UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ambiente para Especificação de Aplicações Multimídia com Suporte de Qualidade de Serviço

Wesley Barbosa Thereza

São Carlos
Agosto/2004

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Ambiente para Especificação de Aplicações Multimídia com Suporte de Qualidade de Serviço"

WESLEY BARBOSA THEREZA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Sérgio Donizetti Zorzo
(Orientador - DC/UFSCar)

Prof. Dr. Luis Carlos Trevelin
(DC/UFSCar)

Prof. Dr. Orivaldo de Lira Tavares
(INF/UFES)

São Carlos
Agosto/2004

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

T398ae

Thereza, Wesley Barbosa.
Ambiente para especificação de aplicações multimídia com suporte de qualidade de serviço / Wesley Barbosa Thereza. -- São Carlos : UFSCar, 2006.
100 p.

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

1. Sistemas multimídia. 2. Qualidade de serviço. 3. Linguagem de marcação. 4. HQML. I. Título.

CDD: 006.7 (20ª)

Dedico este trabalho a Deus, meus pais e irmãos
pelo apoio, amor e força de sempre.

Agradecimentos

Inicialmente a minha família – pai (Testa), mãe (Gansa) e irmãos (Capivara e Cabelo) - pelo apoiou desde o início para a realização deste meu sonho.

A todos os amigos que se fazem presentes na minha vida, me apoiando e sendo meu alento para todas as horas, desde as agradáveis até as mais difíceis.

Ao meu orientador Prof. Sérgio Donizetti Zorzo pelo conhecimento, apoio e, principalmente, paciência.

Aos funcionários e professores do Departamento de Computação, em especial a Cristina (Cris), Prof. Luis Carlos Trevelin, Prof. Hélio C. Guardia e Prof. Célio Estavan Moron.

Aos amigos do laboratório de Sistemas Distribuídos e Redes (aquário), pela ajuda e companheirismo: Eugeni (Mestre), Bruno (Amaral), Alessandro (Quadrado), Lúcio (Warta) e Fabiana (Maninha).

Aos amigos e colegas do PPG-CC pela cumplicidade das dores e sucessos: Moacir (Mo-moacir), Eduardo (Escovar), Juliano (Jukinha), Daniel (Luclédio), Fernando (Genta), Taciana, Alexandre (Clô), Darley (Biriguei), Anderson (Dinho), Pablo (Pablito) e Aureliano (Oreia).

Aos grandes amigos que fiz aqui pelos momentos de alegria e extrema felicidade: Mairum (Casão), Vinícius (Baiano), Renato (Renatim), Ricardo (Rar) e Érico (Sabirila). Com certeza, sem vocês eu não teria finalizado esse trabalho.

Em especial ao grande amigo Fernando (Bocudo). Acho que após todo esse tempo de convivência, passei a considerá-lo como um irmão. Irmão sempre presente em todos os momentos vividos ao longo desses anos. Espero que nossa relação de irmandade permaneça por toda nossa vida. Valeu mesmo, cara.

Abstract

This work describes an environment for specification of distributed multimedia applications with demanded Quality of Service (QoS) detailing.

The description of the application is made in a hierarchic way, in the user, application and resource level, adding the demanded QoS parameters and the adaptation rules to be applied for the Quality of Service monitoring/ (re)negotiation.

These rules and QoS parameters will be responsible for the activation of the system adaptation to guarantee its execution in compliance with the Quality of Service negotiated in the specification.

After the specification phase of the distributed multimedia system, the environment generates the documentation in a specification language of system with QoS detailing and its adaptation/ (re)negotiation forms.

This documentation feeds NIST Net emulator, allowing that the results can be analyzed by the verification of the multimedia system behavior and the emulated system.

Resumo

Esse trabalho descreve um ambiente para especificação de aplicações multimídia distribuída com detalhamento da Qualidade de Serviço (Quality of Service – QoS) exigida.

A descrição da aplicação é feita de forma hierárquica, nos níveis de usuário, aplicação e recursos, acrescentando os parâmetros de QoS exigidos e as regras de adaptação a serem aplicadas para o monitoramento/ (re)negociação da Qualidade de Serviço.

Essas regras e parâmetros de QoS serão responsáveis para a ativação da adaptação do sistema de forma a garantir a sua execução em conformidade com a Qualidade de Serviço negociada na especificação.

Após a fase de especificação do sistema multimídia distribuído, o ambiente gera a documentação em uma linguagem de especificação de sistema com detalhamento da QoS e suas formas de adaptação/ (re)negociação.

Essa documentação alimenta o emulador Nist Net, permitindo que os resultados possam ser analisados mediante a verificação do comportamento do sistema multimídia via sistema emulado.

Sumário

Lista de Figuras

Lista de Tabelas

1 - Introdução

O termo multimídia definido em [1] é usado para se referir a múltiplos tipos de mídia, tecnologias, sistemas e aplicações que manipulam várias mídias. Ainda em [1] sistema multimídia é definido como um sistema capaz de manipular pelo menos um tipo de mídia contínua na forma digital, ou mídia estática.

Já sistemas multimídia distribuído são sistemas que lidam com mídias contínuas e as processam de maneira distribuída, isto é, o sistema é formado por elementos que se interagem a fim de fornecer determinado serviço de maneira satisfatória, como vídeo sob demanda e teleconferência.

As infra-estruturas que suportam os sistemas multimídia distribuídos oferecem serviços com reservas de recursos ou garantias que o desempenho não seja prejudicado em virtude de variação nas condições do ambiente.

Num sistema multimídia distribuído os serviços requisitados para a infra-estrutura do ambiente devem ser tratados de forma global. A aplicação requisita os recursos que estão disponíveis no momento da execução, de forma a garantir a melhor qualidade possível. Esses recursos dependem do meio de acesso da aplicação (PDA, desktop, celular, etc.), da rede de comunicação (Ethernet, ATM, wireless, etc.) e dos sistemas de processamento da mídia (codificador, decodificador, prefetcher, etc.). Pode-se utilizar alguns parâmetros mínimos na requisição de tais serviços, como a exigência de uma dada qualidade na transmissão, recepção e execução da mídia, de forma que tais

serviços sejam oferecidos pela infra-estrutura para atender esta requisição mais abrangente.

É nesse contexto que surge a Qualidade de Serviço como o conjunto de características de um sistema necessário para atingir uma determinada funcionalidade por meio da aferição de parâmetros pré-definidos antes de iniciar a execução da aplicação, sendo que a aplicação possui capacidade de renegociação durante sua execução.

As requisições dos serviços necessitam que os recursos e os próprios serviços oferecidos estejam em conformidade para que a infra-estrutura possa atender o que foi determinado na aplicação com a Qualidade de Serviço desejada pelo usuário. Portanto, a QoS torna-se imprescindível nesse contexto, pois ela delimita os padrões mínimos exigidos para os requisitos do usuário, bem como evidencia se o serviço foi bem sucedido ou não.

A descrição da QoS em uma aplicação multimídia pode ser feita numa linguagem de especificação de Qualidade de Serviço. Tais linguagens são responsáveis por "aproximar" o desenvolvedor da aplicação aos recursos do sistema, sem preocupar-se com a complexidade da implementação. Além disso, facilita o entendimento dos meios envolvidos nas transmissões de dados das aplicações multimídia distribuídas.

As linguagens de especificação, tanto as de especificação de sistemas, como as de especificação de QoS, surgiram devido às diferenças existentes entre os sistemas operacionais, as redes de comunicação, as preferências dos usuários, as aplicações e, ainda, ao dinamismo no uso dos recursos de um sistema. Enquanto as linguagens de especificação de sistemas agregam características capazes de descrever o sistema multimídia distribuído como um todo, desde os requisitos físicos do sistema até parâmetros de QoS, as linguagens de especificação de QoS descrevem somente parâmetros ligados a QoS.

O objetivo desse trabalho é construir um ambiente para especificação de aplicações multimídia distribuída com detalhamento da Qualidade de Serviço exigida, a

fim de analisar os resultados dessa especificação por meio de um sistema multimídia distribuído emulado utilizando o emulador Nist Net.

O ambiente também oferece a possibilidade de inserção de parâmetros de QoS que funcionam como reguladores da Qualidade de Serviço, caso seja necessária a adaptação e a (re)negociação da QoS envolvida. Com isso, pretende-se abordar principalmente esses mecanismos de adaptação e a possível renegociação dos parâmetros, para comprovar o ineditismo desses recursos por meio de um estudo de caso, ou seja, uma aplicação de vídeo sob demanda.

O desenvolvimento deste trabalho iniciou com um levantamento bibliográfico sobre QoS, linguagens de especificação de QoS para sistemas multimídia distribuídos e ambientes de descrição de sistemas multimídia, objetivando a especificação do ambiente proposto. Esse ambiente de especificação, além de permitir o detalhamento do Sistema Multimídia com relevância aos aspectos de qualidade de serviço, permite a implementação de novos recursos, como adaptação e (re)negociação. O ambiente implementado é uma extensão da proposta QoSTalk [2] e da linguagem de especificação de QoS HQML, com as funcionalidades descritas acima. Os testes do ambiente foram realizados com uma aplicação multimídia distribuída de vídeo sob demanda, que foi emulado no Nist Net [50] . Os testes evidenciaram a viabilidade do ambiente com recursos implementados e emulados, com ganhos em tempo de desenvolvimento e na clareza do processo.

O conteúdo deste trabalho está organizado da seguinte forma:

No capítulo 2 é apresentada a conceituação de Qualidade de Serviço, enumerando as prerrogativas inerentes ao escopo de sistemas multimídia distribuídos adotados no âmbito deste trabalho.

No capítulo 3 são apresentadas linguagens de especificação de QoS para aplicações multimídia distribuídas, bem como a análise comparativa das linguagens levando-se em consideração aspectos de reusabilidade, expressividade, independência, declaratividade e extensibilidade.

O capítulo 4 descreve a linguagem de especificação de sistemas proposta e utilizada neste trabalho, baseada na linguagem HQML [2]. Apresenta ainda o modelo básico de aplicação e suas características e vantagens na utilização da mesma na descrição de um sistema multimídia distribuída com garantia de QoS.

No capítulo 5 é apresentado o Ambiente para Especificação de Aplicações Multimídia com Suporte de Qualidade de Serviço proposto. São descritos os aspectos da arquitetura e os recursos do sistema.

Um estudo de caso é apresentado no capítulo 6, com o detalhamento de todos os passos utilizados na modelagem de uma aplicação multimídia e sua utilização em sistema emulado, bem como os resultados obtidos.

No capítulo 7 são apresentadas as conclusões deste trabalho e propostas futuras para extensão.

2 - Qualidade de Serviço

2.1 - Definição

Apesar de Qualidade de Serviço (QoS) não ser um conceito novo, atualmente este termo tem sido muito abordado em pesquisas que se referem a utilização de aplicações multimídia que precisam de alguma garantia na efetivação do serviço desejado.

O primeiro problema existente ao lidar com QoS é a definição do termo, haja vista que vários órgãos internacionais de padronização utilizam uma definição própria como escopo para determinar a evolução dos estudos que envolvem QoS. Por isso, não existe uma definição formal ou comum do que é QoS. Um grande número de definições pode ser encontrado na literatura, nas quais a noção de QoS originalmente serve para descrever certas características técnicas da transmissão dos dados, principalmente dados que não são dependentes do tempo, como dados de mídia discretos (texto e figuras).

Neste trabalho a definição que melhor descreve as aspirações buscadas, baseia-se em [3] e [4], chegando à conclusão de que QoS é o conjunto de características de um sistema (aplicação) necessário para atingir uma determinada funcionalidade através da aferição de parâmetros pré-definidos antes de iniciar a execução da aplicação, sendo que a aplicação possui capacidade de renegociação durante sua execução.

Abaixo estão algumas definições no nível de comunicação utilizadas pelos vários órgãos de padronização do mundo das telecomunicações, padrões internacionais e padrões Internet [5]:

- O International Telecommunication Union (ITU), refere-se a QoS como "Um conjunto de requisitos de qualidade de comportamento coletivo de um ou mais objetos". Alguns parâmetros de QoS descrevem rapidez e confiabilidade na transmissão dos dados, por exemplo, vazão, atraso e taxa de erro;
- A ATM Lexicon define em [6] QoS como "Um termo que se refere a um conjunto de parâmetros de desempenho ATM que caracteriza o tráfego sobre uma dada conexão virtual". Parâmetros de QoS aplicam-se principalmente às camadas do protocolo nos níveis mais baixos, não sendo diretamente observáveis ou verificáveis pela aplicação. Esses parâmetros incluem taxa de células perdidas, taxa de células erradas, taxa de falhas durante a inserção de células na rede, variação do atraso das células, atraso na transferência da célula, e média de atraso na transferência das células;
- Já em [7], os autores definem que Qualidade de Serviço é um requisito que uma classe de aplicações possui. Isso exige que uma série de parâmetros esteja dentro de uma faixa de valores bem definidos, para o seu perfeito funcionamento.

Além dessas definições, várias outras podem ser encontradas na literatura. Uma definição mais geral de QoS para aplicações que se comunicam em tempo real dada por Vogel et al. [8], descreve Qualidade de Serviço como a representação do conjunto daquelas características qualitativas e quantitativas de um sistema multimídia distribuído necessárias para obter funcionalidades requeridas pela aplicação. Funcionalidades incluem a apresentação de dados multimídia para o usuário, bem como sua satisfação.

Outras definições de QoS incluem: um conjunto de parâmetros que define as propriedades dos *streams*[1] [9]; uma descrição quantitativa de quais serviços oferecidos pelo sistema satisfaz as necessidades da aplicação expressa como um conjunto de pares de parâmetros [10].

[1] Streams: fluxos de dados multimídia.

2.2 - Mapeamento de QoS

Dobson em [11] trata QoS em sistemas multimídia distribuídos, primeiramente, a partir do desempenho de toda a rede e, posteriormente, a serviços de características não-funcionais. Já Bernet et al. em [12] descreve um framework para aplicações que requerem controle de QoS. Nesse trabalho foi utilizada uma modelagem apresentada por Nahrstedt em [13], em que os parâmetros de QoS podem ser divididos conceitualmente em camadas, que podem ser observadas na Figura 1. A Camada de Usuário descreve a Qualidade de Serviço do usuário referentes aos anseios, bem como a Camada de Aplicação descreve a QoS para a aplicação, porém a Camada de Recurso descreve QoS nos níveis do Sistema Operacional e da rede. Os parâmetros das QoS de Usuário, Aplicação, Sistema e Rede são apresentados nos itens a seguir.

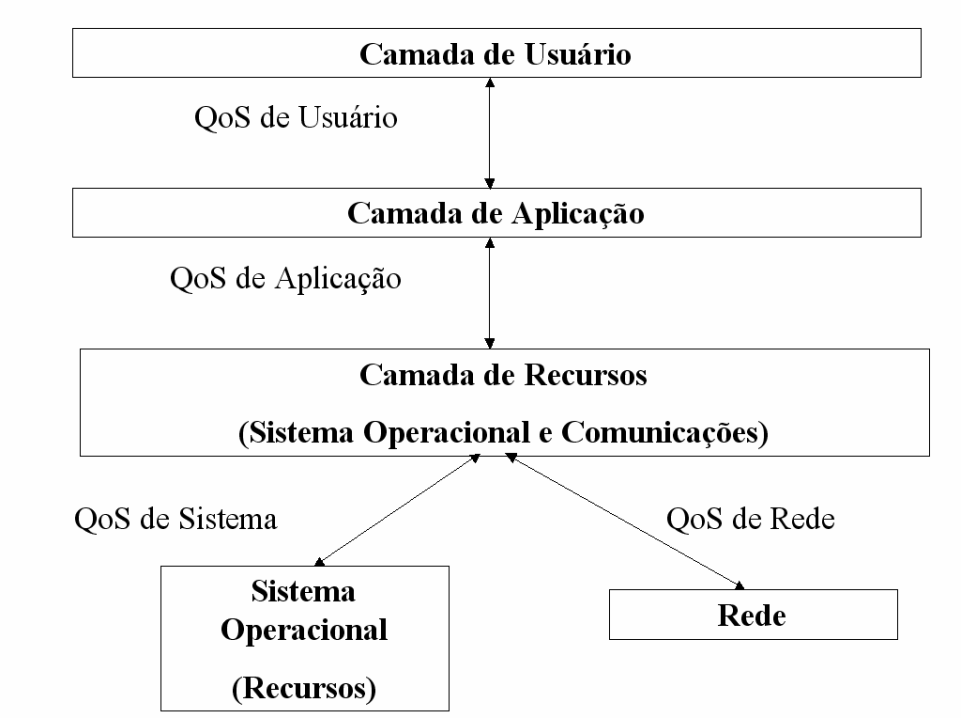


Figura 1: Modelo conceitual de QoS.

2.2.1 - QoS de Usuário

Os parâmetros de qualidade relacionados diretamente aos usuários são necessariamente menos técnicos e mais subjetivos, normalmente mensurados de forma qualitativa. Uma interface apropriada deve ser oferecida para facilitar a escolha dos parâmetros [13].

A essência dessa camada é ocultar, tanto quanto possível, os parâmetros de QoS inerentes ao sistema (freqüentemente sem significado para o usuário). Para isso, devem-se apresentar exemplos que permitem realizar escolhas como tamanhos de imagem e resolução diferentes, ou áudio com qualidade de CD ou telefone. Em um sistema, cada uma das escolhas anteriores impactam nas exigências de QoS de rede. Por exemplo, a seleção de uma imagem de alta resolução pode resultar em um aumento de largura de banda capaz de suportar tais imagens. Além disso, aspectos como tamanho da janela (grande, média, pequena), preço do modelo (taxa de flat, carga por transmissão de bytes) e variação de preço (alta, média, baixa) também são parâmetros relevantes nessa camada.

Uma outra possibilidade é permitir que o usuário escolha a QoS como alta, média, baixa ou qualquer dentro de uma classe de aplicação e essa escolha seja refletida nas camadas inferiores.

2.2.2 - QoS de Aplicação

Nessa camada são descritos os requisitos para a aplicação em termos de qualidade da mídia e relacionamento entre mídias (sincronização intermídia e intramídia). Escolhas realizadas pelo usuário final, na QoS na camada de usuário, são mapeadas em um conjunto de parâmetros de aplicação, os quais devem atender seus anseios.

Nesse nível, os parâmetros são associados a quadros de vídeo (tamanho do quadro, taxa de quadros/segundo), ou amostras (taxa de amostragem, bits/amostragem), etc. Como exemplo, pode ser citado um vídeo com quadros de 640

pixels por 480 pixels com taxa de 30 quadros por segundo. Além disso, parâmetros como tamanho do frame do vídeo, taxa de frame do vídeo, resolução de áudio/imagem e regras de adaptação são parâmetros relevantes nessa camada. Ainda deve ser ressaltado que parâmetros como atraso e variação do atraso, normalmente tratados como parâmetros de rede, podem ser analisados nessa camada, haja vista a necessidade da aplicação de ter garantias quanto aos valores desses parâmetros. Esses parâmetros são mapeados para a Camada de Recurso a fim de serem alocados recursos físicos necessários para a garantia de que os valores dos parâmetros serão alcançados no nível da aplicação.

