe *ClienteView* que herda diretamente as características da classe *JApplet*. Esta relação permite uma interação substancial para aplicação, pois o relacionamento estabelecido entre as classes provê a implementação de uma aplicação hipermídia, desta forma, alcançando a meta desejada com o modelo. O relacionamento entre as classes que compõe a arquitetura da aplicação cliente pode ser observado na Figura 14.

¹⁸ A distribuição aqui citada são as providas por aplicações baseadas em Applet Java.

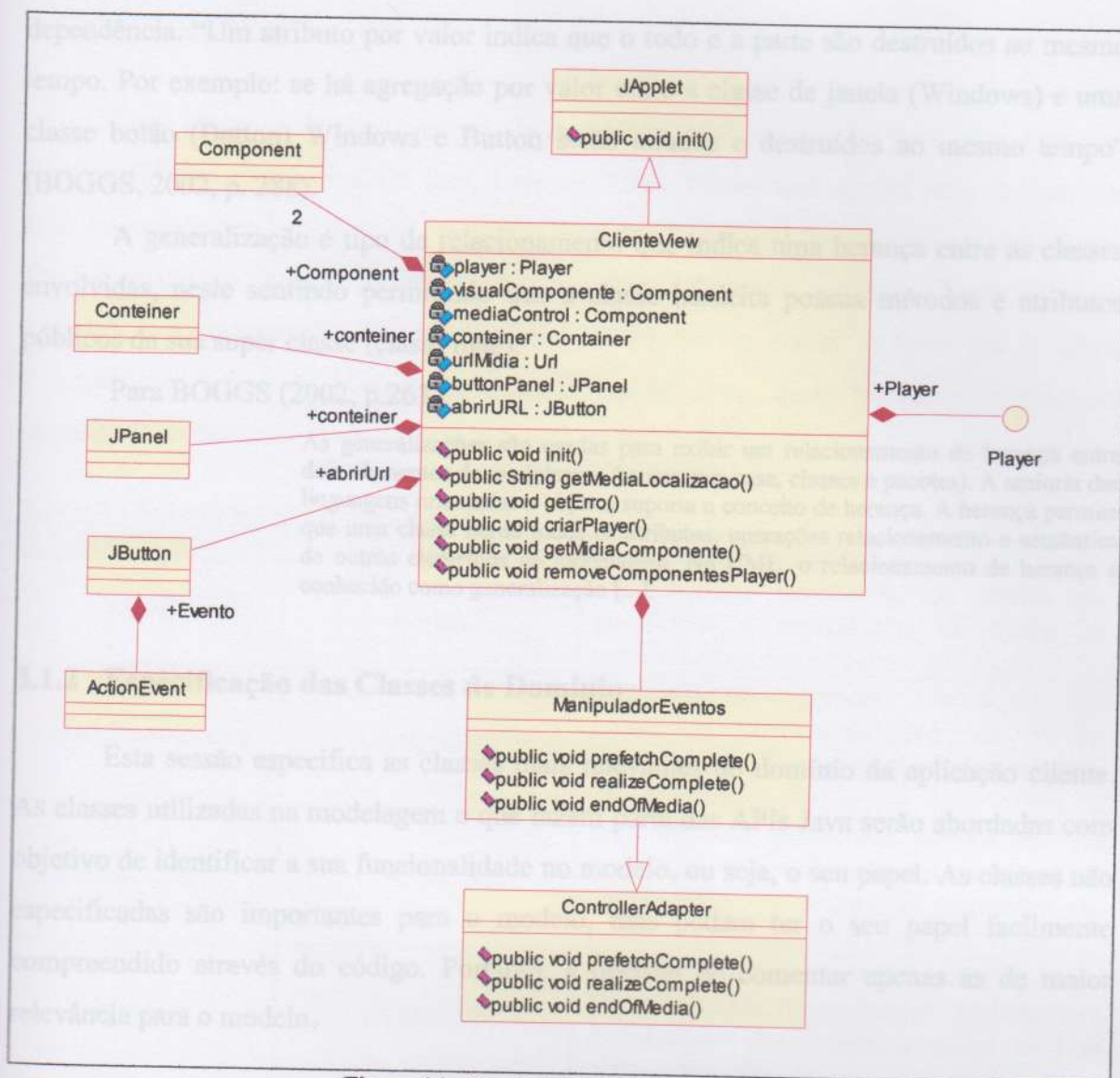


Figura 14 - Domínio da aplicação cliente

Neste modelo foi aplicado dois de tipo de relacionamento: composição e generalização. Os quais foram necessários para mapear o cerne da aplicação hipermídia desejada.

A composição (agregação com passagem por valor) indica que a classe que está compondo a outra no relacionamento tem uma forte dependência. “Uma agregação é uma forma mais concisa de associação. Uma agregação é um relacionamento entre um conjunto e suas partes” (BOGGS, 2002, p. 276). Um outro aspecto importante na composição é que a parte (objeto que compõe) só existirá se o todo existir (objeto composto), consequentemente ambos serão criados e destruídos simultaneamente, devido ao tipo de

dependência. “Um atributo por valor indica que o todo e a parte são destruídos ao mesmo tempo. Por exemplo: se há agregação por valor entre a classe de janela (Windows) e uma classe botão (Button) Windows e Button serão criados e destruídos ao mesmo tempo” (BOGGS, 2002, p. 288).

A generalização é tipo de relacionamento que indica uma herança entre as classes envolvidas, neste sentido permitindo que a classe herdeira possua métodos e atributos públicos da sua super classe (classe mãe).

Para BOGGS (2002, p.263):

As generalizações são usadas para exibir um relacionamento de herança entre dois elementos de modelagem (atores, use case, classes e pacotes). A maioria das linguagens orientadas a objetos suporta o conceito de herança. A herança permite que uma classe herde todos os atributos, operações relacionamento e semântica de outros elementos de modelagem. Na UML, o relacionamento de herança é conhecido como generalização [...].

3.1.2 Especificação das Classes de Domínio

Esta sessão especifica as classes mais relevantes do domínio da aplicação cliente. As classes utilizadas na modelagem e que fazem parte das APIs Java serão abordadas com objetivo de identificar a sua funcionalidade no modelo, ou seja, o seu papel. As classes não especificadas são importantes para o modelo, mas podem ter o seu papel facilmente compreendido através do código. Portanto, a decisão de comentar apenas as de maior relevância para o modelo.

3.1.2.1 JApplet

A classe JApplet tem um papel relevante neste modelo, pois é a partir do relacionamento (generalização) estabelecido entre a classe ClienteView e a JApplet que é possível tornar a aplicação disponível em um *browser* da *web*.

Em um *Applet* o construtor é utilizado apenas para a sua inicialização. Desta forma, o método `public void init()` é invocado após o JApplet ser criado com o objetivo de inicializar as configurações básicas.

Para Niemeyer e Knudsen (2000, p. 599): “[...] Um applet normalmente não realiza trabalho algum em seu construtor; ele se baseia no construtor *default* para a classe JApplet e realiza sua inicialização no método `init()`”.

3.1.2.2 ClienteView

Esta classe é o foco central da aplicação, pois nesta, se concentra todas as funcionalidades necessárias. De fato, a classe ClienteView é um Applet Java que permite criar instancias que podem executar vídeos no lado cliente, seja este ou não em tempo real. De forma efetiva, a herança implementada no modelo torna a classe ClienteView um JApplet. É fundamentada nas características herdadas da classe JApplet que a classe ClienteView facilmente poderá ser incluída em *browser*, desta forma, atendendo aos requisitos de distributividade da aplicação.

Tabela 3 – Especificação da classe ClienteView

Atributos	Tipo	Descrição
player	Player	Este atributo representa um objeto que possibilita a reprodução da mídia de acordo com a sua origem específica.
visualComponente	Component	Este atributo representa um objeto que possibilita a visualização da mídia. Efetivamente é uma espécie de contêiner para a visualização.
mediaControl	Component	Este atributo representa um objeto que possibilita a visualização dos componentes de controle. Os objetos de controles são mutáveis, a depender da fonte de mídia.
container	Container	Este atributo atua como contêiner dos objetos visuais. Neste sentido, facilitando a distribuição dos mesmos sobre a área visível da aplicação.
urlMidia	Url	Este atributo representa um objeto que guardará e/ou receberá a localização da fonte de mídia.
buttonPanel	JPanel	Este atributo representa um objeto para adição de outros objetos. Nesta aplicação, sua funcionalidade é servir de container para os objetos do tipo JButton.
abrirURL	JButton	Este atributo representa um botão. Este objeto é um componente visual na aplicação que efetivamente é utilizado para criar uma associação entre este objeto e o evento que possibilita a escolha de uma URL para localização de mídia.

Métodos		
public void init()	Retorno: void	Este método é herdado da classe JApplet, portanto, nesta classe o método é submetido a uma sobre escrita. Este método é utilizado para inicializar a funcionalidades básicas da classe.
	Parâmetro: -	
public String getMidiaLocalizacao()	Retorno: String	Este método invoca uma das caixas de diálogo utilizada na classe JOptionPane para entrada de uma String que representa a URL da fonte de mídia.
	Parâmetro: -	
public void getErro(String erro)	Retorno: void	Este método é invocado quando é disparada alguma exceção mapeada pela aplicação.
	Parâmetro: String	
public void criarPlayer(String localizacaoMidia)	Retorno: void	Este método cria a partir da URL informada um player para aplicação. A criação deste player irá ocorrer por referência.
	Parâmetro: String	
public void getMidiaComponente()	Retorno: void	Este método cria um componente visual para exibição da mídia e também os objetos de controle.
	Parâmetro: -	
public void removeComponentesPlayer()	Retorno: void	Este método libera o Player para os recursos do sistema e restaura a mídia.
	Parâmetro: -	

3.1.2.3 ManipuladorEventos

A JMF provê uma forma de observar os estados assumidos por um player através de uma classe com perfil de monitor. No modelo foi utilizada a classe ControllerAdapter para estabelecer um relacionamento de generalização sobre a classe ManipuladorEventos. Com este relacionamento a classe ManipuladorEventos assume por herança o comportamento da sua super classe. A partir da herança estabelecida foi criada uma espécie de polimorfismo nos métodos citados na classe ManipuladorEventos com o objetivo de adaptar o comportamento do objeto as necessidades exigidas pela aplicação. De forma mais consistente é possível observar a funcionalidade polimórfica deste modelo na codificação.

Segundo a SUN (2004):

Baseado no tempo existe dois estados primários: Parado (*Stopped*) e Iniciado (*Started*). Para facilitar o gerenciamento dos recursos é possível controlar além dos estados primários mais cinco estados: não realizado (*Unrealized*), verificando recursos (*Realizing*), recursos verificados (*Realized*), obtendo recursos (*Prefetching*), e recursos obtidos (*Prefetched*).

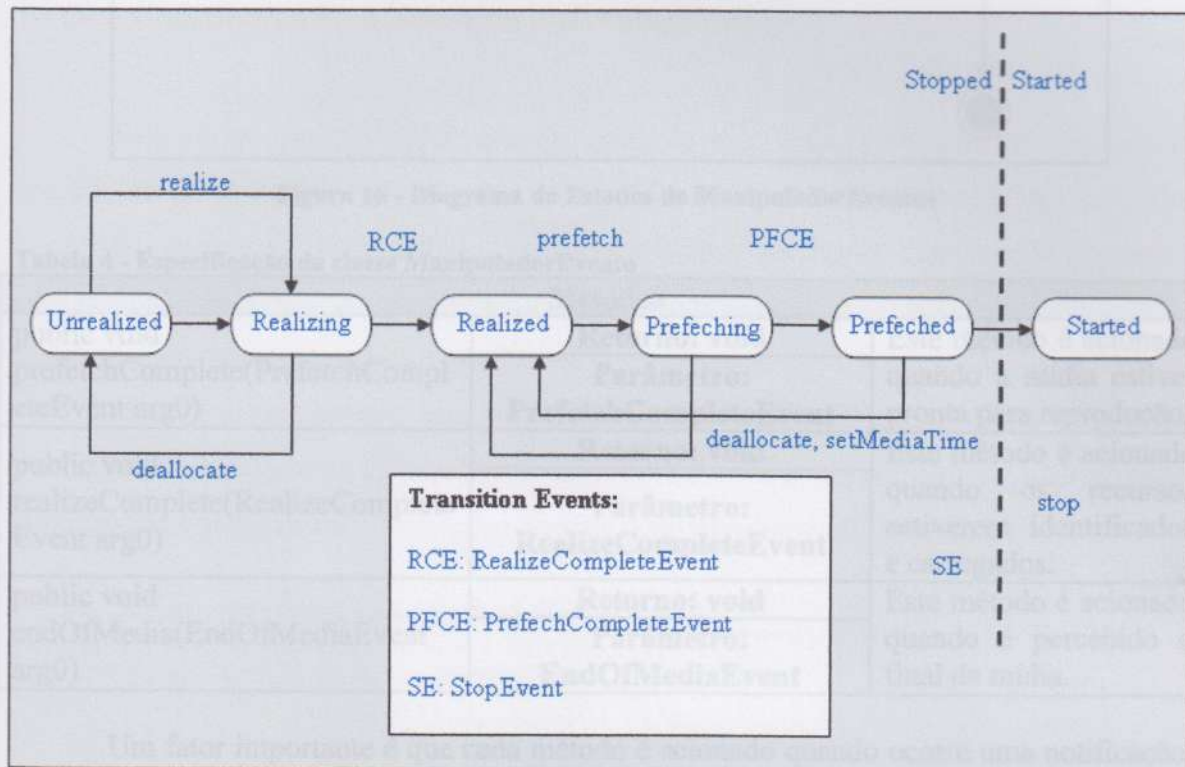


Figura 15 - Estados de um Player, (Fonte: SUN, 2004)

Através da modelagem da classe ManipuladorEvento é possível interagir em três destes estados citados pela SUN(2004), os quais podem ser observados no diagrama de estados do modelo. Esta interação entre as mudanças de estado tem por objetivo observar determinados eventos que possibilitaram um melhor tratamento na reprodução do vídeo.