2.2.3 - QoS de Sistema (Software e Hardware)

Os parâmetros dessa camada representam características de cada host presente no sistema, as quais podem ser especificadas em termos de critérios quantitativos e qualitativos, sendo os quantitativos representados pelo hardware e os qualitativos tratados pelo sistema operacional de cada máquina. Os critérios quantitativos são aqueles que podem ser avaliados em termos de medidas concretas, como bits por segundo, tempo de processamento de uma tarefa, e tamanho da unidade de dados. Os critérios qualitativos especificam os serviços esperados, como sincronização, mecanismos de escalonamento e recuperação de erros.

Os parâmetros, nesse nível, podem ter um grande impacto na qualidade percebida pelo usuário. Por exemplo, a velocidade do barramento, memória disponível e tempo de CPU podem limitar a taxa de quadros da apresentação de um vídeo, bem como uma tela que apresenta imagens em preto e branco não pode mostrar imagens coloridas.

Em se tratando de imagens, a forma de armazenamento ocorre calculando-se o número de cada pixel (H) em cada linha, vezes o número de linhas da imagem (V) e a profundidade do pixel P (bits por pixel), divididos por 8 [14]:

Armazenamento (em Bytes, MBytes, GBytes) = HVP/8

Por exemplo, se uma imagem tem 480 linhas, 600 pixels em cada linha e uma profundidade de pixel de 24 bits, é necessário 864.000 bytes para representá-la.

Para uma única imagem não é preciso dimensionar o tempo de armazenamento, porém se há um tempo limite para a transmissão de cada imagem, a largura de banda pode ser calculada baseado no armazenamento da mídia. Por exemplo, se a imagem de 864.000 bytes deve ser transmitida com um tempo de 2 segundos, então a largura de banda necessária é de 3.456 Mbits/s. Em muitas aplicações, imagens podem ser mostradas em sincronia com uma mídia contínua, como o áudio. Neste caso, a transmissão de imagens impõe restrições de tempo e largura de banda.

Tanto o áudio quanto o vídeo são mídias contínuas. Para calcular a taxa de bits transmitidas em um segundo por um arquivo de áudio deve-se basear na taxa de amostras e número de bits por amostra. A taxa de bits para arquivos de vídeo pode ser calculada da mesma forma que no áudio, porém normalmente é calculada a partir da quantidade de dados em cada imagem (quadro ou frame) vezes o número de quadros por segundo. O número resultante desse cálculo especifica tanto a taxa de bits necessitada pelo canal de transmissão quanto a taxa de transferência do dispositivo de armazenamento.

A Tabela 1 [14] apresenta os requisitos de largura de banda de mídias contínuas (áudio e vídeo) de diferentes qualidades. Já a Figura 2 [14] mostra os requisitos de armazenamento para mídias estáticas e mídias contínuas de diferentes durações. Obviamente que os dados expostos na Figura 2 dependem de fatores como tamanho da imagem e as representações de cores e não são consideradas mídias comprimidas.

Aplicações	Taxa de Dados (kbit/s)
CD-Áudio	1.411,2
DAT	1.536
Telefone Digital	64
Televisão-Qualidade do Vídeo	216.000
VHS-Qualidade do Vídeo	54.000
HDTV	864.000

Tabela 1: Taxa de bits de algumas aplicações de áudio e vídeo. [Willrich, 2004]

Na Tabela 1 e na Figura 2 é possível perceber que as mídias digitais necessitam de grande quantidade de dados para serem representadas, além de farta largura de banda para a transmissão. Por meio dos dados descritos tanto na Tabela 1, como na Figura 2, pode-se calcular que em um mídia de 1.5 horas de áudio com qualidade de CD, ou 36 segundos de vídeo com qualidade de TV precisam de 1 GB para ser armazenada. Essa quantidade de dados precisaria de 800 segundos para ser transferida com uma taxa de 10 Mbits/s.

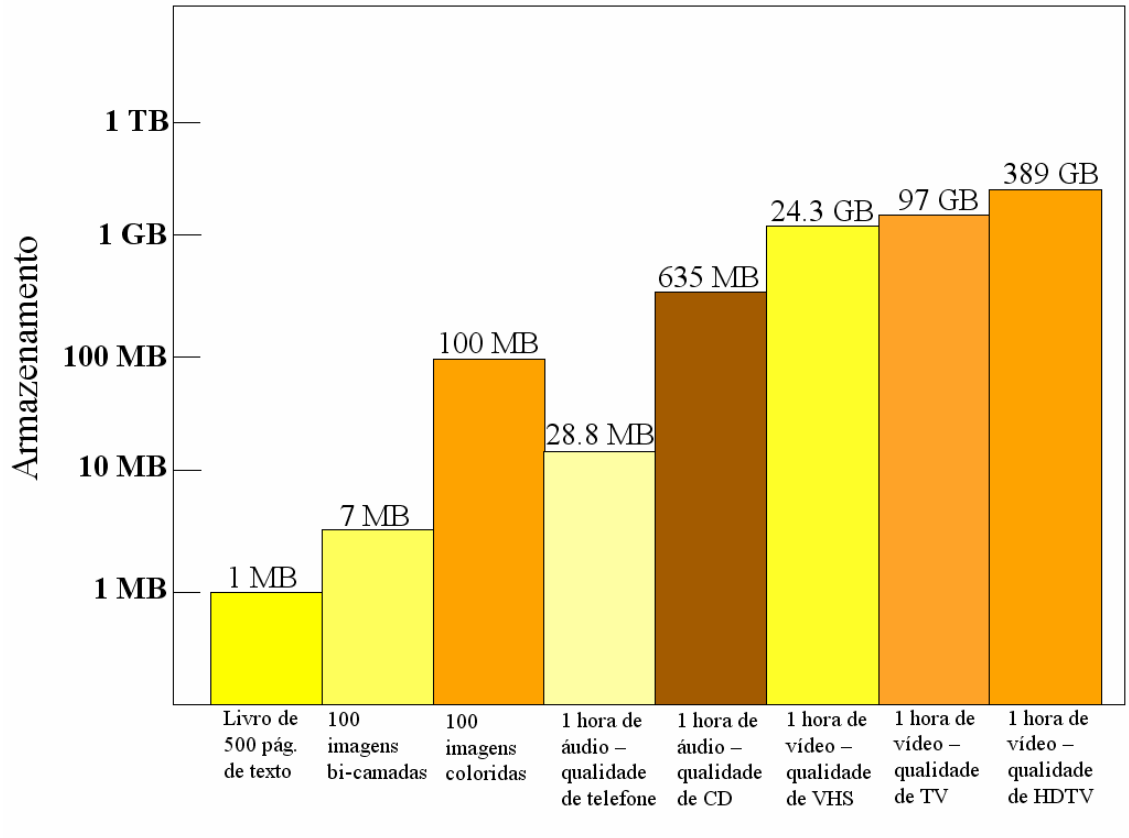


Figura 2: Requisitos de armazenamento para algumas mídias. [Willrich, 2004]

2.2.4 - QoS de Rede

Os requisitos dos serviços de rede são abordados nessa camada, por exemplo, o desempenho, associados a propriedades de pacotes ou bits, tais como, atraso de pacotes, ou taxa de bits. Nesse nível, parâmetros fim-a-fim (delay, jitter e perda de pacotes) são especificados.

Para aplicações multimídia, por exemplo, a rede deve satisfazer os requisitos de transmissão de dados multimídia. O tráfego de mídia discreta (transferência de arquivos ou recuperação de imagens) requer serviços livres de erros, mas é tolerante a atrasos. O tráfego de mídia contínua requer tempo real e transmissão de alta velocidade para um bom desempenho de aplicações. Essa mídia é sensitiva a atrasos e suas variações, mas é tolerante a perdas ocasionais de pacotes. A rede deve suportar os requisitos especificados pela aplicação.

Áudio e vídeo digitais são mídias contínuas dependentes do tempo, e por isso são chamadas mídias dinâmicas ou isócronas. Isso significa que para ter uma boa qualidade quando executados o áudio e o vídeo devem ter taxas de amostragens com intervalos regulares. Por exemplo, se uma "parte" do áudio tem taxa de amostragem de 8kHz, o mesmo deve ser executado com 8.000 amostras por segundo. Essa dependência em relação ao tempo das mídias contínuas implica diretamente na existência de dois parâmetros: o atraso (delay) e a variação do atraso (jitter).

O atraso, comumente tratado como atraso fim-a-fim, é a soma de todos os atrasos em todos os componentes do sistema multimídia, incluindo acesso ao disco, ADC[2], codificação, tempo de processamento, acesso à rede, transmissão na rede, bufferização, decoficação e DAC[3]. Um atraso aceitável é algo muito subjetivo e também dependente da aplicação. Por exemplo, de acordo com Gopal et al. [15] e Hehmann et al. [16] o atraso em aplicações do tipo conversacional [17] deve ser abaixo de 300 ms, porém para aplicações do tipo apresentacional [17] o atraso de alguns segundos é aceitável.

A variação do atraso, ou jitter é a variação do intervalo de tempo entre chegadas de pacotes no receptor. Para uma transmissão de mídia contínua sem problemas, a variação do atraso deve ser mantida bem pequena. Por exemplo, de acordo com Hehmann et al. [16] para transmissões de áudio com qualidade de telefone e vídeo com qualidade de televisão, a variação do atraso deve ser menor do que 10 ms, porém para aplicações de áudio de alta qualidade esse valor cai para menos de 1 ms.

Diferentes dos dados alfanuméricos, onde perdas e erros na transmissão são na sua grande maioria intoleráveis. Erros ou perdas em dados de áudio, vídeo e imagens podem ser tolerados, pois estas perdas e erros de bits não são desastrosos e geralmente não são percebidos pelo usuário.

Para voz, tolera-se uma taxa de erros de bit de 10^{-2}. Para imagens e vídeos pode-se tolerar uma taxa de bits de 10^{-4} a 10^{-6}. Outro parâmetro que mede o erro é a taxa de perda de pacotes. Os requisitos de taxa de perdas de pacote são mais fortes

[2] ADC (Analog to Digital Converter): conversão de analógico para digital. [18]
[3] DAC (Digital to Analog Convertor): conversão de digital para analógico. [19]

que a de erros de bit, porque uma perda de pacote pode afetar a decodificação de uma imagem, por exemplo. Quando técnicas de compressão são utilizadas a taxa de erro de bits deve ser pequena, pois um erro de bit pode causar um erro de descompactação de muitos bits. Técnicas de encobrimento de erros podem ser empregadas para aumentar a qualidade de áudio e vídeo.

2.3 - Abordagem

Considerando QoS como sendo o conjunto de características que um sistema necessita para que a sua funcionalidade possa ser aferida por parâmetros predefinidos, apresenta-se a delimitação do processo de execução do sistema que se dá da forma mostrada na Figura 3: o processamento de QoS num sistema distribuído começa com o estabelecimento dos parâmetros exigidos pelo usuário. Como definido na seção 2.2.1, os parâmetros são relevantes ao que o usuário julga necessário para satisfazer os seus anseios. Esses parâmetros são mapeados e negociados entre os componentes do sistema como um todo, assegurando que todos podem atingir um nível de QoS aceitável. Nesse ponto, os componentes verificam os parâmetros definidos na Camada de Aplicação, pois, deve haver conformidade entre os requisitos escolhidos pelo usuário na camada superior com os requisitos da aplicação. Recursos são, então, alocados e monitorados, havendo possibilidade de renegociação caso as condições do sistema se alterem. Em havendo a necessidade de uma renegociação, novos parâmetros de QoS são remapeados para os componentes a fim de que possa ser garantida a QoS aceitável pelo sistema. A alocação de recursos, bem como a (re)negociação baseada em regras de adaptação, acontecem mediante parâmetros mapeados para a Camada de Recurso.

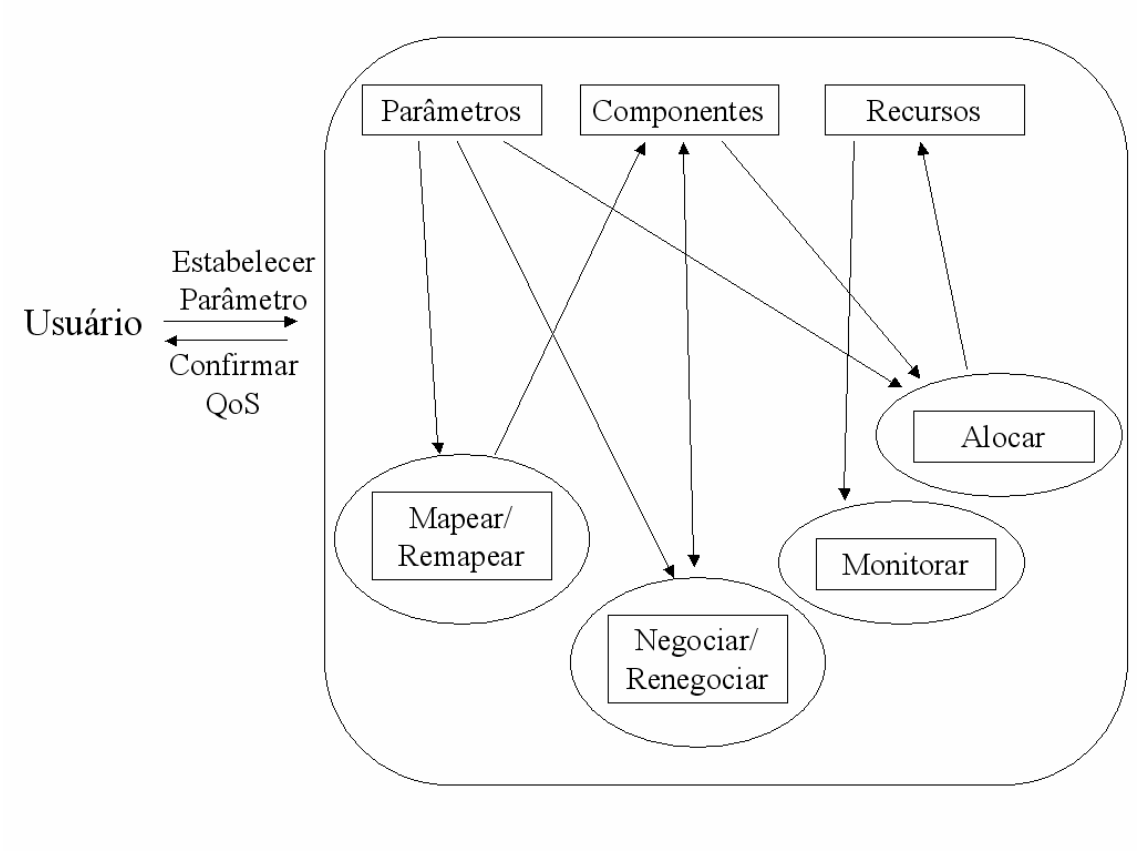


Figura 3: Esquema funcional do controle da Qualidade de Serviço.

Os parâmetros de QoS devem ser escolhidos de acordo com o ambiente usado, pois devido às suas características particulares, redes fixas e redes móveis sem fio, por exemplo, necessitam negociar parâmetros diferentes. Os parâmetros de QoS do ponto de vista da aplicação são mais subjetivos pois tratam, por exemplo, se a imagem em uma videoconferência deve ser em cores ou em preto-e-branco, se o som deve ter qualidade de telefone ou de CD, se pode haver ou não interrupção da comunicação durante um *handoff*[4], entre outras coisas. Deste modo, esse trabalho apresenta um mecanismo que traduza tais desejos em parâmetros de mais baixo nível os quais possam ser negociados pelas entidades de rede, tais como taxa de transmissão ou atraso. Portanto, QoS é algo que deve ser tratado por todo o sistema [20].

[4] Handoff: Em sistemas móveis celulares, o processo de transferir uma chamada em andamento de uma célula transmissora e receptora com uma freqüência definida para uma outra célula transmissora e receptora usando uma freqüência diferente [21].

Após a escolha dos valores dos parâmetros de QoS pela aplicação e sua posterior tradução em parâmetros que o sistema possa compreender, deve-se efetivamente iniciar a negociação com os elementos do sistema envolvidos para verificar se é possível prover a qualidade desejada.

Se a negociação com os elementos do sistema for bem sucedida, é preciso que os recursos necessários para que se garantam a QoS sejam reservados ao longo do caminho da comunicação. Durante a comunicação deve-se monitorar também a QoS e, caso seja violada, tomar as providências necessárias. Além disso, ao longo da comunicação a aplicação deve poder renegociar parâmetros de QoS.

Como a QoS é garantida às aplicações por meio de reserva de recursos ao longo da rede, como largura de banda, *buffers* nos roteadores, ciclos de CPU, entre outros, torna-se importante que não haja desperdício de recursos, pois deste modo, pode-se atender a mais usuários simultaneamente. Esta racionalidade no uso dos recursos também é importante devido ao fato de que o usuário estará eventualmente pagando pela sua utilização e, portanto, deve-se procurar minimizar os custos.

Embora existam muitas soluções para a utilização de QoS em sistemas multimídia distribuídos, os desenvolvedores de aplicação ainda não conseguiram criar aplicações multimídia com QoS embutida, especialmente para aplicações Web [22]. A razão está situada no fato de que não existe uma linguagem de especificação de QoS universal. Mesmo existindo as mais diversas linguagens de especificação de QoS, elas ainda estão muito ligadas com uma linguagem de programação específica ou não podem ser estendidas para alcançar rapidamente os novos serviços de QoS. Dessa forma, é necessário que haja uma análise das linguagens de especificação de QoS e das necessidades do desenvolvedor da aplicação, a fim de que todos os requisitos do usuário possam ser expressos na especificação de QoS. O capítulo a seguir apresenta uma análise comparativa das linguagens de especificação de QoS.

3 - Linguagens de Especificação de Qualidade de Serviço para Aplicações Multimídia Distribuídas

Tipicamente, o gerenciamento de aplicações multimídia distribuídas requer que, antes de iniciar a execução, uma aplicação determine seus requisitos de QoS por meio de parâmetros como taxa de apresentação, qualidade de imagem e som. Esses requisitos são então traduzidos em um conjunto de recursos do sistema (tais com largura de banda de transmissão, memória ou ciclos de CPU por segundo) alocados para a aplicação. Um processo de negociação da QoS, juntamente com uma infra-estrutura de gerenciamento, garante o fornecimento da QoS requerida durante a execução da aplicação, por meio de reserva de recursos.

A QoS envolve um grande número de propriedades, como características de desempenho, disponibilidade, capacidade de obter resposta em um tempo determinado, segurança e versatilidade, além dos aspectos específicos da aplicação. Em relação às especificações de QoS [23], tem-se:

- devem permitir descrições de parâmetros de QoS quantitativos e qualitativos, assim como regras de adaptação;
- devem ser informativas, isto é, devem especificar somente o que é requisitado pelo usuário, porém não como a requisição deve ser executada pela aplicação;

- precisam ser acompanhadas por um processo de mapeamento e compilação para poder mapear a especificação de QoS para dar suporte a políticas e a mecanismos do sistema.

Devido às diferenças existentes entre os sistemas operacionais, as redes de comunicação, as preferências dos usuários, as aplicações e, ainda, ao dinamismo no uso dos recursos de um sistema, torna-se imprescindível especificar propriamente as necessidades e políticas de adaptação da QoS para a entrega e o processamento multimídia.

Para satisfazer essa tarefa complexa, muitas linguagens de especificação de QoS surgiram para suprir essas necessidades. Com tantas especificações de QoS existentes, que passam por várias camadas de QoS e paradigmas de linguagens, há uma forte necessidade de categorizá-las e organizá-las para que se obtenha um amplo e criterioso entendimento sobre esta importante área.

A metodologia para analisar o conjunto de linguagens de especificação de QoS pode empregar a divisão e conquista, como estudado por Jin e Nahrstedt [23], em que foi particionado o conjunto em paradigmas de acordo com as propriedades específicas e avaliada cada paradigma separadamente, assim como foram estudadas as relações entre esses paradigmas.

A QoS em sistemas multimídia deve ser assegurada não somente na camada de rede, mas também nos sistemas finais, como por exemplo os sistemas operacionais e dispositivos periféricos. Por essa razão, a especificação de QoS foi introduzida para sistemas finais e aplicações [24]. Baseado nesse desenvolvimento, as linguagens de especificação de QoS podem ser divididas primeiramente em camadas de acordo com o sistema final a que elas pertencem [25], [26], e então em paradigmas baseados em suas propriedades.

3.1 - Particionamento das Linguagens de Especificação de QoS em Camadas

O particionamento em camada é necessário porque especificações em camadas diferentes apresentam muitas características diferentes. São consideradas 3 camadas: Camada de Usuário, Camada de Aplicação e Camada de Recursos.

Como um simples exemplo para ilustrar o modelo de especificação de 3 camadas, pode-se considerar uma aplicação de Vídeo Sob Demanda em que o usuário quer um vídeo localizado num servidor remoto. Nestes casos, o usuário pode especificar, geralmente sobre uma Interface Gráfica (GUI), a qualidade de mídia desejada para o serviço de Vídeo Sob Demanda entre o conjunto de escolhas, como, alta, média e baixa qualidade, baseada na percepção humana. Esta especificação é então passada para a camada da aplicação para derivar especificações de qualidade de mídia mais concreta, como taxa de quadro do vídeo e resolução imagem/áudio. Essas são especificações inerentes à aplicação, mas com descrições independentes de plataforma e hardware.

Camadas de QoS	Parâmetros/Questões de QoS
Camada de Usuário *(critérios subjetivos)*	- Qualidade de mídia percebida (excelente, bom, regular, ruim) - Tamanho da janela (grande, média, pequena) - Preço do modelo (taxa de *flat*, carga por transmissão de bytes) - Variação de preço (alta, média, baixa)
Camada da Aplicação *(independente de hardware e plataforma)*	- Atributos de QoS específicos à aplicação (tamanho do frame do vídeo, taxa de frame do vídeo, resolução de áudio/imagem, sincronização inter/intrastream) - Regras de adaptação
Camada de Recurso *(dependente de hardware e plataforma)*	- Questões quantitativas (vazão, atraso, jitter, tamanho da memória, tempo de requisição do recurso) - Questões qualitativas (escalonamento do Sistema Operacional, estilo da reserva do recurso, mecanismos de detecção/recuperação de perda) - Regras de adaptação

Tabela 2: QoS para as três camadas: Usuário, Aplicação e Recurso.

Sistema operacional e serviços de comunicação são especificados quantitativa e qualitativamente, onde descrições quantitativas podem incluir vazão, atraso, jitter, tamanho de buffer, e sincronização da distorção de imagem, ao passo que descrições qualitativas podem incluir políticas de escalonamento de CPU e mecanismos de recuperação de erros de transmissão. Um resumo das camadas de QoS e alguns de seus parâmetros são apresentados na Tabela 2 [23].

3.2 - Particionamento das Linguagens de Especificação de QoS em Paradigmas

As linguagens de especificação de QoS podem ser classificadas de acordo com suas propriedades e com as camadas de especificação da Modelagem. Na Camada de Usuário não há linguagens de especificação representativas, uma vez que seus requisitos são de caráter cognitivo. As especificações da Camada de Aplicação divididas quanto aos seus paradigmas (baseada em script, baseada em parâmetro, orientada a processo, baseada em controle lógico, orientada a aspectos e baseada em markup), enquanto as especificações da camada de recursos baseiam-se nas suas granularidades (grossa e fina). De acordo com Jin e Nahrstedt as principais linguagens de especificação, de acordo com o seu paradigma são descritas a seguir.

3.2.1 - Paradigma baseado em script

As linguagens baseadas em script são mais abstratas do que as linguagens imperativas, assim podem ser usadas para especificar objetos em um nível elevado, sem ter que revelar muitos detalhes. Roscoe e Bowen [27] apresentam uma técnica que adiciona suporte de QoS para o Windows NT sem modificar as aplicações ou os componentes do sistema operacional. Nessa solução há a adição de uma parte de software, chamada agente do protocolo, na pilha do protocolo winsock. O agente do

protocolo intercepta as chamadas feitas pela aplicação à rede, e então contata o daemon[5] de política, O detalhamento das políticas são expressas pela linguagem Safe-Tcl [28], para aprender como marcar os pacotes da rede de acordo com a descrição dada pelo script de QoS. Nesse exemplo, a linguagem Safe-Tcl foi usada somente para instruir o agente do protocolo como marcar os pacotes da rede, e também mostra como especificações podem ser escritas separadamente das aplicações, mostrando assim uma boa independência.

3.2.2 - Paradigma baseado em parâmetro

Muitas das aplicações com QoS embutidas atuais, ao especificar a Camada de Aplicação de QoS, usam este paradigma. Neste paradigma, os desenvolvedores de aplicação definem estruturas de dados para expressar e declarar parâmetros qualitativos e quantitativos, sem criar uma nova linguagem de especificação de QoS. Um exemplo de linguagem com paradigma baseado em parâmetro é QoS-A [29], em que é usada uma Arquitetura de Qualidade de Serviço (QoS-A) para tentar garantir a QoS tanto nos sistemas finais quanto nas redes de uma maneira uniforme. Por exemplo, em uma especificação de parâmetro de QoS entre duas partes se comunicando, deve-se definir um contrato de serviço que inclui diversos aspectos: especificação do fluxo, comprometimento de QoS, adaptação de QoS, manutenção de QoS, tipo de reserva e custo. O contrato de serviço é implementado como em uma estrutura da linguagem C, e cada cláusula é representada como uma outra estrutura. Dessa forma, o conjunto de construções de QoS torna-se bastante rico, fazendo a API boa em termos de expressividade, sendo possível acrescentar novas construções de QoS sem dificuldade. QoS-A permite também que os contratos sejam desenvolvidos independentemente das aplicações. Como os parâmetros de QoS e as ações são especificados como estruturas, QoS-A não fornecem facilidades especiais para reuso da especificação.

[5] Daemon: processo executado em segundo plano que realiza operações em períodos predefinidos ou em resposta a determinados eventos.

3.2.3 - Paradigma orientado a processo

O paradigma orientado a processo supõe o modelo, em que os processos, como unidades de execução, se comunicam e se sincronizam um com o outro pela passagem de mensagens, ou portas de comunicação. A especificação de QoS orientada a processo permite associar QoS com as portas finais de comunicação, e também expressar a negociação de restrição de QoS e a monitoração de QoS entre duas portas. Um exemplo de linguagem de especificação de QoS baseada em processo é QuAL [30]. QuAL é uma extensão de Concert/C, e suas construções fornecem meios para a especificação e a negociação de restrições de QoS, especificação de gerenciadores de violação de QoS, e configuração da monitoração de QoS. QuAL tem uma boa expressividade e é fácil para se desenvolver novas construções de QoS quando necessário. Entretanto, as especificações escritas em QuAL são espalhadas através do código funcional, tornando as duas partes (código funcional e especificação de QoS) difíceis de se fazer manutenção. Além disso, a linguagem é mais instrutiva do que declarativa, que deve ser uma característica natural de toda a linguagem de especificação.

3.2.4 - Paradigma baseado em controle lógico

As linguagens baseadas em controle lógico são adotadas em sistemas adaptáveis para especificações de QoS de políticas e do controle de fluxo adaptáveis [31]. Alguns sistemas usam técnicas do controle adaptável tais como o controlador de PID (Proporcional-Integral-Derivative) para especificar e controlar a granularidade fina quando escalonar fluxos ou tarefas. Outros sistemas usam técnicas de controle fuzzy para especificar e ajudar na adaptação de QoS. O controle fuzzy permite que as aplicações expressem políticas e preferências de adaptação de QoS na forma de regras e funções. As regras são especificadas no formato “if-then”. Um exemplo de regra de adaptação nesse formato é “if cpu is very_high and rate is very_low then rate_demand is compress” que diz para o sistema comprimir os dados porque a largura de banda é

muito baixa, mas há uma grande quantidade de cpu disponível. Uma limitação do controle fuzzy é que esse tipo de linguagem tem a intenção de especificar somente ações, mas não outras propriedades de QoS. Assim este tipo de linguagem é pobre em termos de expressividade e, em geral, não possui subsídios suficientes para expressar as necessidades de aplicações multimídia.

3.2.5 - Paradigma baseado em markup

Neste paradigma encontram-se as linguagens baseadas em markup que é um mecanismo para identificar estruturas em documentos, sendo que especificações XML [32] definem uma maneira padrão de adicionar markup a documentos. Exemplos de linguagens que utilizam este paradigma são XQoS [33] e HQML [2]. As características da HQML são expostas no capítulo 4 deste trabalho, porém, faz-se relevante citar que a HQML é expressiva, declarativa e independente, no entanto, não possui nenhuma construção especial que facilite a extensão e o reuso de especificações existentes.

3.2.6 - Paradigma orientado a aspecto

Este paradigma segue a Programação Orientada a Aspecto (AOP) [34], porque as tarefas relacionadas a QoS são exemplos dos aspectos deste paradigma. Os aspectos não são unidades da decomposição funcional do sistema; preferencialmente, eles são as propriedades que afetam o desempenho ou a semântica dos componentes em maneiras sistemáticas. Usando linguagens de programação tradicionais, as implementações de tais aspectos resultariam em um código complicado com o código relacionado a aspecto espalhado por todo o programa. A tecnologia de Programação Orientada a Aspecto foi desenvolvida para suportar a abstração e a composição tanto de componentes funcionais como de aspectos. Usando AOP, uma aplicação pode ser decomposta em componentes funcionais e aspectos, onde aspectos diferentes podem ser

programados em linguagens também diferentes, mas que sejam apropriadas às tarefas, sendo que há ainda um processador de linguagem especial, chamado aspect weaver, que coordena a composição do código, fazendo com que todo o código seja unificado para produzir uma única aplicação executável. O paradigma orientado a aspecto pode ser encontrado no framework Quality Object (QuO) [35], [36], [37]. QuO suporta QoS na Camada de Recurso, que é mapeado para a camada de objeto do CORBA, onde as implementações do objeto distribuído possibilitam acesso às propriedades do sistema CORBA ORB. QuO estende a linguagem de descrição de interface do CORBA com a descrição de QoS denominada de QDL, permitindo especificações de estados de QoS, de recursos do sistema e de mecanismos possíveis a fim de medir e fornecer QoS, e ainda com comportamento para adaptar-se às mudanças em QoS. QDL é bom na maioria dos critérios de avaliação, sendo expressiva, declarativa, independente e extensível.

3.2.7 - Paradigma orientado a objeto

As linguagens deste paradima baseiam-se nos conceitos das linguagens orientadas a objetos para proverem a herança de especificações como exemplo de reuso de código. Um exemplo de linguagem que forneça esses conceitos é QML (QoS Modeling Language) [38]. QML oferece três mecanismos principais de abstração de especificação de QoS: tipo do contrato, contrato, e perfil; em que o tipo do contrato define as dimensões que podem ser usadas para caracterizar um aspecto particular de QoS, os contratos são instâncias de tipos do contrato, e os perfis associam contratos com as interfaces e as operações. Além disso, é boa em termos de independência e extensibilidade. Entretanto, uma limitação de QML é que para cada tipo de contrato que um perfil associa, no máximo um contrato pode ser usado como um contrato padrão dentro do perfil.

3.3 - Critérios para Avaliação das Linguagens de Especificação de QoS

A avaliação dos critérios de cada paradigma é diferente, pois cada um deles é composto por um conjunto de serviços.

Baseado em [23], uma análise dos critérios utilizados para a avaliação dos paradigmas, observando-se que cada paradigma obedece a sua classificação em camadas. Portanto, é necessário que exista um levantamento do que é relevante ou não em cada camada a fim de enumerar as vantagens dos paradigmas em relação à composição de suas camadas.

Alguns critérios relevantes avaliados na Camada de Usuário são os aspectos que proporcionam facilidades para os mesmos, sendo necessário lembrar que o usuário não tem a pretensão de saber o que acontece para que o serviço seja fornecido. O usuário quer que o serviço seja realizado em conformidade com as opções de QoS escolhidas. Para facilitar o desenvolvedor da aplicação é primordial que seja fornecida uma Interface Gráfica amigável, que seja simples para que o usuário consiga realizar a escolha da QoS e que essa escolha reflita ao usuário os custos envolvidos.

Em relação à Camada de Aplicação são considerados cinco critérios para analisar a QoS oferecida:

- Expressividade: é avaliada a especificação de QoS em termos da sua capacidade de especificar uma grande variedade de serviços, seus recursos exigidos e suas regras de adaptação correspondentes.

- Declarativa: uma especificação de QoS deve ser informativa; dessa forma, deve especificar somente o que é exigido, não como a própria especificação é realizada.

- Independência: é avaliada se as especificações podem ser desenvolvidas independentemente do código funcional pela sua legibilidade e pela facilidade no desenvolvimento e na modificação de seus objetivos.

- Extensibilidade: nesse ponto é avaliado o quão fácil é estender uma linguagem para permitir a especificação de novas dimensões de QoS além das que já foram desenvolvidas, através da inclusão de novas classes.

- Reusabilidade: capacidade do código da especificação ser claro, fácil de ser desenvolvido, etc., facilitando o seu reuso por um outro desenvolvedor.

- Adaptabilidade: capacidade da linguagem representar requisitos de adaptação mediante negociação/renegociação da aplicação.

Jin e Nahrstedt [23] apresentam um estudo comparativo das linguagens de especificação de QoS retratado na Tabela 3, onde HQML mostrou-se, de maneira geral, uma linguagem com resultados satisfatórios no que se refere aos critérios importantes para a realização deste trabalho, que são Expressividade, Extensibilidade e Independência, além de ter a facilidade de ser uma linguagem baseada em XML [32].

Linguagens de QoS	**Expressividade**	**Declarativa**	**Independência**	**Extensibilidade**	**Reusabilidade**	**Adaptabilidade**
SafeTcl	Fraco	Regular	Bom	Fraco	Fraco	Bom
QoS-A	Bom	Bom	Bom	Bom	Fraco	Fraco
QuAL	Bom	Fraco	Fraco	Bom	Fraco	Bom
Fuzzy Control	Regular	Bom	Bom	Regular	Fraco	Bom
HQML	Bom	Bom	Bom	Bom	Fraco	Bom
QDL	Bom	Bom	Bom	Bom	Fraco	Bom
QML	Regular	Bom	Bom	Bom	Bom	Bom
XqoS	Bom	Bom	Regular	Bom	Fraco	Bom

Tabela 3: Comparação da camada de aplicação das Linguagens de Especificação de QoS.

Na Camada de Recurso do sistema é avaliado se as linguagens conseguem usar todo o recurso físico requerido e de que forma isso acontece, isto é, se não há perdas ou mau uso do mesmo.

A Tabela 4, apresentada por Gu et al. [39], ilustra a superioridade do HQML sobre as outras linguagens do gênero no que diz respeito ao tratamento das camadas de usuário, de aplicação e de recurso, às facilidades impostas pelo

Mapeamento Automático, Integração com Aplicação Web, Checagem de Consistência, além da Geração de Especificação Automática.

Linguagens de Especificação de QoS	QoS na Camada de Usuário	QoS na Camada de Aplicação	QoS na Camada de Recurso	Mapeamento Automático	Independente de Linguagem	Integração com Aplicação Web	Checagem de Consistência	Geração de Especificação Automática
Index	Sim	Não	Não	Não	Não	Não	Não	Não
SafeTcl	Não	Sim	Não	Não	Não	Não	Não	Não
QoS-A	Não	Sim	Não	Não	Não	Não	Não	Não
QuAL	Não	Sim	Sim	Não	Não	Não	Não	Não
QDL	Não	Sim	Sim	Não	Sim	Não	Não	Não
QML	Não	Sim	Não	Não	Sim	Não	Não	Não
RSL	Não	Não	Sim	Não	Sim	Não	Não	Não
TAOML	Não	Não	Sim	Sim	Sim	Sim	Não	Não
HQML	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
XQoS	Sim	Sim	Sim	Não	Sim	Não	Não	Não

Tabela 4: Comparação entre diferentes Linguagens de Especificação de QoS.

Essas facilidades são primordiais para que o desenvolvedor não se preocupe em como se dá o mapeamento de um modelo proposto pelo mesmo em parâmetros expresso pelo código HQML, abordado sucintamente no próximo capítulo.

4 - Linguagem de Marcação para Especificar QoS Hierarquicamente

O XML (eXtensible Markup Language) [32] é uma linguagem de especificação, usada por aplicações Web multimídia distribuídas num formato universal para documentos estruturados e dados na Web e também extensível. XML por si só não "diz" ao desenvolvedor da aplicação como especificar requisitos de QoS para sua aplicação, precisa definir um conjunto mínimo de tags apropriadas para permitir que os desenvolvedores expressem suas políticas e requisitos de QoS, baseado na sintaxe XML. Os desenvolvedores também podem definir suas próprias tags específicas para um serviço.