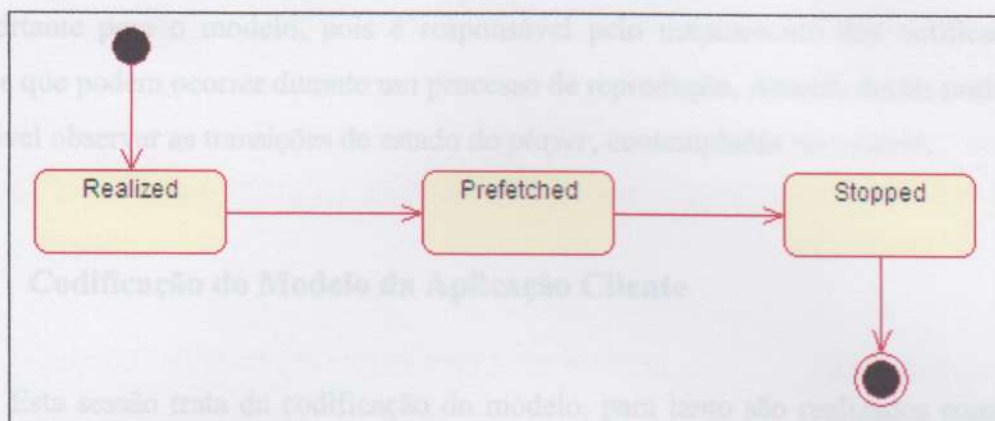


Figura 16 - Diagrama de Estados de ManipuladorEventos

Tabela 4 - Especificação da classe ManipuladorEvento

Métodos		
public void prefetchComplete(PrefetchCompleteEvent arg0)	Retorno: void Parâmetro: PrefetchCompleteEvent	Este método é acionado quando a mídia estiver pronta para reprodução.
public void realizeComplete(RealizeCompleteEvent arg0)	Retorno: void Parâmetro: RealizeCompleteEvent	
public void endOfMedia(EndOfMediaEvent arg0)	Retorno: void Parâmetro: EndOfMediaEvent	Este método é acionado quando é percebido o final da mídia.

Um fator importante é que cada método é acionado quando ocorre uma notificação, o que significa uma mudança de estado do player, mapeado no diagrama de estado (Figura 16).

3.1.2.1 ControlleAdapter

Esta classe é uma espécie de adaptadora que implementa métodos que podem mapear uma série de eventos que estão relacionados a alguma ação. Estas ações podem ser disparadas durante algum processo de *listener* (ouvidor). Classes que herdam ControlleAdapter podem facilmente sobre escrever os seus métodos, desta forma, permitindo realizar determinadas ações durante a percepção de alguma ação mapeada. Há exemplo, pode ser citado o método **endOfMidia()** que é disparado quando uma reprodução chega ao fim, uma vez notificado o final da reprodução podem ser realizadas operações para tratar este processo de finalização de forma mais otimizada. Neste sentido, esta classe

é importante para o modelo, pois é responsável pelo mapeamento das notificações de eventos que podem ocorrer durante um processo de reprodução. Através destas notificações é possível observar as transições de estado do *player*, contempladas no modelo.

3.1.3 Codificação do Modelo da Aplicação Cliente

Esta sessão trata da codificação do modelo, para tanto são realizados comentários dos trechos do código a fim de detalhar os objetivos pretendidos e os resultados alcançados com esta codificação. Os resultados são demonstrados através de figuras que ilustram o código em tempo de execução, bem como, uma descrição textual das ocorrências disparadas no momento da execução.

Para cada quadro que cita os trechos de código da aplicação, segue abaixo um comentário com objetivos alcançados ou com uma descrição da codificação. Desta forma, detalhando os procedimentos adotados durante o desenvolvimento da aplicação.

Quadro 2 - Aplicação cliente: Importando classes

```
import javax.media.ControllerAdapter;  
import javax.media.EndOfMediaEvent;  
import javax.media.Manager;  
import javax.media.MediaLocator;  
import javax.media.NoPlayerException;  
import javax.media.Player;  
import javax.media.PrefetchCompleteEvent;  
import javax.media.RealizeCompleteEvent;  
import javax.media.Time;  
import javax.swing.JButton;  
import javax.swing.JFileChooser;  
import javax.swing.JOptionPane;  
import javax.swing.JPanel;  
import java.awt.Component;  
import java.awt.Container;  
import java.awt.BorderLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.File;  
import java.net.URL;  
import java.io.IOException;  
import java.net.MalformedURLException;  
import java.util.Timer;
```

No trecho de código do Quadro 2 são importadas várias classes que subsidiaram o contexto da aplicação. Neste ponto podem ser observadas as classes da API JMF, bem como, da APIs SWING, AWT. As classes da API JMF são utilizadas como o núcleo da aplicação, e as classes das APIs AWT e SWING são utilizadas na construção da parte visual do sistema.

Quadro 3 - Aplicação cliente: declaração da classe e seus atributos

```
// Declaração da classe e extensão da classe JApplet

public class ClienteView extends javax.swing.JApplet {

private Player player; // Declaração do objeto player.

private Component visualMedia; // Componente para conteúdo visual.

private Component mediaControl; // Componente para conteúdo de controle visual.

private Container container; // Container principal.

private URL arquivoURL; // Objeto que irá conter uma URL que aponta para o
//arquivo de mídia;
```

No Quadro 3 é descrita a declaração da classe ClienteView demonstrado que a mesma herda da classe JApplet. Este herança indica para a JVM (*Java Virtual Machine* – Máquina Virtual Java) que a classe ClienteView assume propriedades de um JApplet que pode ser executada em *browser web* ou em AppletView. Neste quadro também é ilustrado os atributos citados no modelo, desta forma, codificando o relacionamento do tipo agregação com passagem por valor, ou seja, uma composição.

Quadro 4 - Aplicação cliente: método de inicialização

```
//Este método inicializa a JApplet implementando uma sobre escrita.
public void init() {

    this.container = this.getContentPane(); // Obtendo um container.

    JPanel buttonPanel = new JPanel(); // Instanciando um JPanel para servir
    // de container dos botões.

    /*
    * Adicionado o objeto buttonPanel ao container principal e movendo o
    * mesmo para a parte superior através do BorderLayout.NORTH
    */
    this.container.add(buttonPanel, BorderLayout.NORTH);
```

```

// Instanciando e atribuindo um botão o seu objeto container.
// Botão que irá acionar um evento que permitirá localizar uma URL.
JButton abrirURL = new JButton("Abrir Conexão");
buttonPanel.add(abrirURL);

// Associando um evento ao botão abrir URL.
abrirURL.addActionListener(

new ActionListener() {

    public void actionPerformed(ActionEvent event) {

//Solicita que seja informado a localização da mídia
        String nomeEndereco = getMediaLocalizacao();
        if (nomeEndereco != null) {

            criarPlayer(nomeEndereco); // Cria um player.

        }

    }

});

// Ativando uma geração leve dos player para permitir
// uma melhor compatibilidade com os componentes de Interface.

Manager.setHint(Manager.LIGHTWEIGHT_RENDERER, Boolean.TRUE);

this.setSize(300, 300); // Definindo o tamanho do JApplet
this.setVisible(true); // Tornando o JApplet visível

} // Final do init() do JApplet (ClienteView)

```

No Quadro 4, o método **public void init ()** é utilizado para inicializar o objeto JApplet da aplicação, desta forma, instanciando os componentes básicos para o seu funcionamento, bem como, realizando as operações fundamentais que configuram a sua inicialização.

No método de inicialização existem duas operações relevantes:

- Atribuição de um *listener* (ouvidor) ao objeto JButton (abrirURL) que será responsável por identificar o evento de clicar no botão, sendo efetivamente acionado pelo usuário, a fim de localizar uma mídia de vídeo;
- O segundo passo é a preparação do componente Swing para a reprodução de vídeo. Isso é realizado através do método `setHint ()` da classe *Manager*. Foram realizados testes sem utilizar este procedimento, no entanto, foi gerada uma

exceção, informando que o JApplet não tinha um componente visual adequado a exibição de vídeo;

Quadro 5 - Aplicação cliente: método de retorno da mídia

```
/**
 * @return String Este método retorna o endereço da mídia informado pelo
 * usuário
 */
public String getMediaLocalizacao() {

    /*Faz uma chamada a uma caixa de entrada para que o usuário possa informar a
    URL*/
    String entrada = JOptionPane.showInputDialog(this, "Informe a URL");

    if (entrada != null && entrada.length() == 0) { // Testa se o Usuário
// apertou Ok sem digitar
// nada.

        return null;
    } else {

        return entrada; // Retorna a localização da mídia
    }
}
```

O Quadro 5, exibe o método de retorno da localização da mídia, mapeado junto ao evento do botão abrirURL. O mesmo quando acionado faz uma chamada a este método passando uma String solicitando do usuário uma URL e logo após a confirmação da URL pelo usuário, realiza uma validação que verifica se está ou não vazia.

Quadro 6 - Aplicação cliente: método para exibição de erro

```
public void getErro(String erro) {
    // Invoca uma caixa de dialogo para exibir uma mensagem de erro
    JOptionPane.showMessageDialog(this, erro, "Erro",
        JOptionPane.ERROR_MESSAGE);
}
```

O trecho de código exibido no Quadro 6 é um método que é invocado quando ocorre algum erro. A sua estrutura é simples, mas tem uma grande utilidade, pois pode servir de interação entre a aplicação e o usuário informando os procedimentos errados gerados pelo mesmo. Desta forma, servindo como um instrumento auxiliar da aplicação.

Quadro 7 - Aplicação cliente: criação do player

```
/**
 * @param localizacaoMidia
 *      Cria um Player a partir do endereço da Mídia
 */
public void criarPlayer(String localizacaoMidia) {

    /*
     * Esta condição verifica se existe um player, caso seja verdade este é
     * removido para que seja criado outro
     */
    if (this.player != null) {

        this.removeComponentesPlayer();

    }

    // Instanciando um objeto que representa o endereço de origem da mídia
    MediaLocator mediaLocator = new MediaLocator(localizacaoMidia);

    // Testa se o mediaLocator é nulo, caso seja é porque não foi possível
    // abrir o arquivo, ou o caminho não estava correto.
    if (mediaLocator == null) {

        // Utilizando o método getErro() para informar uma ocorrência
        this.getErro("ERRO - Não foi possível abrir o arquivo");
        return;
    }

    try{
        // Cria um player a partir de um MediaLocator
        player = Manager.createPlayer(mediaLocator);

        // Registra um Listener para tratar os eventos do player
        this.player.addControllerListener(new ManipuladorEvento());

        // Chama o método realize para permitir que o player inicie o processo de
        // reprodução
        this.player.realize();

    }catch(NoPlayerException noPlayerException){

        noPlayerException.printStackTrace();

    } catch (IOException e) {
```

```
e.printStackTrace();
}
}
```

O método do Quadro 7 é um dos mais relevantes da aplicação, pois contempla a sua funcionalidade central que é inicializar o processo de reprodução, neste método encontram-se as seguintes operações:

- Criação de um objeto do tipo `MediaLocator` que tem funcionalidade de armazenar a localização da fonte de mídia. É a partir deste objeto torna-se possível utilizar o método `createPlayer` da classe `Manager`;
- Atribuição de um ouvidor (`ManipuladorEvento()`) ao objeto `player`, este ouvidor é mapeado através de uma *Inner Class*¹⁹ (classe interna ou classe aninhada). Com esta atribuição é possível capturar as notificações geradas a cada mudança de estado no `player`.

Quadro 8 - Aplicação cliente: obtendo os componentes visuais do player

```
/**
 * Este método adiciona os componente visual e o de controle para a mídia
 * especificada.
 */
public void getMidiaComponente() {

    // Obtém um componente visual para o Player.
    visualMedia = player.getVisualComponent();

    // Adiciona o componente visual, caso esteja presente.
    if (this.visualMedia != null) {

        container.add(visualMedia, BorderLayout.CENTER);
    }

    // Obtém um componente de controle para o Player.
    mediaControl = player.getControlPanelComponent();

    // Adiciona o componente de controle, caso esteja presente.
    if (mediaControl != null) {

        container.add(mediaControl, BorderLayout.SOUTH);
    }
}
```

¹⁹ Uma *Inner Class* é uma espécie de classe aninhada.