Na prática XML é um padrão para troca de informações entre diversas plataformas, que apenas possibilita a descrição de dados em um arquivo de formato texto. Várias linguagens de especificação de sistemas são derivadas de XML, o que possibilita ao XML ser considerado uma poderosa ferramenta para a publicação de informações. XML é considerada uma "linguagem-mãe" [40] das linguagens de especificação, pois a partir dele é possível serem desenvolvidas novas linguagens.

O XML foi escolhido para ser o modelo do HQML porque existe uma facilidade comum e natural entre as necessidades do XML de criar novos modelos bem sucedidos e os requisitos de especificação de QoS de aplicações multimídia para Web.

O HQML (Hierarchical QoS Markup Language) [2] é uma linguagem de especificação de QoS baseada em XML e suas especificações são classificadas em três níveis ou camadas, em conformidade com o detalhamento do dado no capítulo 3.

Na Camada de Usuário, as especificações HQML fornecem tags para especificar critérios de QoS qualitativos (por exemplo, alto, baixo, médio), tipo de aplicação (vídeo sob demanda, videoconferência, etc), preços para os serviços requeridos e qualquer outra tag desejada pelo desenvolvedor da aplicação, haja vista essa possibilidade de extensão do HQML por meio do ambiente descrito no capítulo 5. As especificações de QoS da camada de usuário são usadas, durante o tempo de execução, para encontrar a melhor medida entre a condição econômica do usuário, o nível de QoS preferido e os níveis de QoS disponíveis oferecidos pelos diferentes provedores de serviço. Não é necessário que os usuários conheçam especificações quantitativas detalhadas de todos os tipos de parâmetros específicos da aplicação que podem, potencialmente, ser muito complexos, como foi apresentado na camada de usuário descrita no capítulo 3.

Na Camada de Aplicação, as especificações HQML fornecem tags para especificar parâmetros de QoS do nível de aplicação de todos os tipos, políticas de QoS específicas à aplicação (regras de adaptação e de reconfiguração), além de qualquer outra tag que o desenvolvedor da aplicação desejar criar. As especificações de QoS dessa camada são usadas por aplicações para ativar e reforçar a QoS em benefício da aplicação mesmo se o suporte do Sistema Operacional e a QoS da rede está ausente.

Na Camada de Recurso, as especificações HQML fornecem tags para especificar diferentes requisitos de recurso de sistema (por exemplo, memória, cpu, largura de banda, etc.). As especificações nessa camada visam garantir os serviços de QoS nos níveis de rede e de sistema abordados no capítulo 2.

Embora HQML siga a sintaxe padrão do XML e pode ser facilmente utilizado, alguns pontos críticos exigem considerações cuidadosas:

- Algumas informações em especificações HQML não podem ser produzidas diretamente. Por exemplo, o desenvolvedor da aplicação não pode conhecer previamente os requerimentos de recurso do sistema para suas aplicações. Para as regras de adaptação, o desenvolvedor da aplicação precisa especificar os valores mínimos dos

parâmetros de cada início de adaptação, que decidirão o tempo de ativação de cada escolha de adaptação. Mas esses valores mínimos não podem ser facilmente produzidos;

- É necessário checar a consistência ou precisão das especificações HQML. Já que é permitido para os desenvolvedores de aplicação usarem HQML para especificar suas próprias políticas e requerimento de QoS, qualquer especificação ilegal pode não produzir os resultados esperados;

- Embora o DTD (*document type definition*), ou definição do tipo de documento [2] possa ser usado para checar alguns erros nos arquivos baseados em XML, ele não contém nada específico sobre especificações de QoS. Por exemplo, deve-se ter certeza que não há *deadlock* nas especificações das configurações da aplicação para uma aplicação multimídia distribuída;

- Os parâmetros de QoS entre dois componentes conectados devem ser consistentes. Se, por exemplo, um codificador MPEGII estiver conectado com um decodificador H261, ou um player de baixa qualidade estiver conectado com um servidor de vídeo de alto desempenho, a aplicação não funcionará apropriadamente.

Tais problemas de inconsistências podem ser solucionados por um verificador da semântica da especificação completa. Um exemplo é o QoSTalk [41], que é um ambiente visual que auxilia os desenvolvedores de aplicação a gerar os arquivos HQML fácil e corretamente.

4.1 - Modelo de Aplicação

Para compreender melhor a aplicação e o correto uso do HQML, é preciso conhecer o modelo de aplicação em que o HQML foi baseado. Foi considerado um modelo com componentes de aplicação genérico para caracterizar a estrutura de aplicações multimídia distribuídas. Todos os componentes de aplicação são construídos como tarefas, que executam operações específicas nos dados multimídia que passam por elas, assim como transformação, agregação, *prefetching*[6] e filtragem. Cada componente aceita a entrada com um nível de QoS Q^{in} e gera uma saída com um nível de QoS Q^{out}.

[6] Prefetching: técnica que antecipa a transmissão de possíveis requisições.

Para poder processar a entrada e gerar a saída, uma quantidade específica de recursos é necessária.

As tarefas podem ser conectadas em um dígrafo acíclico, que é chamado de configuração da aplicação. A configuração da aplicação é o fluxograma sob o qual o dado multimídia passa entre os provedores de serviço e o usuário final. Como exemplo, na aplicação de vídeo sob demanda, ilustrada na Figura 4, o dado é lido do disco, armazenado em buffers no remetente (servidor), transmitido pela rede e novamente armazenado no buffer no receptor (cliente). Então é decodificado antes de ser apresentado para o usuário. Os buffers são considerados componentes de aplicação ao invés de memória de armazenamento, permitindo assim utilizar diferentes formas de gerenciamento do buffer explicitamente.

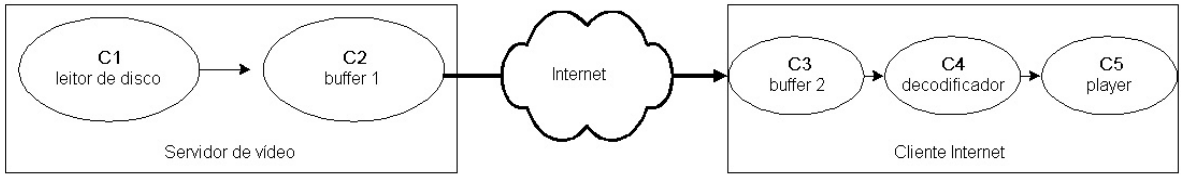


Figura 4: Configuração da aplicação para aplicação de VoD.

Pela grande diferença de implementação de uma mesma estrutura de um sistema distribuído multimídia, o HQML foi projetado para suportar tais diferenças. Como exemplo, a utilização de um dispositivo móvel, como um celular, por um usuário e a utilização de computador desktop por outro. O simples fato de um usuário portar um dispositivo distinto de outro usuário faz com que toda a efetivação da informação para ambos passe por arquiteturas e processos distintos, mesmo que eles busquem a mesma informação de um único servidor. A capacidade do celular de processamento e de recursos é inferior à capacidade do desktop. O HQML é baseado no conceito de Polimorfismo de Serviço [2] para QoS de diferentes sistemas.

Para suportar o polimorfismo de serviço foi desenvolvido um modo hierárquico de caracterização das configurações de aplicação. O modo hierárquico ajuda a solucionar o problema da escalabilidade para aplicações multimídia distribuídas complexas [41].

A arquitetura do HQML contempla a organização hierárquica da aplicação, com detalhamento de QoS nos três diferentes níveis, Usuário, Aplicação e Recurso. A Figura 5 ilustra essa hierarquia.

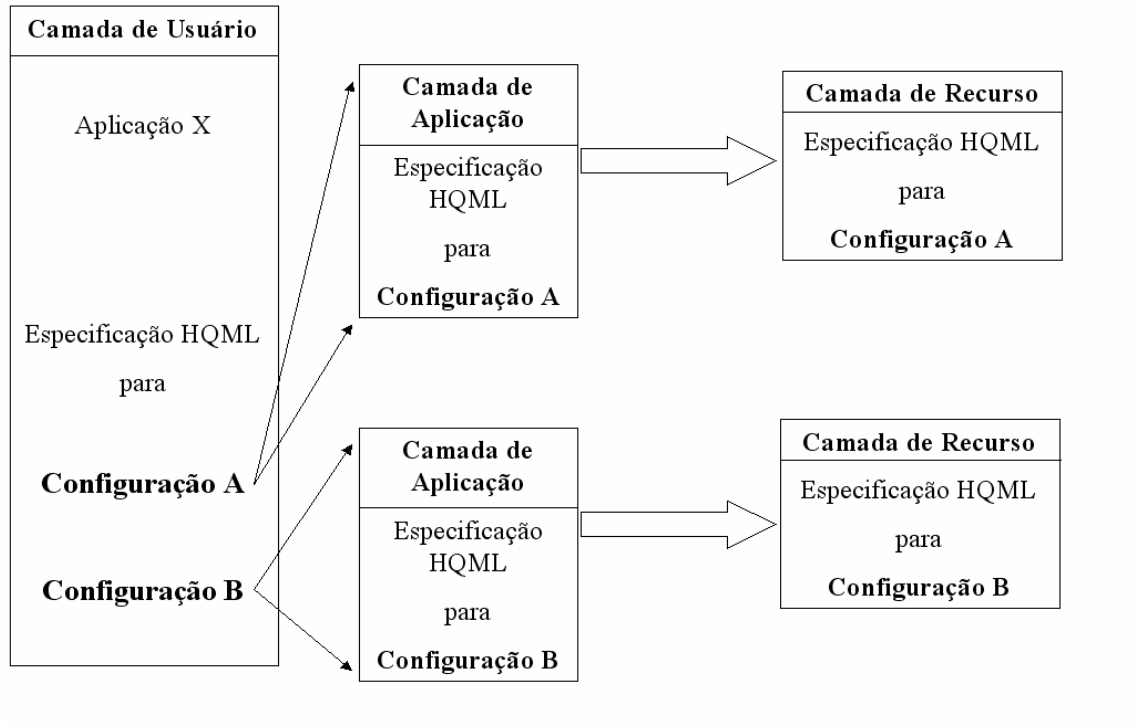


Figura 5: Organização hierárquica do HQML.

Essa hierarquia mostrada na Figura 5 pode ser representada por meio do seguinte exemplo prático: uma aplicação multimídia X qualquer pode ter várias formas de ser configurada, neste caso, Configuração A e Configuração B. Essas configurações A e B podem, por exemplo, ser configurações para uma aplicação de vídeo sob demanda, sendo a configuração A para computadores do tipo desktop enquanto a B para PDAs. Dessa forma, as configurações A e B possuem caminhos distintos, pois A tem que suprir requisitos diferentes de B e vice-versa. Por isso, na camada de usuário, A representa parâmetros como qualidade do vídeo de HDTV, enquanto B, devido a limitações do dispositivo, não pode tratar a qualidade do vídeo com esse valor. Da mesma maneira, na camada de aplicação, existem diferenças na forma de representação dos componentes envolvidos, como a necessidade de um transmissor *wireless* no caso da configuração B. Em relação à camada de recurso, é clara a maior necessidade da configuração A por processamento e largura de banda em relação à B, haja vista o

pouco poder de processamento e de uso da largura de banda dos dispositivos finais na configuração B.

4.2 - Sintaxe HQML para Especificação de QoS da Camada de Usuário

As especificações de QoS da Camada de Usuário são baseadas nas seguintes características [2]: as descrições que abrangem toda a aplicação, por exemplo, o nome da aplicação e qual o serviço provido; e a possibilidade de escolha dos serviços, como por exemplo, usuários podem especificar a qualidade que eles desejam pelo serviço desejado. A seguir, são apresentadas algumas tags possíveis de serem definidas na Camada de Usuário: as tags <User Preferences> e </User Preferences> definem o conteúdo referente às exigências do usuário, contendo todas as outras tags dessa camada; <Application Type> e </Application Type> definem a classe de aplicação multimídia desejada pelo usuário; <Desired QoS> e </Desired QoS> abordam a Qualidade de Serviço requerida pelo usuário; <Application Name> e </Application Name> definem o nome da aplicação; <Price> e </Price> se referem a algum possível valor pago pelo usuário para ter um informação mediante uso de uma aplicação; <Price Variation> e </Price Variation> definem a variação do preço pago pelo usuário; <Price Unit> e </Price Unit> tratam a unidade utilizada pelas tags <Price> e </Price>.

A Figura 6 ilustra um código HQML no nível da Camada de Usuário, sendo perceptível o que é importante para o usuário através das tags descritas no código, por exemplo, <Application Type> Video On Demand </Application Type> e <Desired QoS> High </Desired QoS>. Através dessas tags são definidos parâmetros importantes para a definição da faixa de valores que esses parâmetros devem receber quando na (re) negociação da QoS. O conteúdo descrito pelo par de tags pode variar de acordo com a necessidade do desenvolvedor, podendo o tipo de aplicação ser Videoconference (videoconferência), Audio on Demand (áudio sob demanda), ou qualquer outra aplicação multimídia. O mesmo se aplica ao par de tags que descreve a Qualidade de Serviço desejada, podendo ser ainda Average (média), Low (baixa), ou Any (qualquer).

Essas tags presentes na Figura 6 são as tags criadas automaticamente a partir das definições feitas no ambiente WSE, porém outras tags podem ser criadas de

acordo com a necessidade do desenvolvedor da aplicação por meio do ambiente WSE, que será abordado no capítulo 5. Assim, é possível que o desenvolvedor crie um par de tags <Image> </Image> a fim de quantificar valores para a resolução de uma imagem.

```
<User Preferences>
  <Application Type>
  Video On Demand
  </Application Type>
  <Desired QoS>
  High
  </Desired QoS>
 <Application Name>
 Untitled
 </Application Name>
</User Preferences>
```

Figura 6: Exemplo de Especificações HQML na Camada de Usuário.

4.3 - Sintaxe HQML para Especificação de QoS da Camada de Aplicação

A especificação de QoS da camada de aplicação é necessária para que os desenvolvedores de aplicações reconheçam que para acessar os serviços das camadas mais baixas, e controlar suas qualidades, as aplicações precisam descrever seus requerimentos de QoS.

Uma especificação nesta camada é uma aplicação específica, porém independente de recursos. Dois tipos de características da camada de aplicação podem ser definidas: uma é específica ao desempenho, expressa por parâmetros quantitativos e a outra é específica ao comportamento, expressa por parâmetros qualitativos. Uma linguagem de especificação deve prover certas abstrações para que os programadores não tenham que utilizar detalhes de níveis muitos baixos sobre quais recursos e ações precisam ser invocados. Há duas maneiras de prover abstrações de programação: uma é através de APIs e a outra é através de construtores de linguagem, estendendo uma

linguagem existente ou criando outras completamente novas. Um exemplo de código HQML inerente à camada de aplicação está apresentado na Figura 7.

Neste exemplo estão as principais tags referentes a essa camada que são geradas automaticamente pelo ambiente WSE, que será detalhado no próximo capítulo. As tags compreendidas entre o par de tags <QoSParameters> e </QoSParameters> contém os parâmetros de Qualidade de Serviço e as regras de adaptação definidas no ambiente WSE. Portanto, as tags <Delay>, </Delay>, <Jitter> e </Jitter> definem os parâmetros de QoS abordados na especificação e as tags <Min>, </Min>, <Max> e </Max> definem os valores para cada um desses parâmetros. As tags <AdaptationRules> e </AdaptationRules> contém as regras de adaptação definidas pelas tags <Name> e </Name> e a ordem de aplicação de cada regra baseada nas tags <Order> e </Order>.

Além disso, os valores para os parâmetros nessa camada variam de acordo com a aplicação que se quer especificar, portanto, dependendo da aplicação podem ser usados diferentes tipos de mídia, formatos de mídia, variação da taxa de quadro, ou profundidade da cor, por exemplo. Ainda, dependendo da aplicação tem-se variações diferentes de outros parâmetros, como vazão e taxa de perda. Lembrando que é possível estender o HQML a fim de se criar tags para delimitar os parâmetros que podem ser utilizados por uma nova aplicação.

```
<QoSParameters>
        <Delay>
          <Min>
          1
          </Min>
          <Max>
          2
          </Max>
        </Delay>
        <AdaptationRules>
          <Rule>
            <Name>
            No sound
            </Name>
            <Order>
            101
            </Order>
          </Rule>
        </AdaptationRules>
        <Jitter>
          <Min>
          0
          </Min>
          <Max>
          1
          </Max>
        </Jitter>
        <AdaptationRules>
          <Rule>
            <Name>
            No video
            </Name>
            <Order>
            201
            </Order>
          </Rule>
        </AdaptationRules>
</QoSParameters>
```

Figura 7: Exemplo de Especificações HQML na Camada de Aplicação.

4.4 - Sintaxe HQML para Especificação de QoS da Camada de Recursos

Especificações da Camada de Aplicação somente declaram requisitos em alto nível e de uma forma abstrata. Mais tarde, esses requisitos devem ser traduzidos em necessidades de recurso mais concretas, isto é, descrições de serviços devem ser providas, como quais recursos físicos serão necessários para a aplicação, quando eles precisam ser alocados e quais mecanismos devem ser adotados.

A importância das especificações de QoS da Camada de Recurso, de acordo com [2] são duas: essas especificações permitem que aplicações multimídia utilizem os serviços do Sistema Operacional e de QoS da Rede se eles estiverem disponíveis; e as especificações marcam os valores mínimos que determinam o tempo

de ativação de uma adaptação específica ou uma ação de reconfiguração. A Figura 8 mostra um exemplo de código HQML específico na camada de recurso do sistema. Nesse exemplo, podem ser notadas tags que descrevem as necessidades físicas das máquinas onde as aplicações multimídia serão executadas. As tags <Memory>, </Memory>, <Disk>, </Disk>, <Bandwidth> e </Bandwidth> descrevem essas necessidades que, na verdade, são recursos que serão alocados na execução da aplicação multimídia a fim de garantir os parâmetros de QoS definidos nas camadas superiores (usuário e aplicação). As tags <HardwareEnvironment>, </HardwareEnvironment>, <SoftwareEnvironment> e </SoftwareEnvironment> contém a arquitetura das máquinas, enquanto que as tags <Category> e </Category> definem a qual categoria o componente que está sendo descrito pertence. Já as tags <NodeLabel> e </NodeLabel> descrevem o número de cada componente especificado no ambiente WSE.

As especificações nesta camada são classificadas de acordo com suas granularidades. Duas categorias de granularidade são adotadas: grossa e fina. Através da granularidade grossa é esperada uma especificação meta-nível, enquanto pela granularidade fina são esperadas descrições concretas de recursos requeridos. Em relação a granularidade grossa, algumas especificações na Camada de Recurso de QoS somente especificam requisitos de recursos, preferivelmente, de uma forma abstrata. Por exemplo, eles podem especificar qual (quantidade) recurso é requerido, mas não importa quando os recursos precisam ser alocados, ou qual ação tomar se o requerimento do recurso não for encontrado, ou se várias instâncias do recurso (como processadores) estão disponíveis e qual delas deve-se utilizar.

```
<ServerGroup>
  <Server>
    <Name>
    Server1
    </Name>
    <Category>
    Servers
    </Category>
    <NodeLabel>
    0
    </NodeLabel>
    <HardwareEnvironment>
    Sun Ultra60
    </HardwareEnvironment>
    <SoftwareEnvironment>
    Solaris
    </SoftwareEnvironment>
    <Memory>
    512000 KB
    </Memory>
    <Disk>
    40000 MB
    </Disk>
    <Bandwidth>
    100 Mbits
    </Bandwidth>
     <AtomicComponent>
       <Name>
       Software Server
       </Name>
       <Category>
       Clients
       </Category>
       <NodeLabel>
       1
       </NodeLabel>
       <Memory>
       256000KB
       </Memory>
       <Disk>
       20000MB
       </Disk>
       <Bandwidth>
       50 Mbits
       </Bandwidth>
     </AtomicComponent>
  </Server>
</ServerGroup>
```

Figura 8: Exemplo de Especificações HQML na Camada de Recurso.

Para aplicações multimídia, especificações de granularidades mais finas são requeridas, pois são esperadas de uma especificação na Camada de Recurso com granularidade fina descrições de requerimentos de QoS quantitativa e qualitativa; tempo para o requerimento do recurso, como quando e por quanto tempo os recursos precisam ser alocados e regras de adaptação.

5 - Ambiente para Especificação de Aplicações Multimídia com Suporte de Qualidade de Serviço

O Ambiente para Especificação de Aplicações Multimídia com Suporte de Qualidade de Serviço (WSE) é um ambiente visual desenvolvido na linguagem Java que disponibiliza componentes visuais ao desenvolvedor de aplicações para criar especificações de sistemas multimídia distribuídos com garantia de QoS.

Essas especificações são descritas por meio de figuras geométricas acessadas por botões. Além disso, o WSE possibilita descrever ainda o tipo de ligação entre os componentes do sistema multimídia, bastando clicar no botão correspondente ao tipo de interligação (sem fio ou cabeada).

O estabelecimento de parâmetros de QoS acontece por meio de caixas de diálogo específicas de cada componente (server, client, gateway, peer) e podem ser preenchidas da forma com que o desenvolvedor queira, desde que utilize palavras reservadas que sejam entendidas pelo Executor HQML.

É neste estabelecimento que se encontra o diferencial do ambiente, pois possibita estabelecer limites mínimos e máximos para cada parâmetro de rede atuante na QoS do sistema, além de elencar regras de adaptação do vídeo a cada violação de um dos limites. Isso acarreta na (re)negociação para a utilização de filtros nos vídeos que mantém a reprodução do vídeo de acordo com a qualidade desejada.

Após a especificação do Sistema, a verificação de consistência das Qualidades de Serviço pode ser aferida pelo sistema. O WSE inclui ainda o compilador que executa os mapeamentos automáticos entre os parâmetros de QoS das camadas de

aplicação e de recurso do sistema, auxiliando o desenvolvedor na tarefa de lidar com especificações de QoS em um baixo nível. Além desses componentes, o WSE disponibiliza também o Gerador de HQML e o Executor HQML.

5.1 - Visão Geral da Arquitetura

Os passos necessários a serem seguidos pelo desenvolvedor até a implementação da aplicação, levando em consideração os elementos da arquitetura do WSE presentes na Figura 9 são:

- O início do desenvolvimento de uma especificação de uma aplicação multimídia distribuída acontece no Editor Visual de QoS Hierárquico onde são oferecidas classes de aplicações multimídia (video on demand, audio on demand, video conference, audio conference, new application) e escolhida a Qualidade de Serviço a ser mantida no sistema (high, average, low, any);

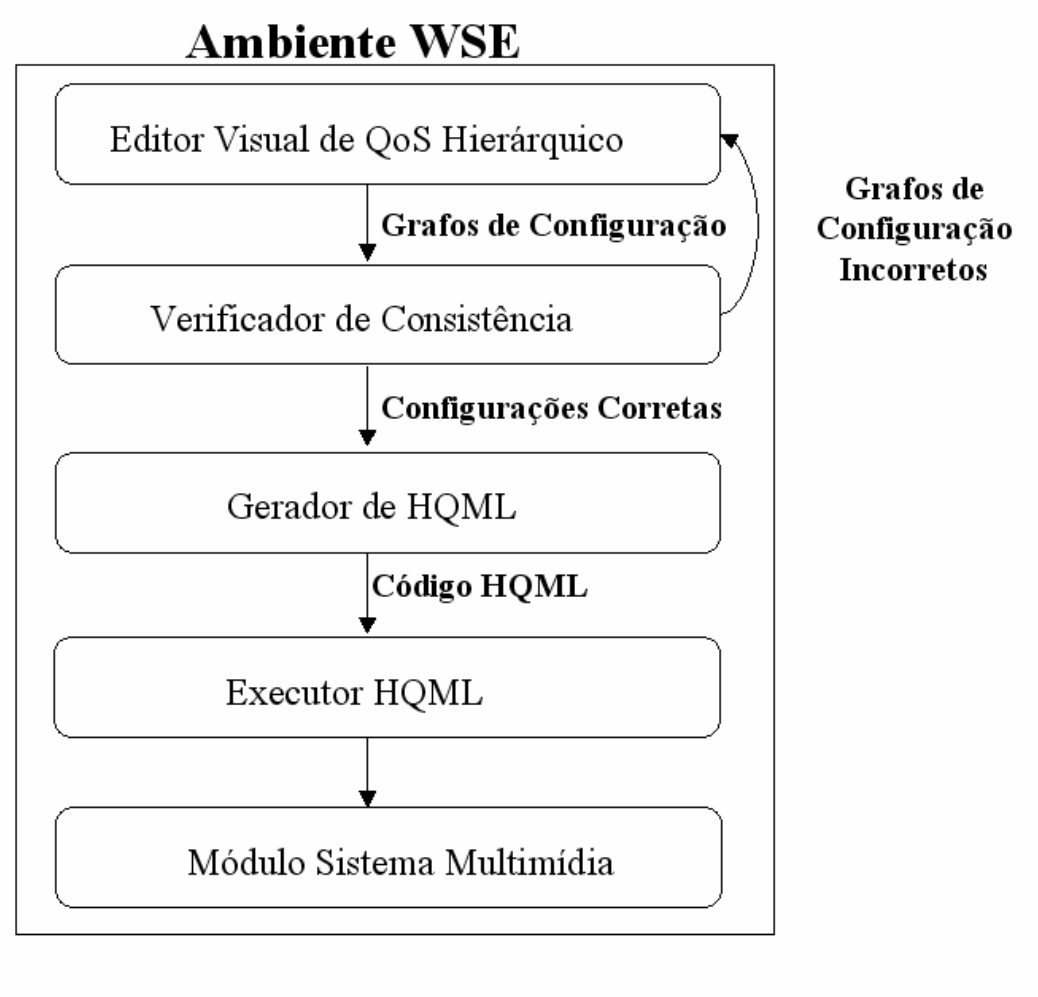


Figura 9: Arquitetura do Ambiente para Especificação de Aplicações Multimídia com Suporte de QoS.

- Posteriormente, o desenvolvedor da aplicação usa o Editor Visual de QoS Hierárquico para desenhar todas as configurações possíveis da aplicação e colocar as entradas de todos os tipos de requisições de QoS das Camadas Usuário e Aplicação;

- Depois, o desenvolvedor usa o Verificador de Consistência para detectar possíveis inconsistências nas especificações de entrada. Se existir qualquer inconsistência, mensagens de erros são retornadas para o desenvolvedor no Editor Visual de QoS Hierárquico;

- Posteriormente, as configurações corretas da aplicação e as especificações de QoS completas (Camadas Usuário, Aplicação e Recurso) são passadas

para o Gerador de HQML, que analisa as configurações da aplicação para gerar o arquivo HQML de forma automática;

- Finalmente, o arquivo HQML é salvo a fim de poder ser lido posteriormente pelo Executor HQML, que tem por finalidade ler o arquivo HQML, entendê-lo e passar esses parâmetros para o sistema multimídia por meio de comandos do Sistema Operacional.

5.2 - Editor Visual de QoS Hierárquico

O Editor Visual de QoS Hierárquico é uma ferramenta visual que permite ao desenvolvedor desenhar as configurações desejadas para aplicações multimídia distribuídas. Este editor foi adaptado do editor presente no QoSTalk [41], adicionando, dentre outras funcionalidades, caixas de diálogo para que o desenvolvedor coloque as entradas para todos os tipos de políticas e parâmetros de QoS. O propósito do Editor Visual de QoS Hierárquico é baseado no modelo descrito na seção 4.1 - Modelo de Aplicação e segue a proposição hierárquica, isto é, o desenvolvedor pode refinar um componente composto desenhando todos os seus subcomponentes em um subframe.

Para facilitar o seu uso, o Editor possui formas diferentes para representar os diferentes tipos de componentes, sendo o componente servidor representado pelo retângulo, o gateway pelo losango, elipse representa o cliente componente, e o retângulo arredondado o componente peer[7].

Utilizando a ferramenta visual, é possível descrever qualquer sistema multimídia distribuído, variando desde sistemas simples compostos por somente um cliente e um servidor até sistemas formados por vários clientes e servidores. Essa forma de descrição de vários clientes e servidores acontece devido à forma com que o Editor descreve cada um dos componentes (servidor, cliente, gateway, peer) envolvidos na modelagem e pela representação hierárquica dos componentes, sendo que cada componente pode ser detalhado por uma nova janela (C*hildFrame*) do Editor, ou por propriedades do componente.

[7] Peer: qualquer host que pode exercer as funções de cliente e servidor.

Os componentes das aplicações, como grupo (união de dois ou mais componentes formando um cluster), hardware e software são representados pelas cores azul, verde e amarelo, respectivamente, o que facilita no entendimento e na representação dos objetos no Editor. Exemplos desses componentes podem ser vistos na Figura 10 representados pelos quadrados.

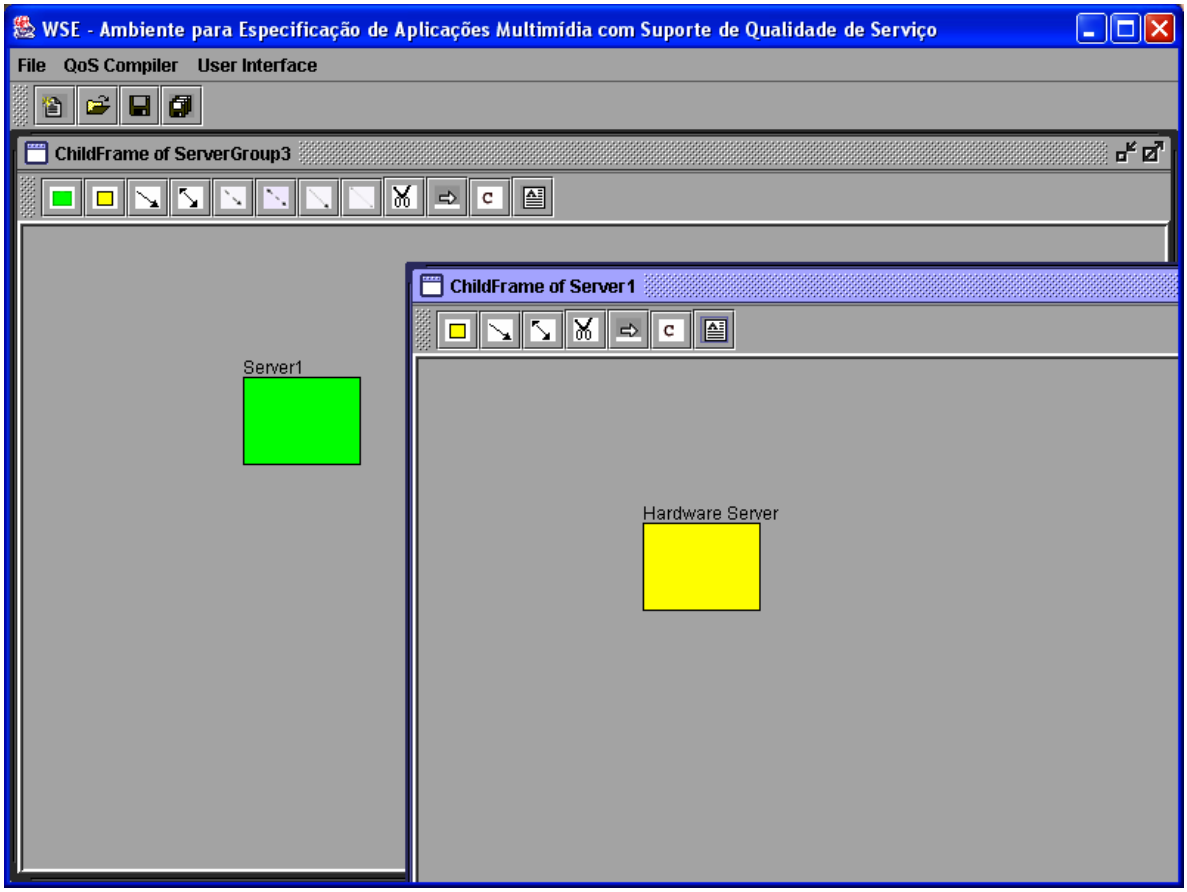


Figura 10: Editor Visual de QoS Hierárquico.

A Figura 10 traz a interface gráfica do Ambiente para Especificação de Aplicações Multimídia com Suporte de Qualidade de Serviço, sendo possível verificar toda a estruturação do ambiente que é composto por uma barra de menus na parte superior, uma barra de atalhos logo abaixo e uma área para a descrição de aplicações multimídia. A presença de dois *ChildFrames* nessa área de descrição caracteriza o desenvolvimento hierárquico dos componentes da aplicação multimídia distribuída, que ocorre da seguinte forma: após a representação do componente grupo, obrigatoriamente são necessárias as representações de ao menos um componente hardware e um componente software. Essas representações devem ser em *ChildFrames* diferentes, isto

é, a partir do componente grupo, deve ser criado um *ChildFrame* para a representação do componente hardware e, a partir desse componente, criado outro *ChildFrame* para a representação do componente software.

O Editor Visual de QoS Hierárquico possui caixas de diálogos referentes a cada componente, quando não será mais refinado. As características dos componentes hardware e software são preenchidas nas suas respectivas caixas de diálogo correspondentes. A Figura 11 mostra a caixa de diálogo referente ao componente software do cliente. Essa é a caixa de diálogo mais importante, pois é nela em que são preenchidos os parâmetros e seus valores, além das regras de adaptação referentes a cada parâmetro, especificamente à Camada de Aplicação. A descrição de QoS na aplicação fica componentizada e tais informações podem ser utilizadas para a Camada de Recursos empregar melhor a distribuição dos recursos em tais componentes.

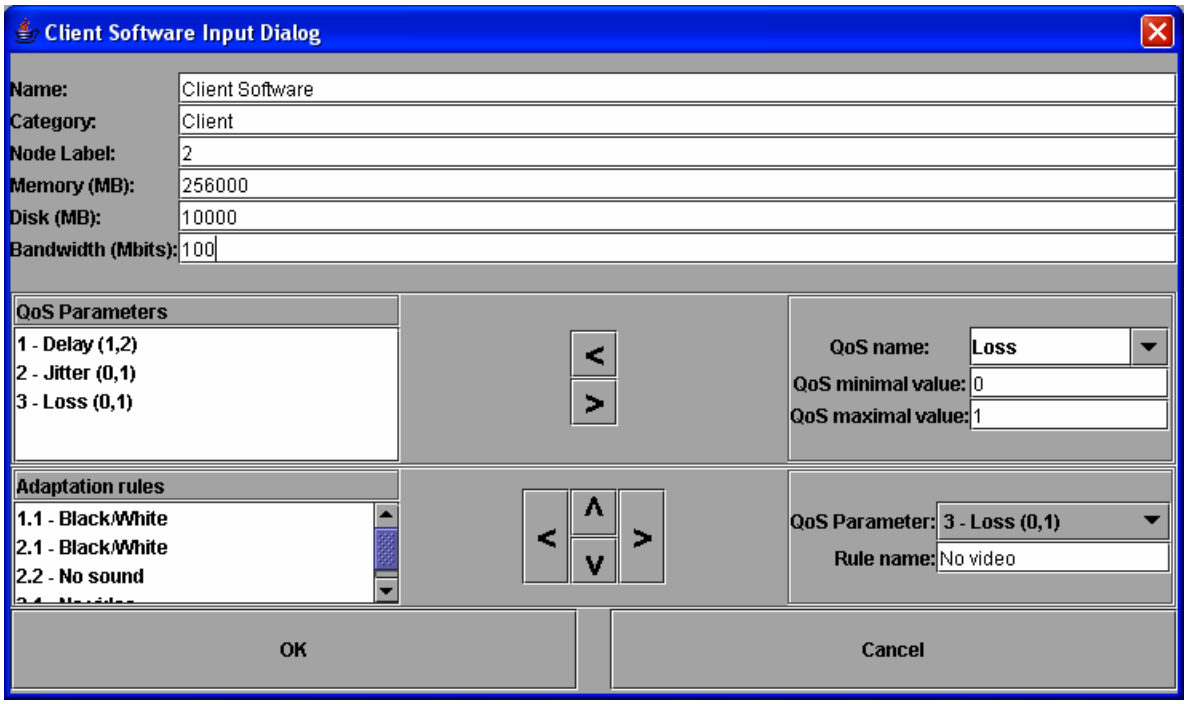


Figura 11: Caixa de diálogo referente ao software do cliente.

A caixa de diálogo referente ao hardware do cliente é mostrada na Figura 12. Essa caixa mostra as características do componente hardware, bem como as especificações referentes à Camada de Recurso que devem ser preenchidas.

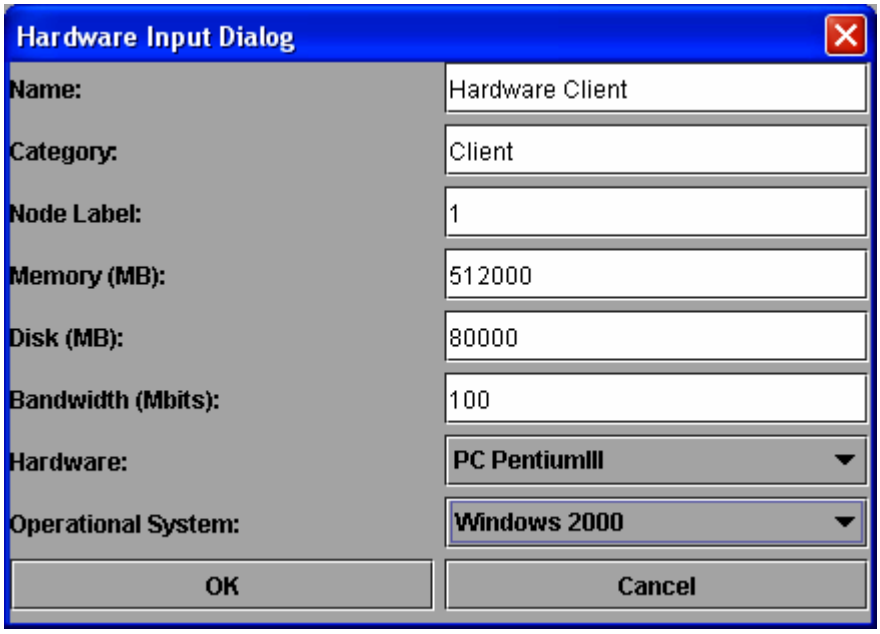


Figura 12: Caixa de diálogo referente ao hardware do cliente.