O método descrito no Quadro 8 é responsável por realizar as operações de obtenção dos componentes visuais do player, que nesta aplicação se dividem em duas partes:

- Obtenção do componente visual, ou seja, do componente de exibição do vídeo. Esta operação é realizada através do método `getVisualComponent()`, este método retorna um objeto do tipo `Component`. O componente visual é uma espécie de recurso obrigatório para exibição de vídeo.
- Obtenção do componente de controle, estes componentes são obtidos através do método `getControlPanelComponent()` que verifica e retorna os recursos de controle adequados a cada tipo de mídia;

Os recursos visuais e de controle de um player são obtidos de forma totalmente transparente pelo desenvolvedor, pois API JMF contemplou no método de criação do player estas verificações de forma dinâmica. Desta forma, não é necessário testar qual o tipo de mídia para gerar seus componentes de exibição e controle, bastando apenas solicitá-lo, como se encontra codificado no Quadro 8. Isso ocorre porque a JMF tem os seus próprios componentes de visualização e controle para cada tipo específico de mídia.

Um exemplo que pode ser citado são os componentes de controle no caso de transmissão ao vivo, aonde não está habilitada às operações de avançar e retroceder.

Quadro 9 - Aplicação cliente: liberando o player

```
/**
 * Este método libera o Player para os recursos do sistema e restaura a
 * mídia
 */
public void removeComponentesPlayer() {

    // remove o componente de vídeo anterior, caso exista.
    if (this.visualMedia != null) {

        this.container.remove(visualMedia);

    }

    // remove os componentes de controle de vídeo anterior, caso exista.
    if (this.mediaControl != null) {

        this.container.remove(mediaControl);

    }
}
```

```

    this.player.close();// Encerra o Player
}

```

O Quadro 9 aborda um trecho do código que é responsável pela finalização efetiva de um player. Este módulo da classe *ClienteView*, trata de liberar os recursos antes de fechar o player, desta forma, tornando possível uma nova exibição, já que os recursos alocados estão disponíveis ao sistema.

O método **removeComponentesPlayer()** executa dois tipos de operações:

- Remoção dos componentes: esta operação é fundamentada pelo método **remove ()** do objeto container. Este método remove os componentes de controle e de exibição criados a partir do player;
- Fechamento do player: o método **close ()** do objeto player é responsável pelo encerramento deste, assim provendo o fechamento do arquivo de mídia ou da conexão com o fluxo de stream media. Desta forma, liberando totalmente os recursos, assim possibilitando a criação de um novo player pelo sistema.

Quadro 10 - Aplicação cliente: *Inner Class*

```

/**
 * Esta é um Inner Class que tem o papel de observar alguns eventos ocorridos *
 * no processo de reprodução do player. Desta forma, sendo possível capturar *
 * determinadas notificações atribuindo ações os eventos notificados.
 */
private class ManipuladorEvento extends ControllerAdapter {

    // Estado em que já é possível obter os componentes visuais e de controle
    public void prefetchComplete(PrefetchCompleteEvent arg0) {

        // Invocando o método responsável pela aquisição dos componentes visuais
        getMidiaComponente();

        validate();// Acionado o método validate da super classe Object
        player.start();// Iniciando o processo de reprodução
    }

    /** Estado em que os recursos já foram identificados e podem carregados pelo
    sistema*/

    public void realizeComplete(RealizeCompleteEvent arg0) {
        // Indica que os recursos foram devidamente carregados;
        player.prefetch();
    }
}

```

```
// Este método indica uma notificação de finalização no processo de reprodução
public void endOfMedia(EndOfMediaEvent arg0) {

// Voltando o tempo da mídia para o instante zero
    player.setMediaTime(new Time(0));

    player.stop(); //Parando a reprodução
}
}
} // Final da classe ClienteView
```

A *Inner Class* descrita no Quadro 10 constitui o trecho de código que implementa o “ouvidor” do player na aplicação. É a través desta classe, que se torna possível identificar algumas mudanças de estado no player, como já comentado. Cada assinatura de método nesta classe representa um sobre escrita nos métodos referentes na sua super classe, desta forma, tornando-se possível interagir em três estados diferentes:

- *Prefetching*: Este estado indica que o player esta obtendo os recursos necessários para exibição da mídia. O estado de *prefetching* é mapeado através do método **realizeComplete()**;
- *Prefetched*: Este estado indica que o player já tem conhecimento dos recursos da mídia, e que a partir deste ponto pode-se iniciar o processo de reprodução. O estado *prefetched* é mapeado no método **prefetchComplete()**;
- *Stopped*: Este estado indica que chegou ao fim da reprodução da mídia e que o processo de reprodução encontra-se parado. O estado *Stopped* é mapeado através do método **endOfMedia()**;

Cada um dos métodos implementados na *inner class* ManipuladorEvento, segue uma lógica que permite acionar operações que possibilitarão a efetiva reprodução da mídia. O Quadro 10 é fortemente indicado para o entendimento da lógica de cada método disposto na *inner class*.

Outros eventos poderiam também ser mapeados na classe ManipuladorEvento, mas para a aplicação desenvolvida estes são suficientes e atendem a contento os requisitos.

3.1.3 Resultados obtidos com a codificação do modelo da Aplicação Cliente

Os primeiros resultados obtidos foi na verificação entre o modelo proposto para aplicação cliente e sua codificação, neste ponto observou-se uma fidelidade entre ambos. Desta forma, alcançando um dos objetivos que foi o de produzir um modelo orientado a objetos que fosse capaz de representar uma aplicação hipermídia de forma consistente.

Entre a modelagem e a codificação o fator consistência pode agregar uma excelente contribuição para outros projetos, pois poderão utilizar o presente modelo como objeto de estudo para outras pesquisas, uma vez que foi possível validar toda a modelagem, desta forma, provando a sua eficiência.

Um outro fator substancial dos resultados obtidos com a aplicação cliente é o produto final, ou seja, a aplicação, pois esta foi utilizada como instrumento verificador do trabalho desenvolvido na construção do sistema hipermídia.

Nas figuras Figura 17, Figura 18 e Figura 19 é possível observar a execução da aplicação hipermídia que representa o produto final entre o modelo e a codificação.

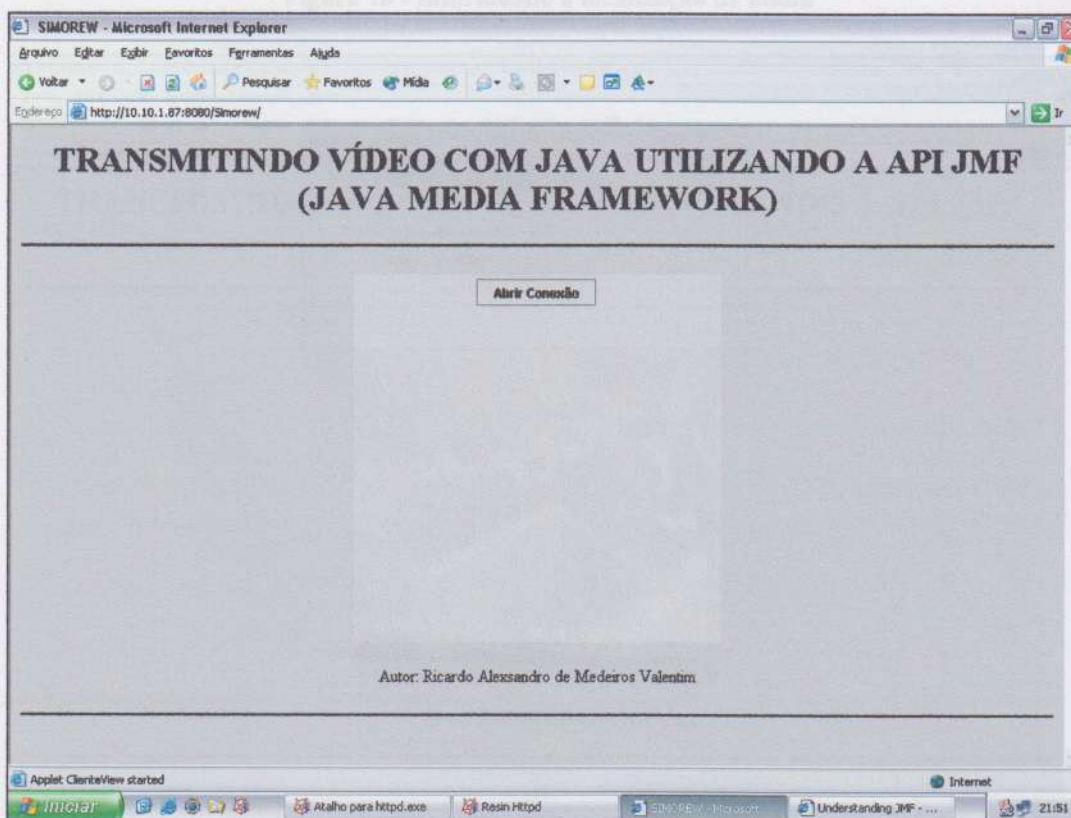


Figura 17 - Tela de entrada

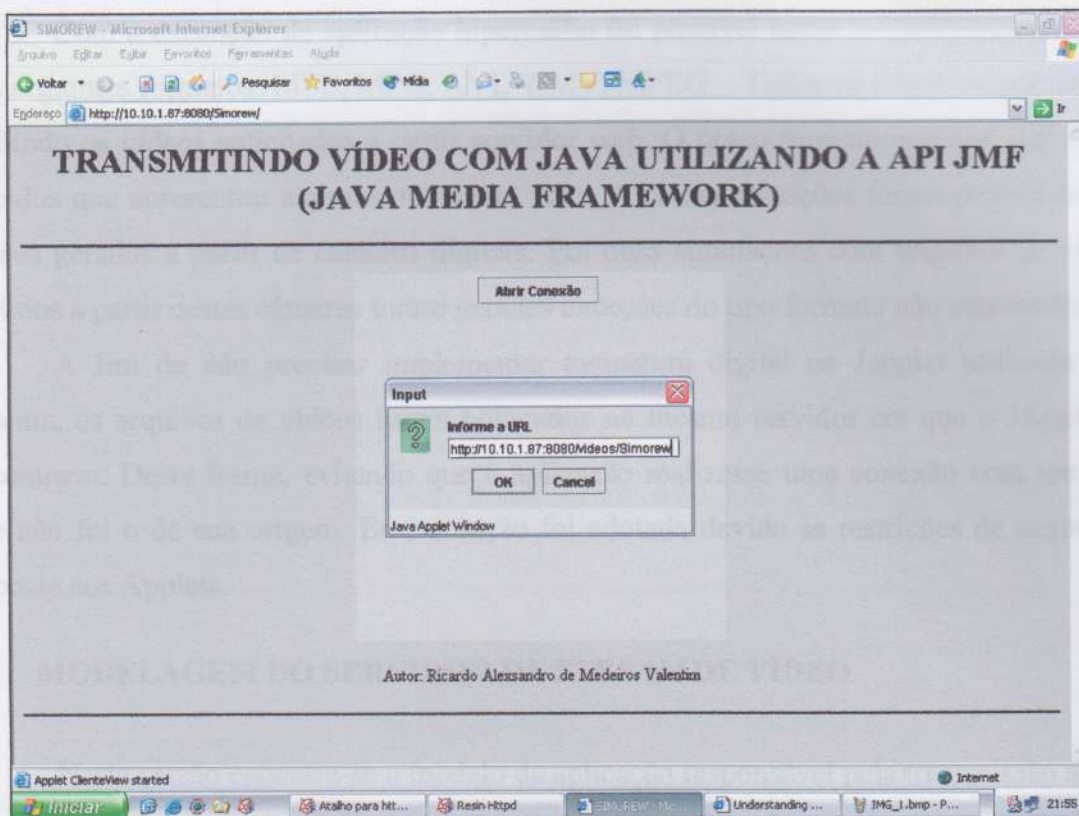


Figura 18 - Informando a localização da mídia

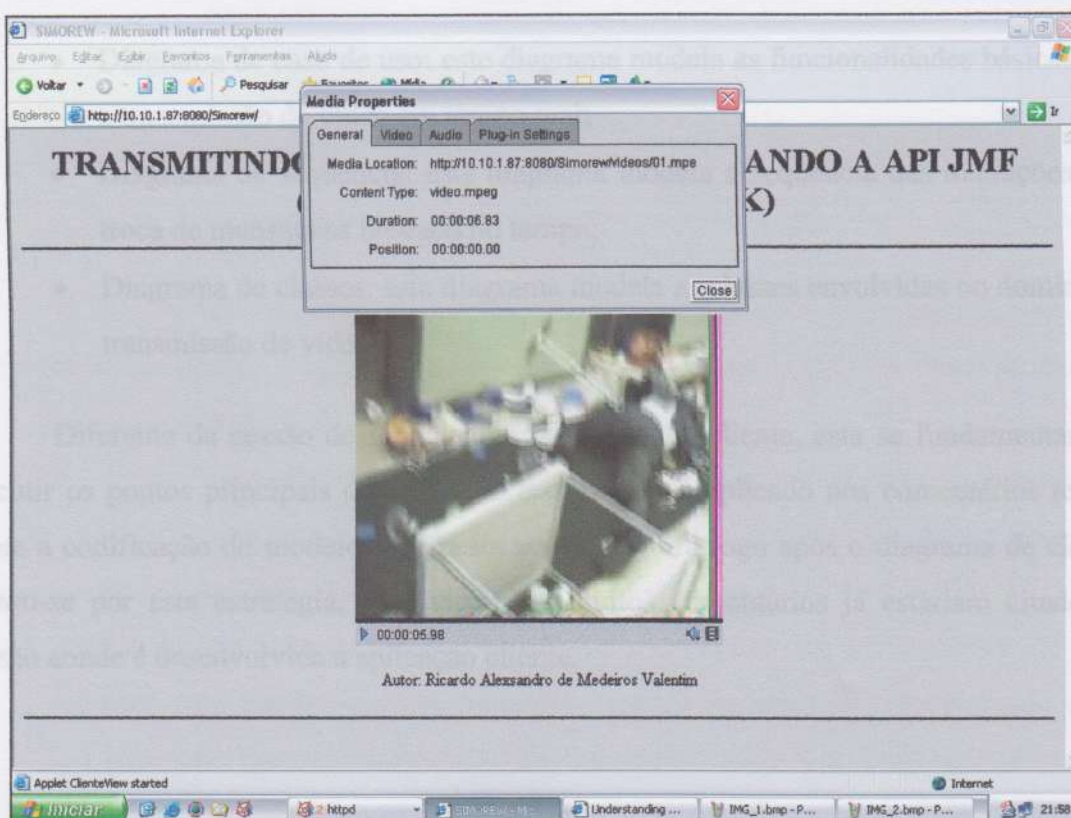


Figura 19 - Reproduzindo vídeo e exibindo um dos componentes de controle

3.2.1 Com a execução da aplicação hipermídia foi possível testar a reprodução de vídeo nos seguintes formatos: MPE, MPG, AVI, WMV, MPEG. Todos os formatos executaram exibindo os vídeos solicitados a partir servidor web. O único formato entre os que foram testados que apresentou algumas restrições foi o AVI. As restrições foram percebidas em vídeos gerados a partir de câmeras digitais. Em duas simulações com arquivos de vídeos obtidos a partir destas câmeras foram geradas exceções do tipo formato não suportado.