5.3 - ConfigG: Gramática Especial de Relação de Símbolo Limitada

As duas principais tarefas da verificação de consistência são encontrar configurações incorretas da aplicação e descobrir parâmetros incompatíveis entre dois componentes conectados. Para isso, utiliza-se uma teoria de gramática de grafos formal onde cada configuração da aplicação é descrita por uma sentença da gramática de grafos. Portanto, os problemas de checagem de consistência nas especificações de QoS visuais são reduzidos para o que sobra após a eliminação dos erros de configuração pela sentença da gramática de grafos e seus analisadores sintático e semântico.

A fim de fortalecer a verificação de consistência e descobrir parâmetros de QoS incompatíveis, existe uma gramática especial chamada ConfigG [2], sendo que a mesma é baseada em uma outra gramática, a Relação de Símbolo [43] (Symbol Relation – SR) que é reconhecida por ser uma gramática de grafos para lidar com estruturas de grafos complexas. A ConfigG previne, por exemplo, inconsistências em configurações semelhantes as descritas na Figura 13, como em (a) onde o componente

"c1" não está conectado com nenhum outro componente; (b) o componente cliente "c4" não recebe nenhum dado do componente o qual está conectado, neste caso o gateway "c3"; e em (c) não há componente cliente para receber a vazão de dados de mídia. Além disso, configurações com parâmetros incompatíveis de QoS como um codificador de MPEGII conectado em um decoficador H261, ou um player de vídeo de baixa qualidade, que consegue reproduzir vídeos com 15 qps (quadros por segundo), conectado com um servidor de vídeo de alta performance que envia streams de vídeo a 30 qps também são analisados pela ConfigG.

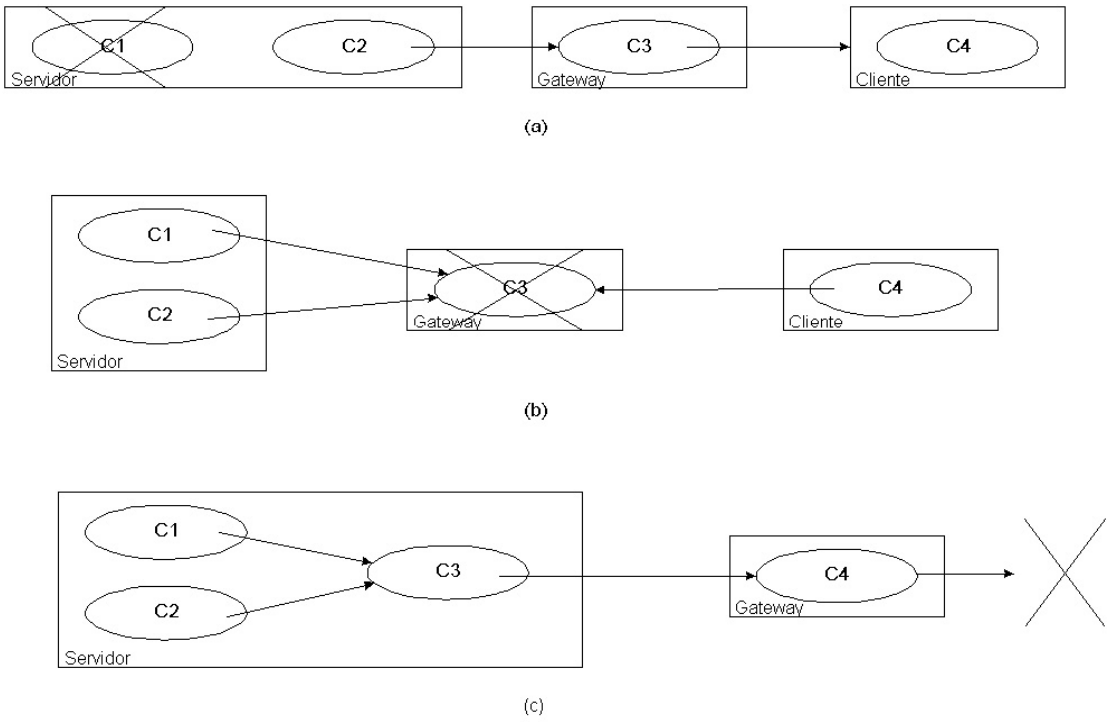


Figura 13: Exemplos de Grafos de Configurações Incorretos.

A verificação de consistência é dividida em dois estágios, sendo o primeiro a tradução das configurações de entrada da aplicação pelo Analisador Sintático da ConfigG em uma sentença da ConfigG. Se o processo de tradução não obtiver sucesso, o grafo da configuração é incorreto, caso contrário, uma árvore de derivação [43] é gerada para o grafo de configuração. No segundo estágio, o Analisador Semântico da ConfigG varre a árvore de derivação para checar as consistências semânticas de acordo com as regras semânticas associadas com cada passo da derivação.

A ConfigG é a sextupla (S, T, V_N, R, SP, RP) onde:

- S é o símbolo inicial;

- O alfabeto T={AudioServer, AudioPlayer, VideoServer, VideoPlayer, MediaServer, MediaPlayer, ImageServer, ImageDisplay, MediaRecorder, VoiceMailRecorder, Prefetch, VisualTracker, RemoteControl, Transcoder, FrameDropper, t} é o conjunto de terminais representando os componentes de serviço (software), como players de áudio e vídeo e visualizador de imagens;

- O conjunto não terminal V_N= {S, SC, GWC, CC, A, B, C, D, AT, BT, CT, DT}. S é o símbolo inicial. SC, GWC, CC e GC representam Server Cluster, Gateway Cluster, Client Cluster e General Cluster, respectivamente. A, B, C e D representam hardware do servidor, hardware do gateway, hardware do cliente e hardware geral, respectivamente. AT, BT, CT e DT são símbolos temporários usados durante a derivação para evitar ambigüidade;

- O Conjunto de Relação de Símbolo R={fl (FixedLink), mul (MobileUserLink), mhl (Mobile-HostLink)} é o conjunto de terminais representando a relação entre os componentes;

- SP é o conjunto de regras de produção para os símbolos, RP é o conjunto de regras de produção para as relações.

O conjunto completo das regras de produção da ConfigG é apresentado no Apêndice C [44].

Esse módulo de verificação de consistência utilizou a implementação feita pelo ambiente QoSTalk [41] que trata a consistência de mesma forma que o WSE.

5.4 - Gerador de HQML

O Gerador de HQML é responsável por recolher todas as características descritas na especificação da aplicação multimídia no Editor Visual de QoS Hierárquico e transformá-las no código HQML. As informações que descrevem as características da especificação são fornecidas mediante o preenchimento das caixas de diálogo correspondentes a cada componente, além das informações escolhidas que definem a

aplicação na Camada de Usuário (tipo de aplicação e QoS da aplicação) e as que são mapeadas para a Camada de Recurso. Além disso, a própria modelagem feita pela descrição dos objetos (servidor, gateway, cliente, ou peer) existentes também fará parte do código HQML.

O código HQML é gerado através da "varredura" de todos os componentes descritos no Editor Visual de QoS Hierárquico. Cada componente possui informações que são descritas por meio das suas caixas de diálogo. Essas informações são recolhidas e mostradas em uma estrutura que representa o código HQML. Esse código é em linguagem de marcação e especifica a modelagem nas Camadas de Usuário, Aplicação e Recurso. Essa estrutura remonta todas as tags utilizadas no código HQML, conforme descrito no capítulo anterior, preenchendo as tags com as informações da especificação.

5.5 - Executor HQML

O Executor HQML também faz parte do WSE e é responsável por verificar o código HQML e configurar um sistema multimídia qualquer, seja este um ambiente de emulação, um framework, ou um esquema de aplicação pré-definido.

A partir do Executor, o sistema multimídia (framework, aplicação ou sistema emulado) é alimentado com os valores presentes nas tags escritas em HQML.

Ao implementar uma aplicação que utilize as informações descritas no código HQML, esse código é base fidedigna para implementação e deve ser consultado a fim de garantir o que foi descrito na modelagem. Essa consulta ocorre no momento em que é iniciada uma aplicação. Neste momento, é chamado o Executor que tem a capacidade de ler e entender os parâmetros descritos no código HQML. Dessa forma, o Executor detém a garantia de que os parâmetros foram interpretados de maneira correta, e que os mesmos serão passados para a aplicação, a fim de serem utilizados na negociação e manutenção da QoS.

O Executor, na realidade, baseia-se em um arquivo do tipo texto que contém somente as informações necessárias para a "alimentação" da classe

ControlServer. Essa classe necessita dos parâmetros de QoS e das regras de adaptação para preencher a tabela que contém os comandos de filtragem do vídeo.

O estudo de caso ilustrando uma modelagem possível por meio do ambiente WSE é uma aplicação de vídeo sob demanda composto por 3 componentes: um cliente, um servidor e o proxy, como descrito na Figura 14, em que o cliente requisita um vídeo ao servidor usando o proxy como mediador. O cliente requisita o vídeo através do Mplayer [45], a implementação do proxy é descrita a seguir e o servidor utilizado é o Apache [46].

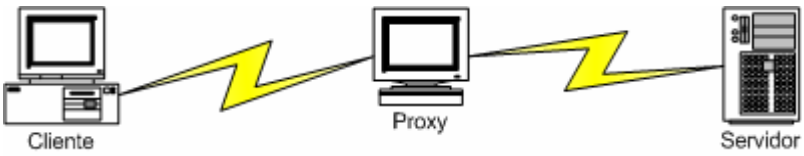


Figura 14: Estrutura da Aplicação de Vídeo Sob Demanda.

As principais classes utilizadas na implementação do proxy nessa aplicação de vídeo sob demanda estão no Apêndice A: Classe Connection, Classe ProxyTunnel, Classe Server, Classe ControlServer, Classe Controller e Classe ControllerGUI.

A criação do caminho entre o cliente e o servidor, mediado pelo proxy, é feita pela Classe Connection. Sua única função é manter esse caminho que foi criado repassando todo o conteúdo que chega ao proxy oriundo do servidor para o cliente e vice-versa. Os comandos que executam a adaptação estão na Classe ControllerGUI. O executor HQML está na Classe ControlServer. É por meio dessa classe que o arquivo HQML é lido e o método setFilter é chamado. Esse método é responsável por invocar o filtro utilizado na adaptação.

O proxy recebe a requisição, verifica qual é o servidor e tenta estabelecer uma conexão com o servidor, repassando a requisição do vídeo feita pelo cliente. O servidor verifica a existência do vídeo (arquivo) requisitado. Se a resposta for positiva, o mesmo envia uma resposta para o proxy estabelecendo a conexão e já enviando os primeiros bytes referentes ao arquivo. O proxy recebe esses bytes e os envia ao cliente, estabelecendo a conexão para a transmissão do vídeo.

No momento em que o proxy recebe os primeiros bytes do arquivo requisitado, já ocorre a medição dos parâmetros da rede, através da classe Controller. Essa medição é feita por um processo que é executado no proxy e por um outro processo executado no cliente. Isso ocorre devido à necessidade do proxy enviar requisições para o cliente e o mesmo ter que responde-las. Essas requisições e respostas são feitas por esses processos que compõe um monitor de rede que verifica alguns parâmetros como atraso e variação do atraso. A vazão e a perda, outros dois parâmetros importantes em se tratando de QoS, são verificados pelo processo do proxy que recebe os bytes enviados pelo servidor e os repassa para o cliente.

À medida que a transmissão dos dados do vídeo é iniciada, a medição dos parâmetros ocorre em paralelo. Esses valores dos parâmetros são requisitos para que ocorra ou não a adaptação no vídeo. Por exemplo, se o valor do atraso está fora do pré-estabelecido como ideal para uma transmissão de vídeo sob demanda com qualidade alta é repassado pelo monitor para o proxy que executa uma "filtragem" nesse vídeo, como determinado pela política de adaptação inserida na modelagem e descrito no código HQML. A filtragem ocorre dependendo das definições inseridas na modelagem e descritas no código HQML. Para cada regra de adaptação, pode ser associado o nome de um filtro, ou um nome de uma ação a ser tomada, que será responsável por explicitar o procedimento a ser executado caso necessário.

A adaptação no vídeo ocorre por meio do comando ffmpeg -qscale X -ab Y -f mpeg -i - -f mpeg -, existente na biblioteca Ffmpeg [47] em que:

- ffmpeg é a chamada a um programa fornecido junto com a biblioteca homônima;
- -qscale significa que o filtro atuará na alteração dessa característica do vídeo;
- X ilustra somente o valor que pode ser colocado como opção na alteração da qscale do vídeo;
- -ab indica que o vídeo sofre alterações na taxa de bits do áudio;
- Y ilustra o valor que pode ser colocado como opção para a alteração da taxa de bits do áudio;

- o primeiro -f significa que um formato do vídeo de entrada está sendo forçado no formato indicado;
- mpeg é o formato forçado para a entrada do vídeo;
- -i indica entrada de vídeo;
- o primeiro sinal "–" indica que o vídeo está sendo capturado pela entrada padrão do comando (stdin);
- o segundo -f significa que um formato está sendo forçado na saída do vídeo;
- mpeg é o formato forçado para a saída do vídeo;
- o segundo sinal de "–" indica a biblioteca Ffmpeg para usar a saída padrão (stdout) como saída do vídeo filtrado.

5.6 - Módulo Sistema Multimídia

O Módulo Sistema Multimídia é qualquer sistema multimídia que vai ser alimentado pela especificação da modelagem no Editor Visual de QoS Hierárquico. Esse sistema pode ser um framework como em [48], uma aplicação multimídia qualquer, como uma teleconferência, ou um sistema emulado.

A fim de garantir resultados válidos para a execução dessa aplicação, optou-se por emular as condições da rede com o uso do Nist Net [49]. A emulação é uma combinação de duas técnicas comuns para testar o código da rede [50]: simulação, que é possível definir como um ambiente sintético para representações do código de execução; e teste real, um ambiente real para a execução de código real. Nestes termos, a emulação é um ambiente "semi-sintético" para a execução de código real. O ambiente é semi-sintético no sentido que é uma implementação real da rede com meios suplementares para introduzir parâmetros de rede.

O Nist Net [49] é uma ferramenta de propósito geral para emulação de performances dinâmicas em redes IP. Esta ferramenta foi projetada para permitir a reprodução de experimentos controlados com aplicações adaptativas (ou sensitivas) à

performance da rede. Por operar no nível de roteamento (nível IP), o Nist Net pode emular características críticas de performance fim a fim, tais como congestionamento, perdas de pacotes e gargalos que limitam a largura de banda.

A emulação oferece assim mais vantagens que as duas situações: simulações (um ambiente controlado que é relativamente rápido e fácil de se montar) e teste real (código real em um ambiente real que evita todas as perguntas sobre a fidelidade da representação). Ainda, a emulação pode minimizar o investimento requerido para testar a rede por sua fácil instalação e uso.

A emulação de rede é uma solução intermediária, propiciando um ambiente capaz de reproduzir situações críticas e executando código do próprio sistema. Com este método, é possível testar desde ambientes grandes e complexos, até ambientes simples, utilizando uma pequena rede em um laboratório, sendo que não é necessário refazer o código do sistema para o ambiente de execução. Além disso, o custo na utilização de um emulador é muito baixo, uma vez que com o mesmo equipamento é possível modelar uma variedade enorme de ambientes.

A implementação do Nist Net consiste em um módulo de extensão para o kernel do sistema operacional Linux e uma interface gráfica para o usuário da aplicação. Em uso, a ferramenta permite a um computador pessoal configurado como roteador, emular numerosos cenários de performances complexas, como limitações de largura de banda, atrasos, duplicação e perdas de pacotes de dados, dentre outros. A interface gráfica permite ao usuário monitorar e selecionar o tráfego que passa através do roteador, e aplicar efeitos selecionados de performance aos pacotes IP do fluxo de informação. Em adição à interface interativa, o Nist Net pode ser dirigido por traços de medidas feitas a partir de condições de redes reais.

O Nist Net é um pacote de emulação que executa na maioria dos PCs equipados com Linux sendo de fácil instalação e de distribuição gratuita. Para interagir com esse emulador, várias ferramentas estão disponíveis no pacote, tais como o Cnistnet e o Xnistnet (a interface gráfica). O primeiro provê uma interface de linhas de comando que permite interação com o emulador através de argumentos como –u (ativa o emulador) e –d (desativa o emulador). O segundo é uma interface gráfica, a qual

permite monitorar e controlar facilmente o fluxo de informação desejado. As Figuras 15, 16 e 17 mostram a interface gráfica Xnistnet e explicam a sua funcionalidade [49].

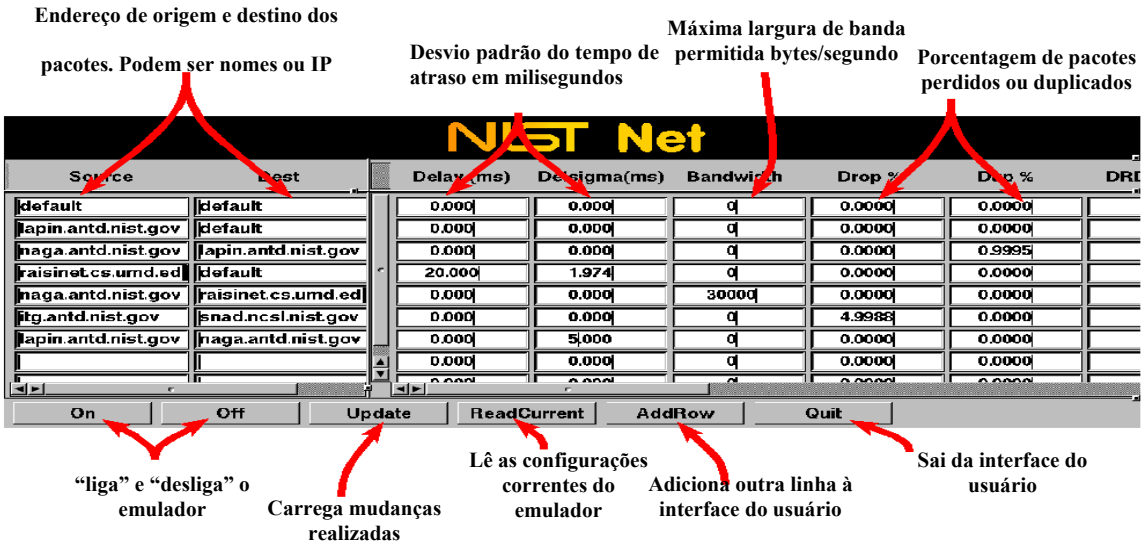


Figura 15: Interface Gráfica Parcial do Emulador Nist Net (Xnistnet).

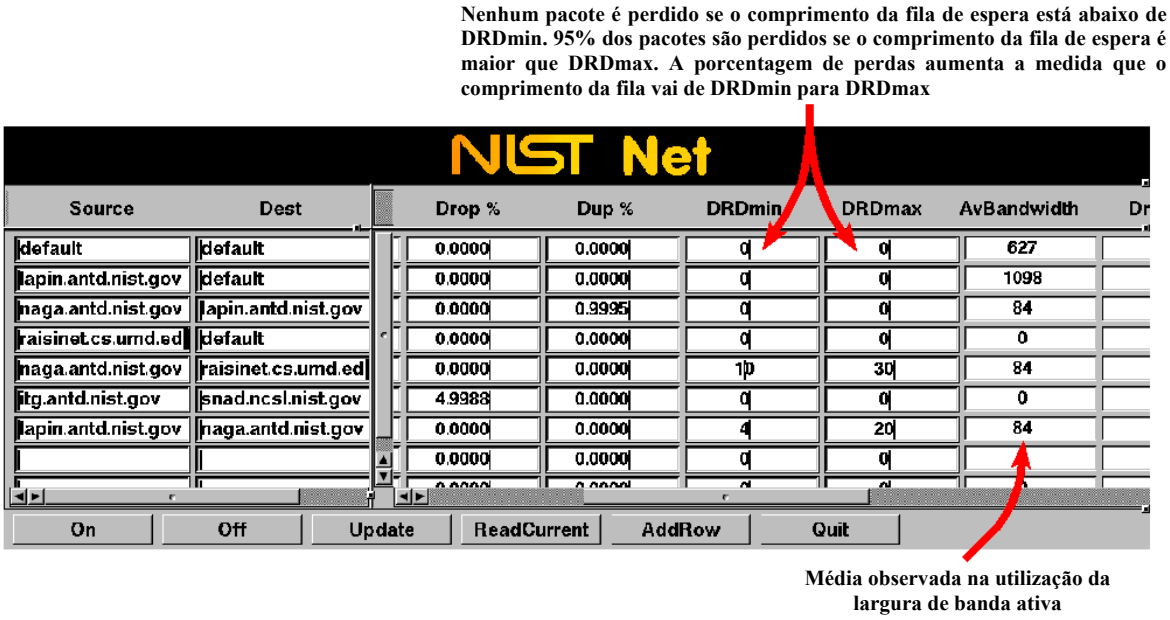


Figura 16: Interface Gráfica Parcial do Emulador Nist Net (Xnistnet).

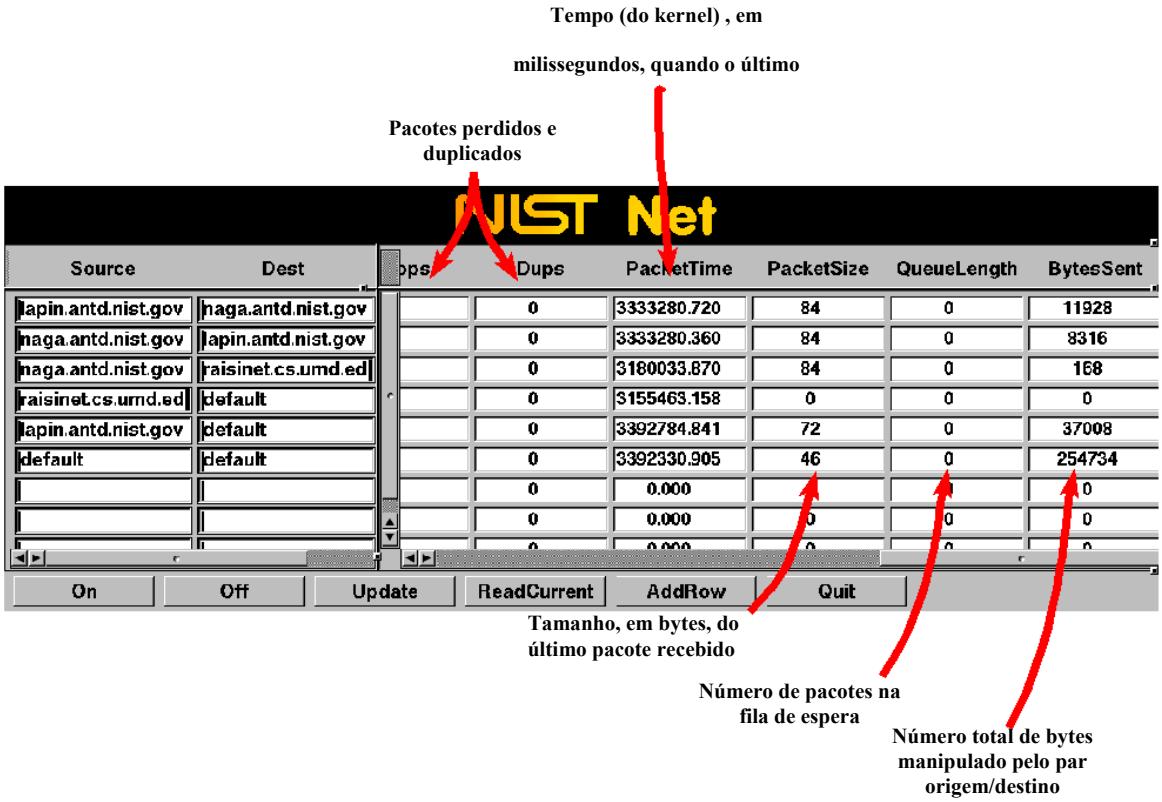


Figura 17: Interface Gráfica Parcial do Emulador Nist Net (Xnistnet).

5.7 – Trabalhos Correlatos

Nessa seção é apresentado o ambiente motivador para a efetivação desse trabalho, o QoSTalk. O QoSTalk é um *framework* de desenvolvimento e programação de QoS visual, que permite de forma visual especificar características específicas da aplicação em diferentes granularidades [51]. Ele possui três partes principais: Construtor de Configuração Hierárquico, Exemplos de Documentação de QoS e o Interpretador de QoS.

O Construtor de Configuração Hierárquico apresenta um conjunto de ferramentas visuais para o desenvolvedor.

Os Exemplos de Documentação de QoS é um arquivo HQML que é gerado automaticamente depois de o desenvolvedor da aplicação configurar uma configuração possível da aplicação.

O Interpretador de QoS traduz o arquivo HQML em um perfil que é necessário para o *framework* de *middleware* distribuído. Durante a execução, o *middleware* instancia componentes distribuídos e reforça a entrega da QoS de acordo com a informação salva no perfil de QoS.

Esse *middleware* é o $2K^Q$ [42], fazendo com que o QoSTalk somente sobre uma arquitetura pré-definida e com aplicações específicas. Ele não é um ambiente voltado para o desenvolvimento de qualquer aplicação multimídia, pois esta deve estar em conformidade com os elementos distribuídos utilizados pelo Interpretador de QoS.

O WSE, diferentemente do QoSTalk, é voltado para o desenvolvimento de qualquer aplicação multimídia, não sendo necessário uma arquitetura distribuída pré-definida, nem a utilização de um Sistema Operacional distribuído para que suas especificações sejam válidas nas três camadas (Usuário, Aplicação e Recurso). Além disso, o WSE agrega o emulador Nist Net para validar as aplicações especificadas por meio de resultados experimentais.

O desenvolvimento de uma aplicação no ambiente WSE e sua interação com o Módulo Sistema Multimídia, que, nesse trabalho, é o emulador de rede Nist Net, será apresentado no capítulo 6 na forma de estudo de caso.

6 - Estudo de Caso: Uma Aplicação de Vídeo Sob Demanda

O ambiente WSE de descrição de sistemas multimídia distribuídos permite que seja especificada a Qualidade de Serviço de seus componentes e as regras de adaptação. A especificação de sistemas multimídia é de forma hierárquica, de modo que a QoS possa também ser descrita nas Camadas de Usuário, Aplicação e Recurso, sendo descrita pelo WSE desde a fase de especificação do sistema até a implementação do mesmo. Este capítulo apresenta um estudo de caso de uma aplicação de vídeo sob demanda que possui aspectos de monitoramento, (re) negociação, notificação e reações a mudanças na QoS. Dessa forma, ficam evidenciadas as características descritas nas camadas de Usuário, Aplicação e Recursos e comprovados os mecanismos de controle da QoS. Serão explicitados todos os passos necessários para a modelagem de uma aplicação multimídia e sua posterior execução no ambiente de emulação utilizado como estudo de caso.

O Ambiente para Especificação de Aplicações Multimídia com Suporte de Qualidade de Serviço, WSE, possui características pertinentes que o tornam ideal para a descrição de sistemas capazes de representar com exatidão os principais componentes envolvidos numa aplicação multimídia distribuída, além de poder descrever características específicas de cada componente.

O desenvolvimento de um modelo de aplicação multimídia distribuída acontece no ambiente WSE, primeiramente, pela escolha do tipo de aplicação a ser modelada (video on demand, audio on demand, video conference, audio conference, new application) e da qualidade de serviço a ser mantida no sistema (high, average, low, any). Nesse estudo de caso, foram escolhidas para a classe de aplicação “video on

demand" (vídeo sob demanda) e para requisitos de Qualidade de Serviço "high" (alta). A figura 18 ilustra essas opções.

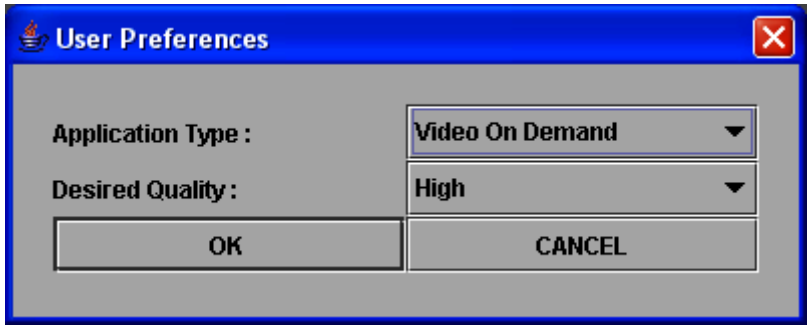


Figura 18: Caixa de diálogos referente a classe de aplicação e a Qualidade de Serviço exigidos na camada de usuário.

É necessária a escolha do tipo de aplicação e da QoS a ser aplicada pois cada aplicação possui valores mínimos e máximos diferentes para cada parâmetro, pois a qualidade "high" para uma aplicação de vídeo sob demanda, necessita que a largura de banda seja maior do que para uma aplicação de videoconferência que também exija qualidade "high". Nesse ponto, fica definida a QoS que será aplicada no nível do usuário, isto é, a qualidade pertinente a aplicação que será evidenciada pelo cliente, como taxa de bits e quadros por segundo em uma transmissão de vídeo e taxa de amostra e bits por amostra em transmissão de áudio.

Após a escolha do tipo de aplicação e da QoS exigida, o ambiente WSE apresenta a descrição do modelo da aplicação multimídia distribuída de forma visual.

Essa representação hierárquica é uma capacidade do editor de representar por categorias cada um dos componentes, podendo ser grupo, hardware ou software. O grupo é descrito pelos componentes de cor azul, o hardware pelos de cor verde e o software pelos componentes de cor amarela. É necessário ressaltar que cada componente é representado por uma figura geométrica, sendo que o retângulo, losango, círculo e retângulo arredondado representam servidor, gateway, peer e cliente, respectivamente.

O grupo é uma forma de apresentar componentes que formam um cluster, ou um conjunto de componentes que desempenham o mesmo papel na modelagem do sistema multimídia. Já o hardware apresenta o conteúdo físico das

máquinas envolvidas na modelagem, como largura de banda, memória, etc., enquanto o software apresenta o sistema operacional e toda a parte que será consumida pelo conteúdo lógico da máquina descrita no hardware.

Em cada grupo, obrigatoriamente, tem-se que apresentar ao menos um componente hardware e para cada hardware ao menos um componente software. Dessa forma fica estruturada a representação hierárquica dos componentes na modelagem, garantindo facilidade no entendimento do modelo, bem como na própria geração do código HQML. Nesse estudo de caso, foram escolhidos os componentes que descrevem um grupo de servidores e um grupo de clientes, como descrito na Figura 19. Em cada componente grupo escolhido, foi ainda inseridos um componente hardware e um componente software, como se a aplicação de vídeo sob demanda fosse composta por um servidor e um cliente.

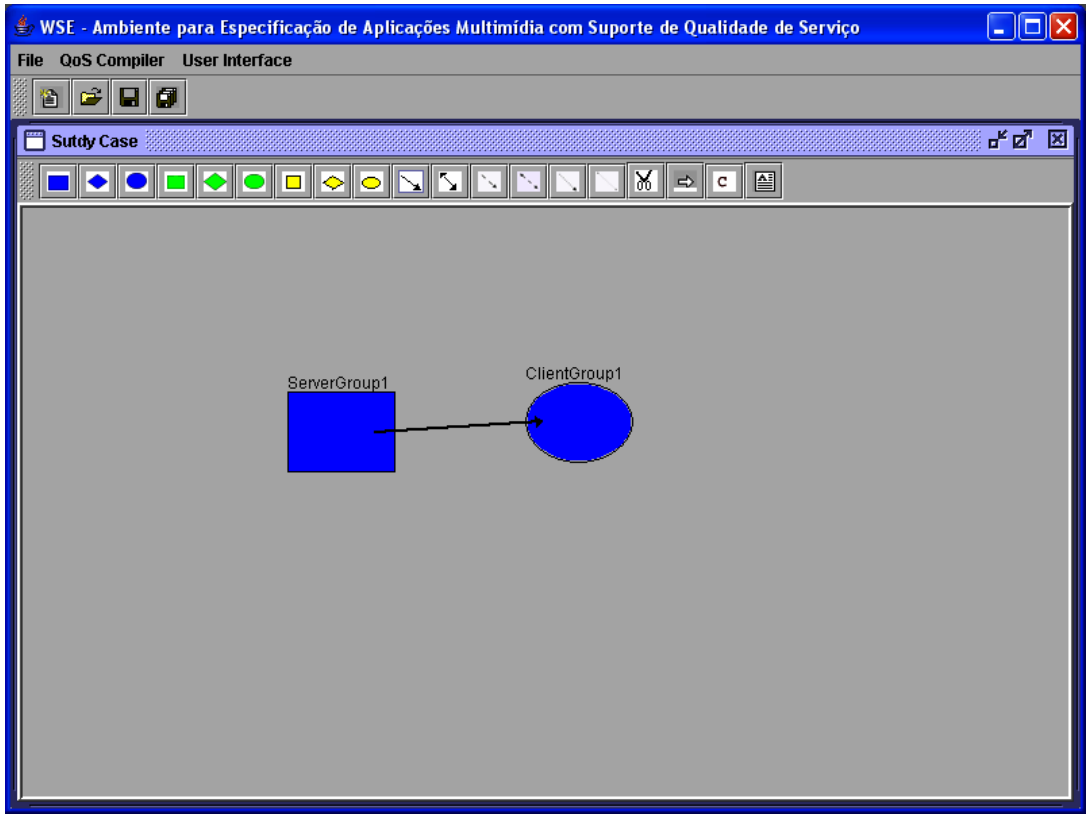


Figura 19: Descrição de uma aplicação de vídeo sob demanda no Editor Visual de QoS Hierárquico.

Após a descrição do modelo no editor, o próximo passo é o preenchimento das caixas de diálogo dos componentes existentes no modelo. Cada

componente com exceção do componente do tipo grupo contém caixas de diálogos a serem preenchidas com informações pertinentes ao sistema que será modelado, como largura de banda, capacidade de memória RAM, capacidade do disco rígido, sistema operacional, arquitetura da máquina (Sun, Pentium, etc.), e muitas outras informações úteis para o funcionamento do sistema multimídia. As caixas de diálogo preenchidas referentes aos componentes hardware e software do cliente, e hardware e software do servidor ficaram como descrito na Figura 20, sendo as caixas de diálogo referentes ao componente hardware aplicadas à Camada de Recurso e as referentes ao software à Camada de Aplicação.

Figura 20: (a) Hardware do Servidor, (b) Software do Servidor, (c) Hardware do Cliente e (d) Software do Cliente.

Dentre essas informações, destacam-se as que são inseridas na caixa de diálogo Client Software Input Dialog, pois é nessa caixa de diálogo que são definidos os

parâmetros de QoS e as regras de adaptação pertinentes a cada parâmetro, além da ordem execução de cada regra, se necessário. Nesse ponto é possível inserir qualquer parâmetro que precisa ser monitorado pela aplicação, como padrão, estão disponíveis os parâmetros atraso (delay), variação do atraso (jitter) e loss (perda de pacotes), porém com a possibilidade de inserção/criação de qualquer parâmetro que o desenvolvedor da aplicação julgar pertinente ao sistema.

As informações condizentes com a QoS e as regras de adaptação são definidas na Camada de Aplicação, isto é, dependendo da variação dos parâmetros, aplica-se uma regra referente ao mesmo, de acordo com a prioridade estabelecida para cada parâmetro. Essas regras, na realidade, são ações, ou codinomes para ações que compreendem a aplicação de filtros, em tempo real, na mídia que vai ser transmitida. Neste estudo de caso, esses filtros são conversões feitas em uma streaming, podendo variar desde a diminuição da taxa de bits, o número de quadros por segundo, até a retirada do áudio de um vídeo. Esses filtros podem ser encontrados em programas e bibliotecas como o Ffmpeg [47], o Mencoder [45] e o Mplayer [45].

Após a edição de todo o modelo, o mesmo passa por uma checagem de consistência por meio do Verificador de Consistência para validar esse modelo a fim de garantir que não exista nenhuma inconsistência, como maior largura de banda no software do cliente do que no hardware do cliente, além de incompatibilidades verificadas pela gramática ConfigG.