A fim de não precisar implementar assinatura digital na Japplet utilizada pelo sistema, os arquivos de vídeos foram colocados no mesmo servidor em que o JApplet se encontrava. Desta forma, evitando que a aplicação realizasse uma conexão com um *host* que não foi o de sua origem. Essa solução foi adotada devido às restrições de segurança imposta aos Applets.

3.2 MODELAGEM DO SERVIDOR DE *STREAM* DE VÍDEO

Nesta sessão encontra-se o modelo da aplicação responsável pela transmissão ao vivo de vídeo em tempo real. Neste modelo serão abordados três artefatos:

- Diagrama de caso de uso: este diagrama modela as funcionalidades básicas para a transmissão de vídeo em tempo real;
- Diagrama de seqüência: este diagrama modela a seqüência das interações com troca de mensagens baseada no tempo;
- Diagrama de classes: este diagrama modela as classes envolvidas no domínio da transmissão de vídeo.

Diferente da sessão de modelagem da aplicação cliente, esta se fundamentará em discutir os pontos principais da aplicação e o mesmo é aplicado nos comentários tecidos sobre a codificação do modelo, que nesta sessão seguirá logo após o diagrama de classes. Optou-se por esta estratégia, haja vista que muitos comentários já estariam citados na sessão aonde é desenvolvida a aplicação cliente.

3.2.1 Diagrama de Caso de Uso

O diagrama de caso de uso utilizado abstrai uma série de funcionalidades, desta forma, abordando apenas as mais relevantes para o entendimento em alto nível dos requisitos da aplicação.

Em alto nível para processar uma transmissão de vídeo em tempo real são necessárias três etapas diferentes. Com base em Deitel (2001, p.1136): “O processamento da mídia consiste em três partes distintas inicialização do processo, configuração do formato e transmissão de dados”.

Diante do contexto citado por Deitel (2001) foi criado o diagrama de caso de uso representado na Figura 20.

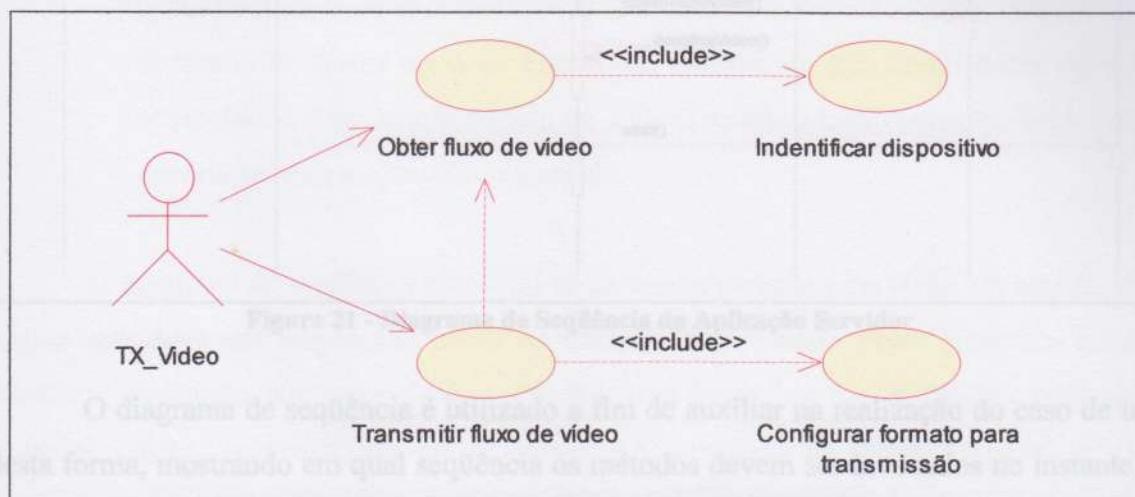


Figura 20 - Diagrama de Caso de Uso da Aplicação Servidor

O caso de uso ilustra além das funcionalidades o nível de dependência entre elas. Desta forma, abordando também uma metodologia de implementação dos casos de uso.

Neste modelo existem dois casos críticos:

- Obter fluxo de vídeo: este caso de uso tem a sua construção baseada em identificar a localização do fluxo de vídeo. Desta forma, incluído um outro caso de uso que tem o papel de identificar o dispositivo de captura de vídeo;
- Transmitir fluxo de vídeo: este caso de uso implementa a transmissão de vídeo mas, para que isso ocorra, o mesmo depende da localização do fluxo de vídeo. Um outro fator é configuração do formato do vídeo que deve ocorrer antes da sua transmissão;

3.2.2 Diagrama de Seqüência

O diagrama de seqüência abordado na Figura 21 ilustra quais métodos são acionados e qual o instante de chamada das operações dentro de uma linha de tempo, onde a seqüência lógica é definitiva para a realização do processo de transmissão.

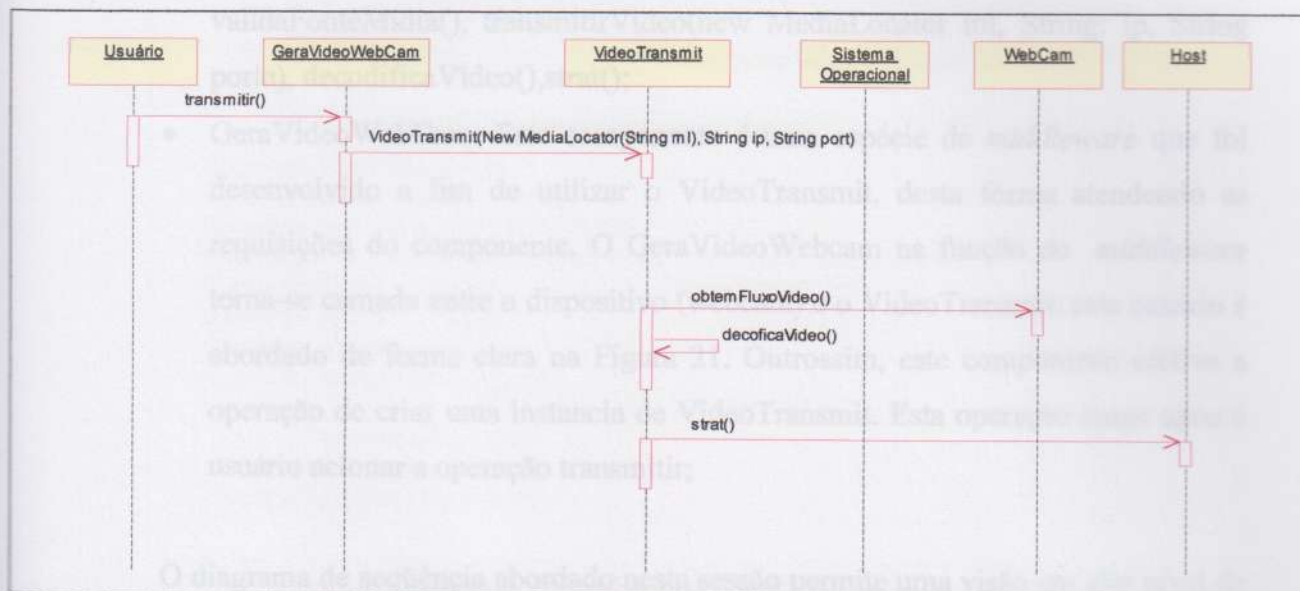


Figura 21 - Diagrama de Seqüência da Aplicação Servidor

O diagrama de seqüência é utilizado a fim de auxiliar na realização do caso de uso, desta forma, mostrando em qual seqüência os métodos devem ser invocados no instante da execução. Portanto, mapeando a troca de mensagens entre os objetos participantes do processo de transmissão.

As entidades mapeadas no diagrama de seqüência disparam chamadas de métodos, os quais podem ser citados:

- **Usuário:** O usuário tem o papel de dar início ao processo de inicialização do servidor de *stream* de vídeo, para tanto este inicializa o servidor através do método **transmitir ()** que na aplicação esta associado ao botão transmitir;
- **VideoTransmit:** Este objeto mapeado no diagrama de seqüência encapsula as operações necessárias para a transmissão de mídias. Neste trabalho este componente foi obtido diretamente do site da SUN, aonde o mesmo é citado no guia da API JMF. Esta estratégia é adotada a fim de acelerar o processo de desenvolvimento, desta forma, implementado uma das propostas da orientação a

objetos, que é a reutilização de código. Este componente tem um papel relevante para aplicação, pois permite realizar a transmissão dos fluxos de stream de vídeo já formatado. Desta forma, viabilizando a transmissão via protocolo RTP. Para tanto, o VideoTransmit dispara em determinados instantes de tempo quatro operações que são fundamentais para todo o processo de transmissão, estas são: `validaFonteMidia()`, `transmitirVideo(new MediaLocator ml, String: ip, String porta)`, `decodificaVideo()`, `strat()`;

- **GeraVideoWebCam:** Este componente é uma espécie de *middleware* que foi desenvolvido a fim de utilizar o VideoTransmit, desta forma atendendo as requisições do componente. O GeraVideoWebcam na função de *middleware* torna-se camada entre o dispositivo (webcam) e o VideoTransmit, este cenário é abordado de forma clara na Figura 21. Outrossim, este componente efetiva a operação de criar uma instancia de VideoTransmit. Esta operação surge após o usuário acionar a operação transmitir;

O diagrama de seqüência abordado nesta sessão permite uma visão em alto nível da lógica que deve ser empregada sobre os componentes. Diante deste contexto, tenta-se nortear a realização dos casos de uso provendo um arcabouço das interações necessárias entre os objetos colaboradores no processo de transmissão de vídeo. O papel de cada componente do modelo pode ser compreendido no diagrama de classes abordado na sessão subsequente, onde é exposta uma especificação das classes, bem como, dos tipos de relacionamentos introduzidos entre elas.

3.2.3 Diagrama de Classes

O diagrama de classes da aplicação servidora aborda os relacionamentos entre as classes que compõe o modelo. Portanto, utilizando como referencia a lógica utilizada no diagrama de seqüência (Figura 21), deste modo, ampliando a visão sobre a problemática empregada no processo de construção dos artefatos de software que implementam a aplicação de transmissão de vídeo.

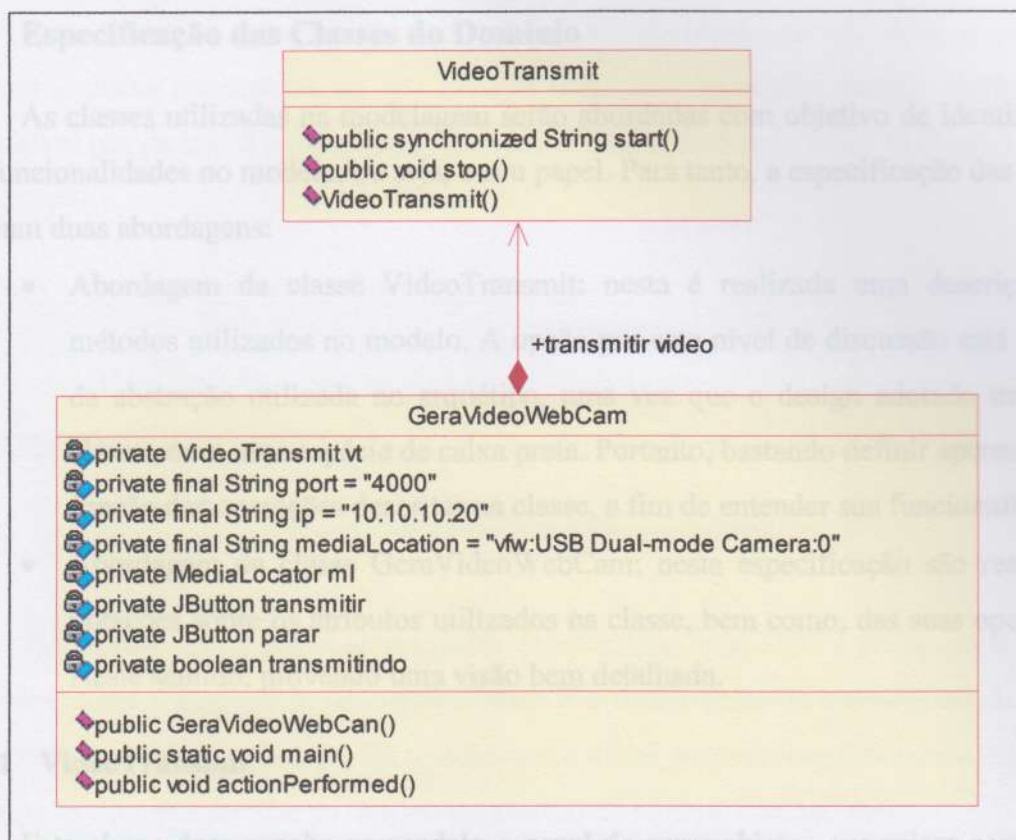


Figura 22 – Domínio da Aplicação Servidor

O modelo descrito na Figura 22 ilustra um relacionamento do tipo agregação, onde é mapeado uma passagem por valor da classe VideoTransmit para a classe GeraVideoWebCam. Este relacionamento especificado por uma agregação unidirecional (*Unidirectional Aggregation*), ocorre pelo fato da classe GeraVideoWebCam ser um artefato intermediário entre VideoTransmit e o dispositivo de captura de vídeo.

A estratégia adotada no modelo tem por objetivo reutilizar de forma transparente as funcionalidades da classe VideoTransmit. Desta forma, reduzindo a quantidade de codificação empregada no desenvolvimento da aplicação para transmissão de vídeo em tempo real. O sentido unidirecional mapeado no relacionamento indica um alto nível de desacoplamento da classe VideoTransmit, o que é um fator interessante com relação a sua reutilização.

3.2.4 Especificação das Classes do Domínio

As classes utilizadas na modelagem serão abordadas com objetivo de identificar as suas funcionalidades no modelo, ou seja, o seu papel. Para tanto, a especificação das classes seguiram duas abordagens:

- Abordagem da classe VideoTransmit: nesta é realizada uma descrição dos métodos utilizados no modelo. A opção por este nível de discussão está no fato da abstração utilizada no arquétipo, uma vez que o design adotado trata esta classe como uma espécie de caixa preta. Portanto, bastando definir apenas qual a função das operações descritas na classe, a fim de entender sua funcionalidade;
- Abordagem da classe GeraVideoWebCam: nesta especificação são realizadas notações sobre os atributos utilizados na classe, bem como, das suas operações. Neste sentido, provendo uma visão bem detalhada.

3.2.4.1 VideoTransmit

Esta classe desempenha no modelo o papel de gerar objetos que sejam capazes de transmitir vídeo em tempo real. De forma efetiva, uma instancia da classe VideoTransmit é iniciada por seu construtor recebendo os seguintes parâmetros: <URL da mídia>, <IP de destino> e <Porta de destino> . Através destes dados informados a um objeto do tipo VideoTrasnmit é possível enviar um fluxo de stream vídeo para alguma máquina na rede.

Tabela 5 - Especificação da classe VideoTransmit

Métodos		
public synchronized String start()	Retorno: void	Este método inicia o processo de transmissão de vídeo
	Parâmetro: -	
public VideoTransmit()	Retorno: -	Este é um método construtor da classe, por tanto é utilizado para inicializar um objeto do tipo VideoTransmit com seus recursos básicos.
	Parâmetro: MediaLocator locator, String ipAddress, String port	
public void stop()	Retorno: void	Este método é utilizado para finalizar o processo de transmissão de vídeo.
	Parâmetro: -	

Para iniciar o processo de transmissão através desta classe, deve-se além de informar o dados necessários no seu método construtor invocar o método start() que é responsável por da início a transmissão do vídeo sobre o protocolo RTP. O Quadro 11 cita um exemplo.

Quadro 11 - Aplicação servidora: código para transmissão

```
// Instanciando um objeto para transmissão de vídeo
VideoTransmit vt = new VideoTransmit(new MediaLocator(url), ip, port);

// Iniciando a transmissão
vt.start();
```

3.2.4.2 GeraVideoWebCam

Esta classe tem o papel de mediador da aplicação, haja vista à ponte que à mesma estabelece entre o dispositivo de captura vídeo e o componente de transmissão de stream mídia. Para uma efetiva operação de middleware a classe mapeia com atributos estáticos: o endereço IP e a porta do host de destino para envio do vídeo em tempo real.

Um outro fator importante é a fonte de mídia que também é mapeada de forma estática, pois nesta aplicação a mesma sempre será obtida através da porta USB, na qual as câmeras digitais convencionais estão conectadas. Desta forma, contribuindo para uma substancial redução de código, uma vez que não será preciso gerar uma rotina para obtenção dinâmica dos *drives* disponíveis pelo S.O. (Sistema Operacional) para identificação da câmera digital.

Tabela 6 - Especificação da classe GeraVideoWebCam

Atributos		
vt	VideoTransmit	Este atributo é um objeto que permitirá executar o processo de transmissão.
final port = "4000";	String	Este atributo é uma constante que armazena a porta de destino para transmissão de vídeo.
final ip = "10.10.10.20"	String	Este atributo é uma constante que armazena o IP do host de destino para transmissão de vídeo.
final mediaLocation = "vfw:USB Dual-mode Camera:0"	String	Este atributo é uma constante que armazena a localização da fonte de mídia.

ml	MediaLocator	Este atributo representa um objeto que guardará ou receberá a localização da fonte de mídia.
transmitir	JButton	Este atributo representa um objeto que servirá na interface gráfica para o usuário acionar o processo de transmissão de vídeo.
parar	JButton	Este atributo representa um objeto que servirá na interface gráfica para que o usuário interrompa o processo de transmissão de vídeo.
transmitindo	boolean	Este atributo é um sinalizador que indica o estado em que se encontra a aplicação (transmitindo ou não vídeo).
Métodos		
GeraVideoWebCam()	Retorno: -	Este é método construtor da classe que inicializa os recursos básicos da aplicação. Um exemplo são os recursos visuais.
	Parâmetro: -	
public void actionPerformed(ActionEvent e)	Retorno: String	Este método é implementado a fim de capturar os eventos de clique nos botões: transmitir e parar.
	Parâmetro: -	
public static void main(String[] args)	Retorno: void	Este método é invocado no momento em que a classe é efetivamente instanciada. A sua função é tornar esta classe passível de execução.
	Parâmetro: String	

A classe GeraVideowebCam tem todo o seu comportamento definido nestes três métodos, onde são desenvolvido todo o comportamento da classe na aplicação. Uma melhor visão em torno da simplicidade e da eficiência desta classe pode ser observada em sua codificação, que segue na sessão 3.2.5.

3.2.5 Codificação e Resultados

Esta sessão detalha os trechos de código mais relevantes da classe GeraVideoWebCam, bem como, cita resultados obtidos com a codificação.

Os resultados nesta sessão são demonstrados através de figuras que ilustram o código em tempo de execução, bem como, uma descrição textual das ocorrências disparadas neste momento.

Para cada quadro onde são citados os trechos de código da aplicação, segue abaixo um comentário com os objetivos alcançados ou com uma descrição da codificação. Desta forma, detalhando os procedimentos que são substanciais durante o desenvolvimento da aplicação.

Quadro 12 - Aplicação servidora: método construtor

```
public GeraVideoWebCam() {
    // Definido o layout do objeto container da aplicação
    setLayout(new BorderLayout());

    // Instanciando o botão que acionará o procedimento de transmissão de
    // vídeo
    transmitir = new JButton("Transmitir");

    // Instanciando o botão que acionará o procedimento de interrupção do
    // processo de transmissão de vídeo

    parar = new JButton ("Para Transmissão");

    // Adicionado aos botões o ActionListener implementado na aplicação
    parar.addActionListener(this);

    transmitir.addActionListener(this);

    // Instanciando o MediaLocator que irá conter a fonte de mídia
    ml = new MediaLocator(mediaLocation);

    // Instanciando o componente de transmissão de vídeo
    vt = new VideoTransmit(ml, ip, port);

    try { // Definindo a posição dos botões no form da aplicação
        add(transmitir, BorderLayout.NORTH);
        add(parar, BorderLayout.SOUTH);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

No Quadro 12 é observado o procedimento para iniciar os componentes fundamentais para que a aplicação possa ser executada. Neste sentido, são observados dois fatores:

- Criação dos componentes visuais e adequação deste a uma lógica de apresentação;
- Criação do componente de transmissão de vídeo passando em seu construtor os dados (constantes da classe) necessários para acionar o fluxo de stream média a partir de uma webcam.

Quadro 13 - Aplicação servidora: método ouvitor

```
public void actionPerformed(ActionEvent e) {  
  
    // Obtém o componente acionador do evento clicar  
    JButton c = (JButton) e.getSource();  
  
    // Verifica se o usuário clicou no botão transmitir e já não havia  
    // clicado antes  
    if ((c == transmitir)&&(!transmitindo)) {  
  
        System.out.println("Vídeo sendo transmitido"+"\\n");  
  
        vt.start();  
  
    // Verifica se o usuário clicou no botão parar  
    }else if(c == parar){  
  
        System.out.println("\\n"+"Transmissão de vídeo parada"+"\\n");  
        vt.stop();  
  
    }  
  
}
```

O trecho de código ilustrado no Quadro 13 aborda o método implementado para servir como observador dos eventos que podem ser acionados pelo usuário. Para tanto, é utilizado o método *getSource()* para obter o botão acionador do evento. Desta forma, tratando a ação e disparando os procedimentos de transmitir ou de interromper a transmissão, de acordo com o botão acionador na interface do sistema.

Figura 23 - Console do servidor de stream de vídeo

Todo código empregado no desenvolvimento desta aplicação encontra-se devidamente comentado e segue nos anexos da monografia. Desta forma, para uma visualização bem específica da codificação é recomendado à leitura dos anexos que podem ser transpostos a uma ferramenta de desenvolvimento Java para teste mais efetivos, o que poderá facilitar o entendimento da codificação.

3.2.6 Resultados obtidos com a codificação do modelo da Aplicação Servidor

Através da aplicação desenvolvida para a transmissão de vídeo em tempo real foi possível gerar de forma consistente um fluxo de *stream* mídia que pudesse ser obtidos com sucesso por uma aplicação reprodutora de mídia. A aplicação utilizada para reproduzir o vídeo capturado de uma webcam foi a desenvolvida por este trabalho. Desta forma, consolidando a coesão entre as técnicas de hipermídia utilizada em tal aplicação, com os recursos disponibilizados pela API JMF para a transmissão de vídeo em tempo real.

A execução das aplicações pode ser observada nas figuras 23 e 24, as quais ilustram toda a codificação escrita em tempo de execução.

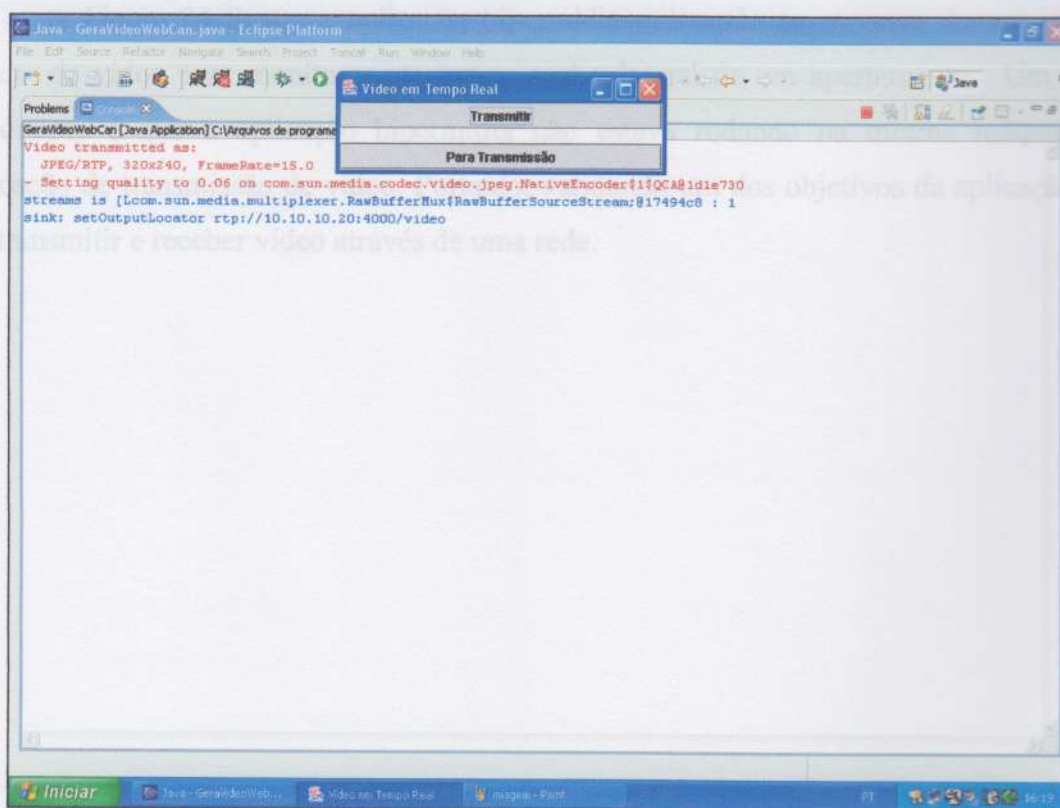


Figura 23 - Console do servidor de stream de vídeo

A Figura 23 ilustra a execução do servidor de *stream* mídia capturando e enviando vídeo em tempo real para uma máquina situada em rede local. Para que a ação de enviar vídeo seja consolidada, primeiro o usuário clicou no botão transmitir e logo em seguida o servidor de stream mídia exibiu algumas informações em seu console. As mensagens exibidas indicam:

- O estado em que o servidor se encontra: transmitindo ou parado;
- Formato em que o vídeo está sendo multiplexado, tamanho de exibição do vídeo e número de quadros (não é um valor preciso, pode variar para mais ou para menos): “JPEG/RTP, 320x240, FrameRate=15.0”;
- Qualidade empregada nas imagens multiplexadas e *plugin* utilizado no processo de multiplexação: “Setting quality to 0.06 on com.sun.media.codec.video.jpeg.NativeEncoder\$1\$QCA@18941f7”;
- Local de destino do envio do fluxo de vídeo: “streams is [Lcom.sun.media.multiplexer.RawBufferMux\$RawBufferSourceStream;@11ca803 : 1 sink: setOutputLocator rtp://10.10.10.20:4000/video”.

A Figura 24 ilustra a aplicação hipermídia utilizando os recursos do servidor de *stream* de vídeo para monitorar em tempo real a lateral de um apartamento. Um fator, importante é que a aplicação hipermídia não estava rodando na mesma máquina da aplicação de transmissão de vídeo. Portanto, validando um dos objetivos da aplicação que era transmitir e receber vídeo através de uma rede.

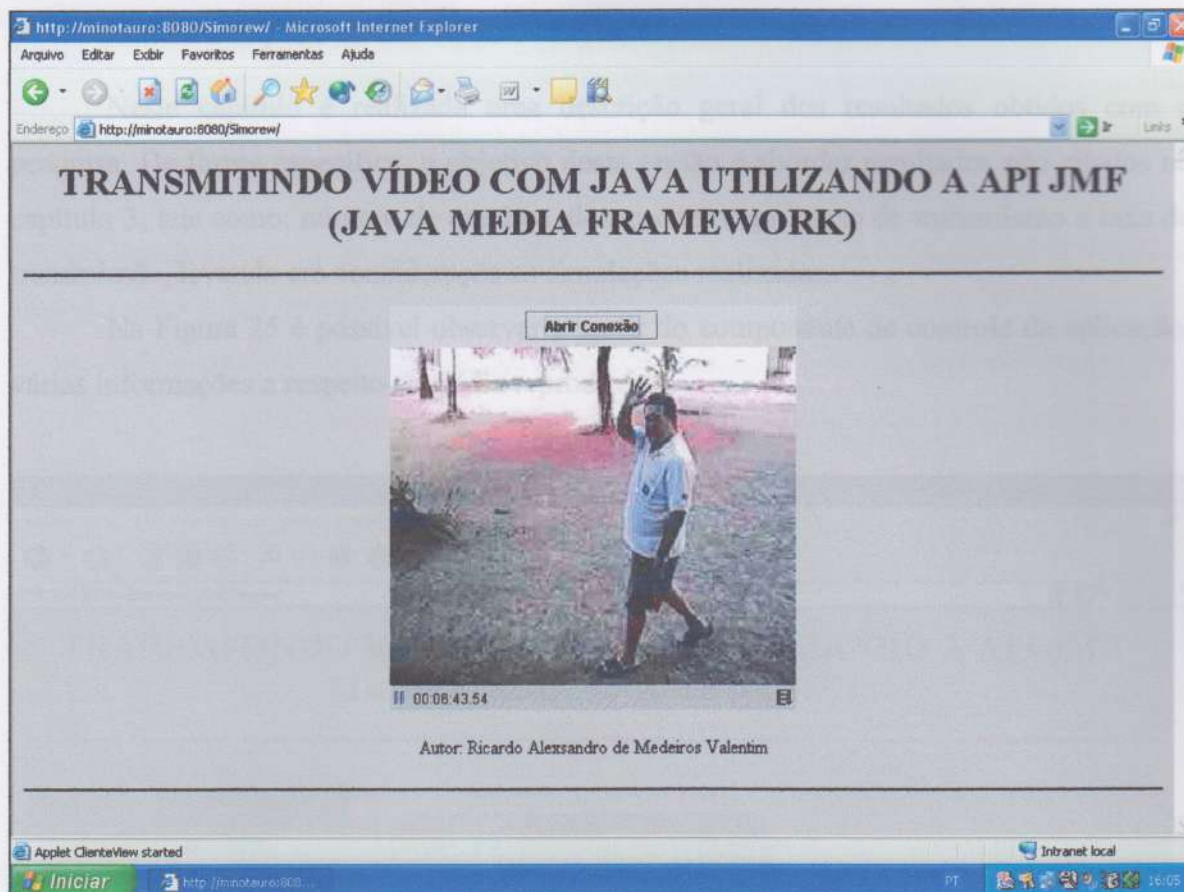


Figura 24 - Aplicação cliente capturando vídeo em tempo real

Figura 25 - Dados a respeito da transmissão

Os dados ilustrados nesta figura são:

- Tempo de execução do vídeo: 00:01:01:10;
- Taxa de transmissão: 1294.043 Kbps;
- Número de quadros: 21 fps.

4 VERIFICAÇÃO DOS RESULTADOS

Neste capítulo é realizada uma descrição geral dos resultados obtidos com a pesquisa. De forma específica, o objetivo desta sessão é abordar resultados não citados no capítulo 3, tais como: número de quadros alcançados no processo de transmissão e taxa de transmissão, levando em consideração as simulações realizadas.

Na Figura 25 é possível observar a partir do componente de controle da aplicação, várias informações a respeito da mídia reproduzida.

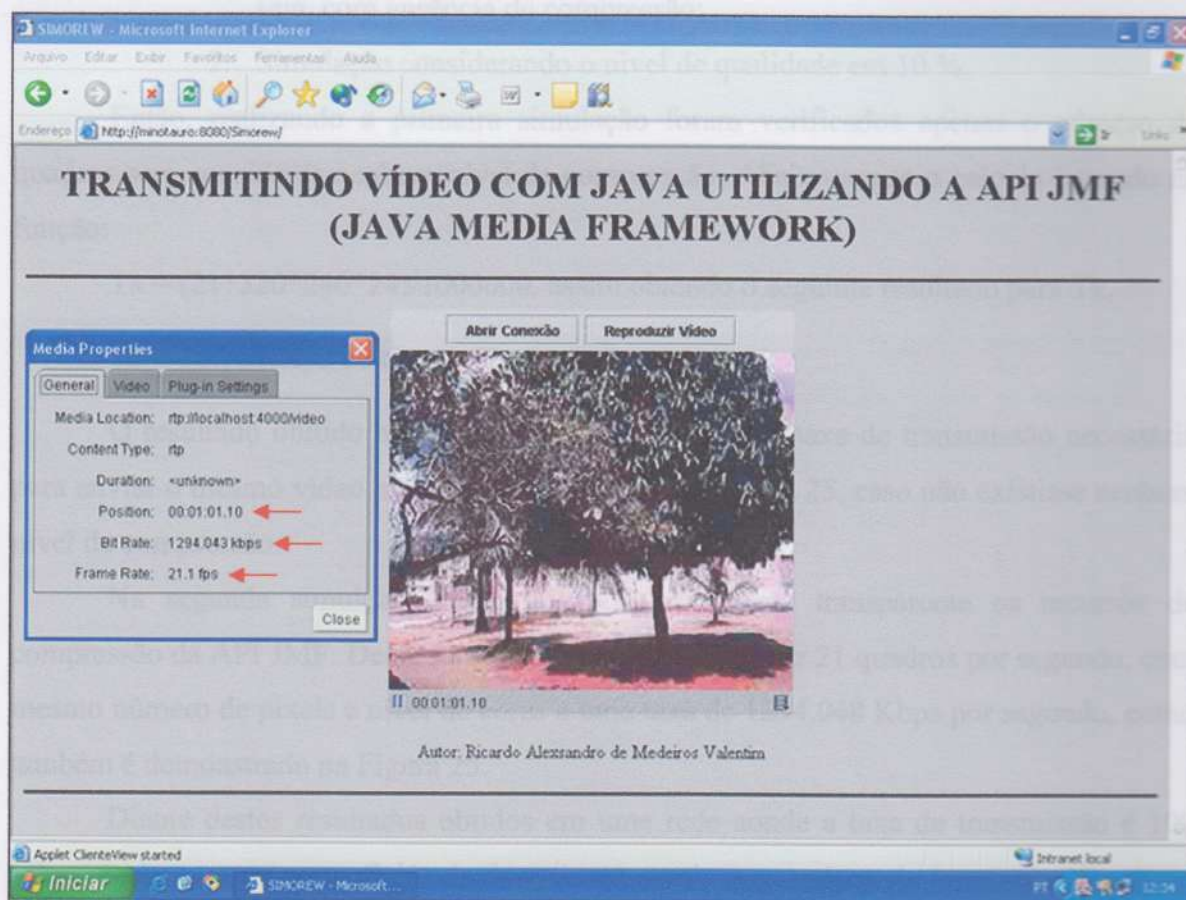


Figura 25 - Dados a respeito da transmissão

Os dados ilustrados nesta figura são:

- Tempo de exibição do vídeo: 00:01:01:10;
- Taxa de transmissão: 1294.043 Kbps;
- Número de quadros: 21 fps.

Com base nos dados referentes à taxa de transmissão e número de quadros por segundo será avaliado o nível de compressão aplicado aos bits de vídeo capturados na fonte de mídia, ou seja, qual o percentual aplicado pelo multiplexador da API JMF sobre os quadros de vídeos antes do instante da transmissão. Para efetuar a análise do nível de compressão alcançada pela aplicação, será utilizada a seguinte função: $T_x = N_q * N_p$ (esta função foi discutida no segundo capítulo).

Este processo de verificação de resultados realiza uma avaliação sobre dois prismas:

1. Simulação considerando o nível de qualidade do vídeo em 100%, ou seja, com ausência de compressão;
2. Simulação considerando o nível de qualidade em 10 %.

Então, realizando a primeira simulação foram verificados apenas o número de quadros sem considerar nenhum nível de compressão. Abaixo segue o calculo baseado na função:

$T_x = (21 * 320 * 240 * 24) / 1000000$, assim obtendo o seguinte resultado para T_x ,

$T_x = 38.70$ Mbps de taxa de transmissão,

O resultado obtido na primeira simulação indica a taxa de transmissão necessária para enviar o mesmo vídeo exibido pela aplicação na Figura 25, caso não existisse nenhum nível de compressão.

Na segunda simulação foram utilizados de forma transparente os recursos de compressão da API JMF. Desta forma, foi possível transmitir 21 quadros por segundo, com mesmo número de pixels e nível de cores a uma taxa de 1294.048 Kbps por segundo, como também é demonstrado na Figura 25.

Diante destes resultados obtidos em uma rede aonde a taxa de transmissão é 100 Mbps, pode-se verificar a eficiência da aplicação, pois economizou de forma significativa a largura de banda da rede. Caso não existisse essa possibilidade de compressão, mais de 38% da largura de banda estaria sendo utilizada apenas pela aplicação de vídeo.

Realizado um estudo na codificação classe VideoTransmit, foi observado que poderia ser especificado um nível de qualidade²⁰ sobre os quadros exibidos no processo de transmissão de vídeo. Nos comentários encontrados na classe estava especificado que um

²⁰ O nível de qualidade pode variar entre 0 e 1 na casa dos números reais.

bom nível para exibição das imagens seria de 0.5 (50%), no entanto com este nível só possível obter entre 11 e 15 quadros por segundo. Portanto, a fim de elevar o número de quadros por segundo esse valor foi reduzido para 0.1 (10%), onde foi possível chegar até 29 quadros por segundo. Foram realizados experimentos reduzindo o nível até 0.06, mas observou-se que não mais influenciava sobre as taxas de transmissão.

Mesmo alcançando níveis muito bons de compressão de vídeo, pode-se verificar a necessidade de largura de banda para obter uma qualidade desejável, de modo que se torna basicamente inviável transmitir vídeo em tempo real para redes que não possuem uma infra-estrutura adequada.

Por fim, as validações verificadas na API JMF no tocante a transmissão de vídeo em tempo real mostraram-se bastante eficazes. Isso porque, foi possível desenvolver uma aplicação que viabilizou uma transmissão fim a fim de *stream* de vídeo de maneira relativamente simples.

A simplicidade em torno do desenvolvimento ocorre pela metodologia utilizada na Plataforma Java, que desenvolve suas APIs abstraindo detalhes de baixo nível. Esse fator foi claramente observado na utilização da API JMF durante todo o processo de desenvolvimento do sistema.

Interessante neste cenário foi possível visualizar uma continuidade deste trabalho nas áreas de transmissão de vídeo em tempo real sob demanda, como também nas áreas de transmissão de vídeo em tempo real em redes multicast. Estas duas áreas estão bastante próximas em vários cenários tais como: nas áreas de vídeo conferência, tele-medical, segurança entre outras. Sendo assim, este trabalho poderá servir de subsídio para um estudo de tecnologias com Java e Multimídia que a cada dia tornam-se mais presentes no cotidiano social.

5 CONSIDERAÇÕES FINAIS

Nunca existiram tantas possibilidades de comunicação quanto às disponibilizadas nos dias atuais, haja vista, os vários meios desenvolvidos para a propagação da informação. Foi neste sentido, que este trabalho se propôs a realizar um estudo que contemplasse permear os meandros da transmissão de vídeo em tempo real. Para tanto, pesquisando uma tecnologia que fornecesse tais possibilidades, tal é a tecnologia Java.

Diante de uma grande rede de comunicação atualmente estabelecida na sociedade contemporânea, cresce também a necessidade de expandir não só os meios de comunicação, mas também a qualidade em que as informações trafegam por este meio.

Motivado pela perspectiva de poder dar uma contribuição mais efetiva junto a esta sociedade em que os avanços da tecnologia tem tornado os limites geográficos bastante estreitos, toda a pesquisa desenvolvida por este trabalho monográfico visou sempre realizar um estudo que pudesse servir de parâmetro para outros projetos de pesquisas, sejam estes acadêmicos ou corporativos.

Contextualizado pelas ações em torno deste trabalho de pesquisa, foi possível realizar um aprofundamento substancial sobre todos os conceitos abordados. Desta forma, cumprindo as metas da academia, pois foi possível obter de forma sólida e coesa uma necessária fundamentação teórica e empírica.

Imerso neste cenário foi possível visualizar uma continuidade deste trabalho nas áreas de transmissão de vídeo em tempo real sob demanda, como também nas áreas de transmissão de vídeo em tempo real em redes multicast. Estas duas áreas estão bastante atuantes em vários cenários tais como: nas áreas de vídeo conferencia, tele-medicina, segurança entre outras. Sendo assim, este trabalho poderá servir de embasamento para um estudo de tecnologias com Java e Multimídia que a cada dia iram tornar-se mais presente no cotidiano social.

REFERÊNCIA BIBLIOGRÁFICA

- BOGGS, W.; BOGGS, M. **Mastering UML com Rational Rose 2002**. Rio de Janeiro: alta Books. 2002.
- CESTA, A. A.; RUBIRA, C. M. F. **Tutorial: A linguagem de Programação Java – Orientação a Objetos**. Campinas: Instituto de Computação da UNICAMP, 2002.
- COMER, D. E. **Redes de Computadores e Internet**: 2. ed. Porto Alegre: Bookman. 2001.
- CONCEIÇÃO, A. F. **Estudo Preliminar Sobre Transmissão de Multimídia em Computação Móvel**. São Paulo: USP - Universidade Estadual de São Paulo. Disponível em: <<http://www.ime.usp.br/~arlindo/articile/mobile-multimidia.doc>>. Acesso em: 28 jan. 2004.
- DEITEL, H. M.; DEITEL, P.J. **Java Como Programar**. 3. ed. Porto Alegre: Bookman. 2001.
- _____. **Java Media Framework e Java Sound**. 4. ed. Porto Alegre: Bookman. CD-ROM. 2001.
- LAR. **Sistema Digital de Vídeo Vigilância**. Curitiba: Lar Tecnologia e Automação de Ambientes. Disponível em: <<http://www.lartecnologia.com.br/page13.html>>. Acesso em: 23 jan. 2004.
- LARMAN, C. **Utilizando UML e Padrões**: uma introdução à análise e ao projeto orientado a objetos. Porto Alegre: Bookman. 2002.
- MEDEIROS, M. S. “**Trabalho Sobre os Protocolos de Suporte a Multimídia Rtp, Rtcp e Rtsp**”. Artigo. 2003. Departamento de Engenharia Elétrica – COPPE/UFRJ.
- MACHADO, C. E. R. **Desenvolvimento de uma Aplicação Multimídia de Videoconferência em Java**. 2001. Monografia (Ciências da Computação) UNIMEP – Universidade Metodista de Piracicaba, Piracicaba.
- NIEMEYER, P.; KNUDSEN, J. **Aprendendo Java**. Rio de Janeiro: Campus. 2000.
- PAGE-JONES, M. **Fundamentos do Desenho Orientado a Objetos com UML**. São Paulo: Makron Books. 2001.
- PÁTIO VIRTUAL. **Pais aprovam sistemas de vigilância nos colégios**. São Paulo: Pátio Virtual. Disponível em: <<http://www.patiovirtual.com.br/noticia1.asp>>. Acesso em: 13 jan. 2004.
- _____. **Câmeras em escola tranquilizam pais de alunos**: impossibilitados pelo trabalho, eles aprovam tecnologias que viabilizam a monitoração à distância. São Paulo: Pátio

Virtual. Disponível em: <<http://www.patiovirtual.com.br/noticia1.asp>>. Acesso em: 14 jan. 2004.

PAULO FILHO, W. P. **Multimídia Conceitos e Aplicações**. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora. 2000.

PROJETEC. **Sistema de Monitoramento através de câmeras**. Novo Hamburgo: Projetec Informática. Disponível em: <www.projetec.com.br/monit.htm>. Acesso em: 23 jan. 2004.

PRESSMAM, R S. **Engenharia de Software**. São Paulo: Markron Books. 1995.

RUMBAUGH, J.; BLAHA, M.; WILLIAM, P.; EDDY, F.; LORESEN, W. **Modelagem e Projetos Baseados em Objetos**. Rio de Janeiro: Campus. 1994.

SOARES, L. F. G.; LEMOS, G.; COLCHER, S. **Redes de Computadores: das LANs, MANs e WANs às Redes ATM**. 2. ed. Rio de Janeiro: Campus. 1995.

SUN. **Java Media Framework API Guide**. Sun Microsystem. Disponível em: <<http://www.java.sun.com/jmf/>>. Acesso em 05 fev 2004.

WOLFMAN, D. E. **Criando em Multimídia**. Rio de Janeiro: Campus. 1994.

ANEXOS

ANEXO 1 - CÓDIGO DA APLICAÇÃO CLIENTE

Quadro 14 - CÓDIGO FONTE ClienteVies.java

```
// Cliente Vies para controlar a saída
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;
import java.util.logging.*;
```

```
// Classe Vies para controlar a saída
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;
import java.util.logging.*;

// Classe Vies para controlar a saída
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;
import java.util.logging.*;

// Classe Vies para controlar a saída
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;
import java.util.logging.*;

// Classe Vies para controlar a saída
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;
import java.util.logging.*;
```

ANEXOS

ANEXO 1 – CÓDIGO DA APLICAÇÃO CLIENTE

Quadro 14 - CÓDIGO FONTE: ClienteView.java

```
//Classe Player para tratar a mídia.
import javax.media.ControllerAdapter;
import javax.media.EndOfMediaEvent;
import javax.media.Manager;
import javax.media.MediaLocator;
import javax.media.NoPlayerException;
import javax.media.Player;
import javax.media.PrefetchCompleteEvent;
import javax.media.RealizeCompleteEvent;
import javax.media.Time;

//Classe JBUTTON para tratar os eventos um usuário poderá disparar.
import javax.swing.JButton;
//Classe JFileChooser responsavel por instanciar objetos do tipo caixa de
// dialogo para que o usuário possa obter um arquivo de mídia.
import javax.swing.JFileChooser;
//Classe JOptionPane responsavel por exibir mensagens em caixas de
//diálogos.
import javax.swing.JOptionPane;
//Classe JPanel para servir conteineiner dos botões.
import javax.swing.JPanel;
//Classe Component e Container para servir de container do conteúdo
//visual e do controle da aplicação.
import java.awt.Component;
import java.awt.Container;
//Classe BorderLayout para servir de gerenciador de layout da
//aplicação.
import java.awt.BorderLayout;
//Classe ActionListener que serve de calsse anonima para operacionalizar
//os eventos que serão instanciados apartir da classe ActionEvent.
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```

//Classe File representará uma instância do arquivo de mídia
import java.io.File;
//Classe URL representará uma instância da localização do arquivo de
//mídia
import java.net.URL;
// A classe IOException representa uma possível exceção gerada no
//momento de escrita ou leitura de arquivos de mídia.
import java.io.IOException;
//Classe MalformedURLException que exibe e captura erros de mal
//formação de urls.
import java.net.MalformedURLException;
// Esta classe é utilizada para obter um instante de tempo.
import java.util.Timer;

/**
 * @author Ricardo Alexsandro de Medeiros Valentim
 */
public class ClienteView extends javax.swing.JApplet {

    private Player player;// Objeto player
    private Component visualMedia;// Componente para conteúdo visual.
    private Component mediaControl;// Componente para conteúdo de controle
    //visual.
    private Container container; // Container principal.
    private URL arquivoURL; // Objeto que irá conter uma URL que aponta para
    //arquivo de mídia;

    //Este método é equivalente a um construtor.
    public void init() {
        this.container = this.getContentPane();// Obtendo um container.
        JPanel buttonPanel = new JPanel();//Instanciando um JPanel para
        //servir de container dos botões.

        /*
        * Adicionado o objeto buttonPanel ao container principal e movendo o
        * mesmo para a parte superior através do BorderLayout.NORTH
        */
    }

```

```

this.container.add(buttonPanel, BorderLayout.NORTH);

// Instanciando e atribuindo um botão o seu objeto container.
// Botão que irá acionar um evento que permitirá
// localizar uma URL.
JButton abrirURL = new JButton("Abrir Conexão");
buttonPanel.add(abrirURL);
// Associando um evento ao botão abrir URL.
abrirURL.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent event) {
            String nomeEndereco = getMediaLocalizacao();//Solicita que
//seja informado a localização da mídia
            if (nomeEndereco != null) {
                criarPlayer(nomeEndereco);// Cria um player.
            }
        }
    }
);

// Instanciando e atribuindo um botão o seu objeto container.
// Botão que irá acionar um evento que permitirá
// reproduzir um vídeo a partir de uma URL Padrão.
JButton abrirURLPadrao = new JButton("Reproduzir Vídeo");
buttonPanel.add(abrirURLPadrao);

// Associando um evento ao botão abrir URL Padrão.
abrirURLPadrao.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent event) {
            String nomeEndereco = "rtp://localhost:4000/video";

            if (nomeEndereco != null) {
                criarPlayer(nomeEndereco);// Cria um player.
            }
        }
    }
);

```

```

    }

    });

    // Ativando uma geração leve dos player para permitir
    // uma melhor compatibilidade com os componentes de Interface.
    Manager.setHint(Manager.LIGHTWEIGHT_RENDERER, Boolean.TRUE);
    this.setSize(300, 300);
    this.setVisible(true);
} // Final do init() do JApplet (ClienteView)

/**
 * @return String Este método retorna o endereço da mídia informado pelo
 * usuário
 */
public String getMediaLocalizacao() {
    String entrada = JOptionPane.showInputDialog(this, "Informe a URL");
    if (entrada != null && entrada.length() == 0) { // Testa se o usuário
//apertou Ok sem digitar nada.
        return null;

    } else {
        return entrada;
    }
}

/**
 * @return File Este método permite ao usuário localizar o arquivo de
 * mídia forma visual.
 */
private File getArquivo() {
    JFileChooser fileChooser = new JFileChooser(); // Instanciando um
//objeto visual que abre um caixa de dialogo para localizar o arquivo.

    // Lista os arquivos que podem ser lidos
    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    int resultado = fileChooser.showOpenDialog(this); // Abre a caixa de
//diálogos

```

```

        if (resultado == JFileChooser.CANCEL_OPTION) { // Testa se o usuário
//cancelou a operação.
            return null; // Se tiver cancelado retorna nulo.
        } else {
            return fileChooser.getSelectedFile(); // Caso contrario retorna
//um objeto do tipo File
        }
    }

    public void getErro(String erro) {
        JOptionPane.showMessageDialog(this, erro, "Erro",
            JOptionPane.ERROR_MESSAGE);
    }

    /**
     * @param localizacaoMidia
     * Cria um Player a partir do endereço da Midia
     */
    public void criarPlayer(String localizacaoMidia) {
        /*
         * Esta condição verifica se existe um player, caso seja verdade este
         * removido para que seja criado outro
         */
        if (this.player != null) {
            this.removeComponentesPlayer();
        }

        // Instanciando um objeto que representa o endereço de origem da mídia
        MediaLocator mediaLocator = new MediaLocator(localizacaoMidia);
        // Testa se o mediaLocator é nulo, caso seja é porque não foi possível
        // abrir o arquivo, ou o caminho não estava correto.
        if (mediaLocator == null) {
            // Utilizando o método getErro() para informar uma ocorrência
            this.getErro("ERRO - Não foi possível abrir o arquivo");
            return;
        }
        try{
            player = Manager.createPlayer(mediaLocator); // Cria um player a partir
            //de um MediaLocator

```

```

// Registra um controllerListener para o tratar os eventos do player
    this.player.addControllerListener(new ManipuladorEvento());
    this.player.realize();// Chama o método realize para permitir que
//inicie o processo de reprodução pelo player.
} catch (NoPlayerException noPlayerException) {
    noPlayerException.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}

/**
 * Este método adiciona os componente visual e o de controle para a mídia
 * especificada.
 */
public void getMidiaComponente() {
    // Obtém um componente visual para o Player.
    visualMedia = player.getVisualComponent();

    // Adiciona o componente visual, caso esteja presente.
    if (this.visualMedia != null) {
        container.add(visualMedia, BorderLayout.CENTER);
    }

    // Adiciona o componente de controle, caso esteja presente.
    mediaControl = player.getControlPanelComponent();
    if (mediaControl != null) {
        container.add(mediaControl, BorderLayout.SOUTH);
    }
}

/**
 * Este método liberar o Player para os recursos do sistema e restaura a
 * mídia.
 */
public void removeComponentesPlayer() {
    // remove o componente de vídeo anterior, caso exista.
    if (this.visualMedia != null) {
        this.container.remove(visualMedia);
    }
}

```


ANEXO 2 – CÓDIGO DA APLICAÇÃO SERVIDORA

Quadro 15 - CÓDIGO FONTE: GeraVideoWebCam.java

```
/*
 * GeraVideoWebCan.java
 *
 * Criada em 29 de Julho de 2003, 23:49
 * Modificada em 15 de novembro de 2004, 04:30 (Madrugada)
 */

/**
 * @author Ricardo A. de M. Valentim
 */

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;

import javax.media.MediaLocator;

public class GeraVideoWebCam extends Panel implements ActionListener {

    // Atributo que representa um transmissor em RTP streaming vídeo
    private VideoTransmit vt;

    //Atributos de configuração para transmissão de vídeo
    private final String port = "4000";

    private final String ip = "10.10.10.20";

    private final String mediaLocation = "vfw:USB Dual-mode Camera:0";
```

```

private MediaLocator ml = null;

private JButton transmitir = null;

private JButton parar = null;

private boolean transmitindo = false;

public GeraVideoWebCam() {

    // Definido o layout do objeto container da aplicação
    setLayout(new BorderLayout());

    //Instanciando o botão que acionará o procedimento de transmissão de
    //vídeo
    transmitir = new JButton("Transmitir");

    //Instanciando o botão que acionará o procedimento de interrupção do
    //processo de transmissão de vídeo

    parar = new JButton ("Para Transmissão");

    // Adicionado aos botões o ActionListener implementado na aplicação
    parar.addActionListener(this);

    transmitir.addActionListener(this);

    // Instanciando o MediaLocator que irá conter a fonte de mídia
    ml = new MediaLocator(mediaLocation);

    // Instanciando o componente de transmissão de vídeo
    vt = new VideoTransmit(ml, ip, port);

    try {
        // Definindo a posição dos botões no form da aplicação
        add(transmitir, BorderLayout.NORTH);
        add(parar, BorderLayout.SOUTH);
    }
}

```

```

        } catch (Exception e) {

            e.printStackTrace();

        }
    }

    public static void main(String[] args) {

        // Criando um form para conter a aplicação
        Frame f = new Frame("Video em Tempo Real");

        GeraVideoWebCam gvwc = new GeraVideoWebCam();

        //Criando uma classe anônima que verifica a ação de fechamento da
        //aplicação
        f.addWindowListener(new WindowAdapter() {

            public void windowClosing(WindowEvent e) {

                System.exit(0);

            }

        });

        //Adicionando a aplicação ao objeto form
        f.add("Center", gvwc);

        // Organizando os componentes sobre o form.
        f.pack();

        //Definindo o tamanho do form.
        f.setSize(new Dimension(320, 100));

        // Tornando o form visível
        f.setVisible(true);
    }

```

```

public void actionPerformed(ActionEvent e) {

    // Obtém o componente acionador do evento clicar
    JButton c = (JButton) e.getSource();

    // Verifica se o usuário clicou no botão transmitir e já não havia
    // clicado antes
    if ((c == transmitir)&&(!transmitindo)) {

        System.out.println("Vídeo sendo transmitido"+"\\n");
        vt.start();

        // Verifica se o usuário clicou no botão parar
    }else if(c == parar){

        System.out.println("\\n"+"Transmissão de vídeo
parada"+"\\n");
        vt.stop();

    }

}

} //Final da classe GeraVideoWebCam

```

ANEXO 3 – CÓDIGO DO COMPONENTE DA SUN

Quadro 16 - CÓDIGO FONTE: VideoTransmit.java

```
package br.farn.simorew;

/*
 * @(#)VideoTransmit.java      1.7 01/03/13
 *
 * Copyright (c) 1999-2001 Sun Microsystems, Inc. All Rights Reserved.
 */

import java.awt.*;
import javax.media.*;
import javax.media.protocol.*;
import javax.media.protocol.DataSource;
import javax.media.format.*;
import javax.media.control.TrackControl;
import javax.media.control.QualityControl;
import java.io.*;

public class VideoTransmit {

    // Input MediaLocator
    // Can be a file or http or capture source
    private MediaLocator locator;
    private String ipAddress;
    private String port;

    private Processor processor = null;
    private DataSink  rtptransmitter = null;
    private DataSource dataOutput = null;

    public VideoTransmit(MediaLocator locator,
                        String ipAddress,
                        String port) {

        this.locator = locator;
    }
}
```

```

    this.ipAddress = ipAddress;
    this.port = port;
}

/**
 * Starts the transmission. Returns null if transmission started ok.
 * Otherwise it returns a string with the reason why the setup
 * failed.
 */
public synchronized String start() {
    String result;

    // Create a processor for the specified media locator
    // and program it to output JPEG/RTP
    result = createProcessor();
    if (result != null)
        return result;

    // Create an RTP session to transmit the output of the
    // processor to the specified IP address and port no.
    result = createTransmitter();
    if (result != null) {
        processor.close();
        processor = null;
        return result;
    }

    // Start the transmission
    processor.start();

    return null;
}

/**
 * Stops the transmission if already started
 */
public void stop() {
    synchronized (this) {

```

```

        if (processor != null) {
            processor.stop();
            processor.close();
            processor = null;
            rtptransmitter.close();
            rtptransmitter = null;
        }
    }
}

private String createProcessor() {
    if (locator == null)
        return "Locator is null";

    DataSource ds;
    DataSource clone;

    try {
        ds = Manager.createDataSource(locator);
    } catch (Exception e) {
        return "Couldn't create DataSource";
    }

    // Try to create a processor to handle the input media locator
    try {
        processor = Manager.createProcessor(ds);
    } catch (NoProcessorException npe) {
        return "Couldn't create processor";
    } catch (IOException ioe) {
        return "IOException creating processor";
    }

    // Wait for it to configure
    boolean result = waitForState(processor, Processor.Configured);
    if (result == false)
        return "Couldn't configure processor";

    // Get the tracks from the processor

```

```

TrackControl [] tracks = processor.getTrackControls();

// Do we have atleast one track?
if (tracks == null || tracks.length < 1)
    return "Couldn't find tracks in processor";

boolean programmed = false;

// Search through the tracks for a video track
for (int i = 0; i < tracks.length; i++) {
    Format format = tracks[i].getFormat();
    if ( tracks[i].isEnabled() &&
        format instanceof VideoFormat &&
        !programmed) {

        // Found a video track. Try to program it to output JPEG/RTP
        // Make sure the sizes are multiple of 8's.
        Dimension size = ((VideoFormat)format).getSize();
        float frameRate = ((VideoFormat)format).getFrameRate();
        int w = (size.width % 8 == 0 ? size.width :
            (int)(size.width / 8) * 8);
        int h = (size.height % 8 == 0 ? size.height :
            (int)(size.height / 8) * 8);

        VideoFormat jpegFormat = new
VideoFormat(VideoFormat.JPEG_RTP,
                                new Dimension(w, h),
                                Format.NOT_SPECIFIED,
                                Format.byteArray,
                                frameRate);

        tracks[i].setFormat(jpegFormat);
        System.err.println("Video transmitted as:");
        System.err.println(" " + jpegFormat);
        // Assume succesful
        programmed = true;
    } else
        tracks[i].setEnabled(false);
}

```

```

        if (!programmed)
            return "Couldn't find video track";

        // Set the output content descriptor to RAW RTP
        ContentDescriptor cd = new
ContentDescriptor(ContentDescriptor.RAW_RTP);
        processor.setContentDescriptor(cd);

        // Realize the processor. This will internally create a flow
        // graph and attempt to create an output datasource for JPEG/RTP
        // video frames.
        result = waitForState(processor, Controller.Realized);
        if (result == false)
            return "Couldn't realize processor";

        // Set the JPEG quality to .5.
        setJPEGQuality(processor, 0.5f);

        // Get the output data source of the processor
        dataOutput = processor.getDataOutput();
        return null;
    }

    // Creates an RTP transmit data sink. This is the easiest way to
//create
    // an RTP transmitter. The other way is to use the RTPSessionManager
//API.
    // Using an RTP session manager gives you more control if you wish to
    // fine tune your transmission and set other parameters.
    private String createTransmitter() {
        // Create a media locator for the RTP data sink.
        // For example:
        // rtp://129.130.131.132:42050/video
        String rtpURL = "rtp://" + ipAddress + ":" + port + "/video";
        MediaLocator outputLocator = new MediaLocator(rtpURL);

        // Create a data sink, open it and start transmission. It will wait

```

```

// for the processor to start sending data. So we need to start the
// output data source of the processor. We also need to start the
// processor itself, which is done after this method returns.
try {
    rtptransmitter = Manager.createDataSink(dataOutput,
outputLocator);
    rtptransmitter.open();
    rtptransmitter.start();
    dataOutput.start();
} catch (MediaException me) {
    return "Couldn't create RTP data sink";
} catch (IOException ioe) {
    return "Couldn't create RTP data sink";
}

return null;
}

/**
 * Setting the encoding quality to the specified value on the JPEG
 * encoder.
 * 0.5 is a good default.
 */
void setJPEGQuality(Player p, float val) {

    Control cs[] = p.getControls();
    QualityControl qc = null;
    VideoFormat jpegFmt = new VideoFormat(VideoFormat.JPEG);

    // Loop through the controls to find the Quality control for
    // the JPEG encoder.
    for (int i = 0; i < cs.length; i++) {

        if (cs[i] instanceof QualityControl &&
            cs[i] instanceof Owned) {
            Object owner = ((Owned)cs[i]).getOwner();

```

```

        // Check to see if the owner is a Codec.
        // Then check for the output format.
        if (owner instanceof Codec) {
            Format fmts[] =
((Codec)owner).getSupportedOutputFormats(null);
            for (int j = 0; j < fmts.length; j++) {
                if (fmts[j].matches(jpegFmt)) {
                    qc = (QualityControl)cs[i];
                    qc.setQuality(val);
                    System.err.println("- Setting quality to " +
                        val + " on " + qc);
                    break;
                }
            }
            if (qc != null)
                break;
        }
    }
}

/*****
 * Convenience methods to handle processor's state changes.
 *****/

private Integer stateLock = new Integer(0);
private boolean failed = false;

Integer getStateLock() {
    return stateLock;
}

void setFailed() {
    failed = true;
}

private synchronized boolean waitForState(Processor p, int state) {

```

```

p.addControllerListener(new StateListener());
failed = false;

// Call the required method on the processor
if (state == Processor.Configured) {
    p.configure();
} else if (state == Processor.Realized) {
    p.realize();
}

// Wait until we get an event that confirms the
// success of the method, or a failure event.
// See StateListener inner class
while (p.getState() < state && !failed) {
    synchronized (getStateLock()) {
        try {
            getStateLock().wait();
        } catch (InterruptedException ie) {
            return false;
        }
    }
}

if (failed)
    return false;
else
    return true;
}

/*****
 * Inner Classes
 *****/

class StateListener implements ControllerListener {

    public void controllerUpdate(ControllerEvent ce) {

        // If there was an error during configure or

```

```

        // realize, the processor will be closed
        if (ce instanceof ControllerClosedEvent)
            setFailed();

        // All controller events, send a notification
        // to the waiting thread in waitForState method.
        if (ce instanceof ControllerEvent) {
            synchronized (getStateLock()) {
                getStateLock().notifyAll();
            }
        }
    }
}

/*****
 * Sample Usage for VideoTransmit class
 *****/

public static void main(String [] args) {
    // We need three parameters to do the transmission
    // For example,
    // java VideoTransmit file:/C:/media/test.mov 129.130.131.132
    //42050

    if (args.length < 3) {
        System.err.println("Usage: VideoTransmit <sourceURL> <destIP>
<destPort>");
        System.exit(-1);
    }

    // Create a video transmit object with the specified params.
    VideoTransmit vt = new VideoTransmit(new MediaLocator(args[0]),
                                           args[1],
                                           args[2]);

    // Start the transmission
    String result = vt.start();
}

```

```

// result will be non-null if there was an error. The return
// value is a String describing the possible error. Print it.
if (result != null) {
    System.err.println("Error : " + result);
    System.exit(0);
}

System.err.println("Start transmission for 60 seconds...");

// Transmit for 60 seconds and then close the processor
// This is a safeguard when using a capture data source
// so that the capture device will be properly released
// before quitting.
// The right thing to do would be to have a GUI with a
// "Stop" button that would call stop on VideoTransmit
try {
    Thread.currentThread().sleep(60000);
} catch (InterruptedException ie) {
}

// Stop the transmission
vt.stop();

System.err.println("...transmission ended.");

System.exit(0);
}
}

```