Se o modelo não tiver inconsistências, o mesmo está pronto para a geração do código HQML referente ao que foi descrito no modelo. No HQML todo o modelo fica representado por tags que identificam todas as suas características, desde a definição do tipo de aplicação e QoS adotados, até os recursos de hardware que serão consumidos pelo sistema multimídia modelado. Para que o HQML pudesse suportar as políticas (regras) de adaptação, bem como os parâmetros e uma possível (re) negociação da QoS, foram feitas modificações na sua DTD para que novas tags fossem criadas para cumprir essa finalidade. É baseado no código HQML que será feito todo o mapeamento da aplicação multimídia distribuída, da Camada de Usuário para a de Aplicação e da Camada de Aplicação para a Camada de Recurso. O código HQML gerado por esse estudo de caso está disponível no Apêndice D.

Esse código HQML gerado é responsável por conter todas as informações pertinentes ao modelo do sistema multimídia, para a partir daí, haver um mapeamento para uma aplicação capaz de agregar todas as informações e utilizá-las conforme descrito em suas características. Ao implementar uma aplicação que utilize as informações descritas no código HQML, esse código é base fidedigna para implementação e deve ser consultado a fim de garantir o que foi descrito na modelagem. Essa consulta ocorre no momento em que é iniciada uma aplicação. Neste momento, é chamado um "executor" que tem a capacidade de ler e entender os parâmetros descritos no código HQML. Dessa forma, o executor detém a garantia de que os parâmetros foram interpretados de maneira correta, e que os mesmos serão passados para a aplicação, a fim de serem utilizados na negociação e manutenção da QoS. O executor funciona baseado em um analisador que reconhece as tags presentes no arquivo HQML e captura seus valores. Assim, esses valores formarão uma tabela no proxy que designa o filtro correto, por meio de comandos, dependendo do intervalo dos valores dos parâmetros.

Foi escolhido o modelo de implementação da classe de aplicação Vídeo Sob Demanda, descrito na seção 5.6 do capítulo anterior.

Por meio da emulação do Nist Net, os parâmetros são medidos e, de acordo com a variação dos mesmos, mudanças na qualidade dos vídeos são notadas devido à adoção de filtros determinados para cada faixa de valores dos parâmetros. O Nist Net é "alimentado" por meio de um *script* que contém os comandos a serem executados com os valores para os parâmetros de QoS que desejam ser medidos.

Os filtros utilizados encontram-se na Tabela 6 no Apêndice E e variam de acordo com o parâmetro e com o seu respectivo valor. É interessante ressaltar os valores alcançados em relação ao tamanho do arquivo original na coluna Porcentagem. Esses valores correspondem aos tamanhos dos arquivos após a aplicação do filtro. Portanto, é possível obter um arquivo com pouco mais de 5% do tamanho do arquivo original se o filtro aplicado tirar o vídeo e diminuir a qualidade do áudio a 32 kbits por segundo. Isso indica que o número de pacotes a serem enviados para o cliente diminui, o que favorece a aplicação quando há uma largura de banda delimitada e quando a perda de pacotes está alta.

Os resultados obtidos ao final de todo o processo são explicitados através das imagens presentes na Figura 21.

Na Figura 21.a é mostrado um quadro do vídeo original, isto é, sem a aplicação de nenhum filtro. Já nas Figuras 21.b e 21.c são mostrados quadros após a utilização dos filtros que diminuem a qualidade do vídeo pela metade e o de pior qualidade, respectivamente. Pode ser percebido que esses quadros possuem qualidade pior devido estarem mais quadriculados que o original. Isso ocorreu porque os filtros que foram aplicados diminuem a taxa de amostras do vídeo, causando o efeito chamado pixellation [1] em que pixels distintos representados por retângulos pequenos são notados na imagem.

Esses filtros foram utilizados baseando-se na Tabela 5.

Parâmetro	Faixa de Valores	Filtro
Perda	10 – 20%	qscale 8 ab 96
	20 – 24%	qscale 10 ab 96
	25 – 30%	qscale 15 ab 64
	31 – 35%	qscale 30 ab 32
	36 – acima	inaceitável
Largura de Banda	10 – 50 Mbits	qscale 5 ab 96
	50 – 100 Mbits	qscale 30 ab 32
	100 – acima	inaceitável
Delay	0 – 10000 ms	Ignorado
Jitter	0 – 10000 ms	Ignorado

Tabela 5: Valores dos parâmetros e os respectivos filtros aplicados.

É válido ressaltar que o vídeo original tinha por volta de 3134 KB e conforme a aplicação de cada filtro, seu tamanho final era diminuído, o que pode ser observado pelos dados da Tabela 6 no Apêndice D. Especificamente nas aplicações dos filtros da Tabela 5 para a solução do aumento da perda de pacotes, os tamanhos finais do vídeo foram:

- qscale 8 ab 96 = 2030KB;
- qscale 10 ab 96 = 1754 KB;
- qscale 15 ab 64 = 1212KB;

- qscale 30 ab 32 = 674KB.

Pode ser percebido que a diminuição do número de quadros contribui para que o vídeo continuasse a sua reprodução, porém, isso é inversamente proporcional a qualidade do vídeo, pois, ao diminuir o número de quadros com a aplicação dos filtros, a qualidade caiu causando deformações na imagem e aumento de ruídos no áudio. Os valores dos tamanhos do vídeo após a aplicação dos filtros variou de, aproximadamente, 64% a 21% em relação ao tamanho original, respectivamente qscale 8 ab 96 e qscale 30 ab 32.

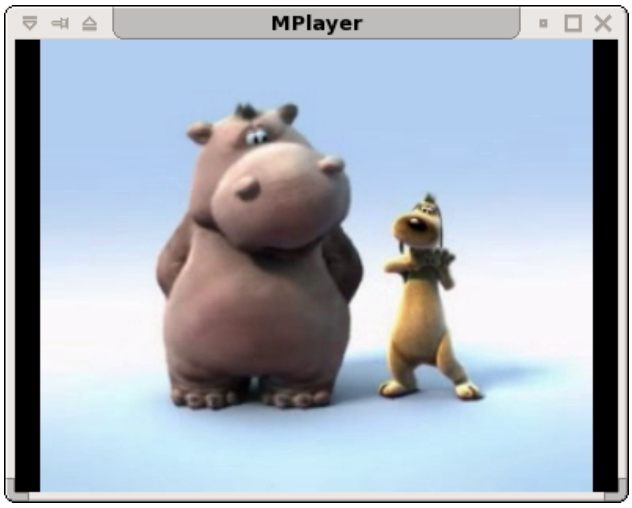
O mesmo acontece quando ocorre a aplicação de filtros para conter os problemas com relação a largura de banda. Neste caso, os tamanhos finais do vídeo após a aplicação dos filtros foram:

- qscale 5 ab 96 = 2790KB;

- qscale 30 ab 32 = 674KB.

Esses tamanhos correspondem a, aproximadamente, 89% e 21% do tamanho original, sendo este último filtro considerado o que deixa o vídeo pior, porém ainda com a possibilidade de se ter vídeo e áudio, mesmo com uma degradação dos mesmos.

Existe ainda a possibilidade de aplicação de filtros que diminuam ainda mais o número de quadros do vídeo, acarretando em arquivos com pouco mais de 5% do tamanho original, porém estes não foram considerados na Tabela 5 por tirarem o vídeo ou o áudio completamente.



(a)

(b)

(c)

Figura 21: Imagens com aplicação de filtros. (a) Imagem original. (b) Imagem com aplicação do filtro qscale 15 ab 64. (c) Imagem com aplicação do filtro qscale 30 ab 32.

Pode ser percebido que não foram aplicados filtros em relação ao atraso e a variação do atraso, pois não foi notada nenhuma diferença no comportamento do vídeo na variação dos valores dos mesmos desde zero até 10000 ms. Portanto, os valores relevantes ficaram por conta da largura de banda e da perda de pacotes.

7 – Conclusões e Trabalhos Futuros

Esse trabalho descreve um ambiente para especificação de aplicações multimídia distribuída com detalhamento da Qualidade de Serviço (WSE) exigida, permitindo especificações desses sistemas multimídia de forma hierárquica e restrições de QoS também de forma hierárquica nas Camadas de Usuário, Aplicação e Recurso. Não há na literatura tal abordagem sem restrições a arquitetura e a classes de aplicações multimídia. Sendo as principais vantagens do WSE:

- possibilidade de ilustrar qualquer aplicação multimídia distribuída por meio do Editor Visual de QoS Hierárquico, sem nenhuma restrição;

- ainda no Editor, o usuário pode estabelecer parâmetros de QoS e regras para adaptação/(re)negociação da aplicação multimídia distribuída, caso seja necessário;

- a gramática ConfigG que valida de maneira automática a descrição feita pelo Editor, deixando a modelagem de aplicações multimídia distribuída extremamente rápida, haja vista que não é necessário nenhuma implementação;

- o código HQML obtido por meio do Gerador HQML, que ilustra toda a configuração descrita pelo Editor Visual de QoS Hierárquico, bem como os parâmetros de QoS e as regras de adaptação. Este código é uma documentação da especificação do sistema multimídia em linguagem de marcação que descreve todo o sistema;

- o Executor HQML que é o responsável por verificar a documentação HQML obtida pelo Gerador HQML e configurar um sistema multimídia qualquer, seja este um ambiente de emulação, um framework, ou um esquema de aplicação pré-definido.

O ambiente WSE avalia as aplicações multimídia por meio de um emulador, trazendo ao desenvolvedor resultados experimentais da aplicação, o que facilita o desenvolvimento de tais aplicações haja vista a rapidez na aquisição de resultados prévios.

Para a validação de resultados foi realizado um estudo de caso utilizando uma aplicação de vídeo sob demanda como aplicação multimídia distribuída. Como resultado experimental, foi constatado o comportamento esperado do ambiente, bem como da aplicação de vídeo sob demanda, haja vista que todos os módulos realizaram os seus papéis com sucesso, e o vídeo efetivamente sofreu alterações como adaptação às regras descritas no Editor. A adaptação sofrida pelo vídeo e a (re)negociação da sua QoS são fatores positivos, pois minimiza o trabalho de um desenvolvedor de aplicações multimídia devido fato de ele não ter que implementar todo o sistema para saber se seus resultados serão válidos ou não, basta utilizar o ambiente WSE com o Emulador NistNet para tanto.

Em contrapartida ajustes devem ser feitos no WSE de maneira que mais aplicações multimídia distribuídas possam ser implementadas, haja vista que somente aplicações de vídeo sob demanda são possíveis de serem validadas. Por isso, como trabalhos futuros, é proposto um mapeamento automático de uma aplicação multimídia descrita no ambiente WSE para o emulador Nist Net, ou para outros ambientes reais sem interferências. Além disso, a implementação de Executores para outras classes de aplicação multimídia, ou mapeamento da especificação feita no ambiente WSE para arquiteturas existentes.

Referências Bibliográficas

[1] Lu, G. “Communication and Computing for Distributed Multimedia Systems”. Artech House, 1996.

[2] Gu, X.; Nahrstedt, K.; Yuan, W.; Wichadakul, D.; Xu, D. “An XML-based QoS Enabling Language for the Web”, Journal of Visual Language and Computing (Special Issue on Multimedia Languages for the Web), Publicação Acadêmica, 2001.

[3] Hermann, A. C. “Qualidade de Serviço em Aplicações Multimídia Distribuídas”, Programa de Pós-Graduação em Ciência da Computação, 3ª Semana Acadêmica do CPG-CC, Porto Alegre, 1998.

[4] Lunardi, S. C. “Integração e Qualidade de Serviço na Internet”, Porto Alegre, Relatório de Trabalho, 1999.

[5] Hansen, G. “Quality of Service (QoS)”, Objects Services and Consulting, Inc, Janeiro de 1997. Disponível em http://www.objs.com/survey/QoS.htm. Acessado em 22 de julho de 2004.

[6] http://www.tticom.com/atmglosy/atmlex.htm#qos, visualizado em 22 de julho de 2004.

[7] Shenker, S.; Partiridge, C.; Guerin, R. "Sprecification of Guaranteed Quality of Service". RFC 2212, 1997.

[8] Vogel, L.; Kerherve, B.; Bochmann, G.; Gecse, J. "Distributed Multimedia and QoS: A Survey", IEEE Multimedia, p. 10-19, 1995.

[9] Wolf, L. C.; Griodz, C.; Steirmetz, R. "Multimedia communications", Proceedings of IEEE, vol 85, , nº 12, p. 1915-1933, 1997.

[10] Li, V. O. K.; Liao, W. "Distributed Multimedia Systems". Proceedings of IEEE, p. 1061-1108, 1997.

[11] Dobson, G. "Quality of Service: a Service-Centric Perspective". Computer Department, Lancaster University, 2004.

[12] Bernet, Y. et al. "A Framework for Differentiated Services". Internet Draft, Disponível em: http://www.ietf.org/proceedings/98aug/I-D/draft-ietf-diffserv-framework-00.txt. Acessado em: 15/07/2004.

[13] Nahrstedt, K.; Steinmetez, R. "Resource Managemente in Networket Multimedia Systems". IEEE Computer, pags. 52-63, 1995.

[14] Willrich, R. "Dados Multimídia". Disponível em:

http://www.sidie.nurcad.ufsc.br/treinamento/cap2.pdf. Acesso em: 15 de Julho de 2004.

[15] Gopal, P. M.; Wong, J. W.; Majithia, J. C., "Analysis of Playout Strategis for Voice Transmission using Packet Switching Techniques," Preformance Evaluation, vol. 4, pp. 11-18, 1984.

[16] Hehmann, D.; Salmony, M.; Stuttgen, H., "Transport Services for Multimedia Applications on Broadband Networks," Computer Communications, vol. 13, nº 4, pp. 197-203, 1990.

[17] ITU "ITU-T Study Group 12 – Workshop on QoS and user-perceived transmission quality in evolving networks". ITU Telecommunication Standardization Sector, 2001.

[18] Nave, C. R. HyperPhysics. Department of Physics and Astronomy. Georgia State University, 2000. Disponível em: http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/adc.html. Acessado em: 25/07/2004.

[19] Nave, C. R. HyperPhysics. Department of Physics and Astronomy. Georgia State University, 2000. Disponível em: http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/dac.html. Acessado em: 25/07/2004.

[20] Goulart, C. C.; Nogueira, J. M. S.; Neufeld, G. "A Scheme for Dynamic QoS Renegotiation at Intermediate Nodes". Em Wanderley Lopes de Souza and Rogério Drummond, editores, Anais do 15o. Simpósio Brasileiro de Redes de Computadores (SBRC '97), pages 383-398, São Carlos, SP, 1997. UFSCar.

[21] Federal Standart 1037C: Glossary of Telecommunications Terms, http://www.its.bldrdoc.gov/fs-1037, visualizado em 26/08/2003.

[22] Lunardi, Sediane C. "Integração e Qualidade de Serviço na Internet", Porto Alegre, 1999.

[23] Jin, J.; Nahrstedt, K. "Classification and Comparison of QoS Specification Languages for Distributed Multimedia Applications", Relatório Técnico, University of Illinois at Urbana-Champaign, Illinois, EUA, 2002.

[24] Nahrstedt, K.; Smith, J. M. "The QoS Broker". IEEE Multimedia Magazine, 2(1):53–67, 1995.

[25] Steinmetz, R.; Nahrstedt, K. "Multimedia: Computing, Communications and Applications". Prentice Hall, 1995.

[26] Nishio, N.; Tokuda, H. "Simplified Method for Session Coordination Using Multi-level QoS Specification and Translation". Em Fifth International Workshop on Quality of Service (IWQoS'97), Nova Iorque, EUA, 1997.

[27] Roscoe, T.; Bowen, G. "Script-driven Packet Marking for Quality of Service Support in Legacy Applications". In Proceedings of SPIE Conference on Multimedia Computing and Networking 2000, San Jose, CA, 2000.

[28] Levy , J. Y.; Ousterhout, J. K.; Welch, B. B.. "The Safe-Tcl Security Model". Relatório Técnico TR-97-60, Sun Microsystems Laboratories, 1997.

[29] Campbell, A. T. A Quality of Service Architecture. PhD thesis, Computing Department, Lancaster University, 1996.

[30] Florissi, P. QoSME: QoS Management Environment. PhD thesis, Department of Computer Science, Columbia University, 1996.

[31] Nahrstedt, K.; Li, B. "Dynamic Reconfiguration for Complex Multimedia Applications". In Proceedings of IEEE International Conference on Multimedia Computing and Systems (IEEE ICMCS '99), Florence, Itália, 1999.

[32] World Wide Web Consortium. eXtensible Markup Language. http://www.w3c.org/XML/, visualizado em 10/07/2004.

[33] Exposito, E.; Gineste, M.; Peyrichou, R.; Senac, P.; Diaz, M. "XqoS: XML-Based QoS Specification Language". 2002.

[34] Mendhekar, A.; Maeda, C.; Lopes, C. V.; Loingtier, J.; Irwin, J.; Kiczales, G.; Lamping, J. "Aspect-Oriented Programming". In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finlândia, Springer-Verlag LNCS 1241, 1997.

[35] Zinky, J. A.; Loyall, J. P.; Schantz, R. E.; Bakken, D. E. "Specifying and Measuring Quality of Service in Distributed Object Systems". In Proceedings of ISORC'98, Kyoto, Japão, 1998.

[36] Bakken, D. E.; Zinky, J. A.; Schantz, R. D. "Architectural Support for Quality of Service for CORBA Objects". Theory and Practice of Object Systems, 1997.

[37] Schantz, R. E.; Zinky, J. A.; Karr, D. A.; Vanegas, R.; Loyall, J. P.; Bakken, D. E.; Anderson, K. R. "QoS Aspect Languages and Their Runtime Integration". Lecture Notes in Computer Science, Springer - Verlag, 1511, 1998.

[38] Frolund, S.; Koistinen, J. "QML: A Language for Quality of Service Specification". Technical Report HPL-98-10, HP Laboratories, 1998.

[39] Gu, X.; Nahrstedt, K. "An Event-Driven, User-Centric, QoS-aware Middleware Framework for Ubiquitous Multimedia Applications". *Proc.of 9th ACM Multimedia (Multimedia Middleware Workshop)*, 2001.

[40] http://www.htmlstaff.org/xml/xml71.php, visualizado em 12/05/2003.

[41] Gu, X.; Wichadakul, D.; Nahrstedt, K.. "Visual Quality of Service Programming Environment for Ubiquitous Multimedia Services", em Proceedings of IEEE International Conference on Multimedia and Expo 2001 (ICME2001), Tókio, Japão, 2001.

[42] Kon, F.; Campbell, R.; Mickunas, M. D.; Nahrstedt K.; Ballesteros, F. J. "2K: A Distributed Operating System for Dynamic Heterogeneous Environments". 9th IEEE International Symposium on High Performance Distributed Computing. Pittsburgh, 2000.

[43] Ferrucci, F.; Pacini, G.; Satta, G. "Symbol-Relation Grammars: A Formalism for Graphical Languages". Information and Computation, 131, 1996.

[44] Gu, X. "Visual Quality of Service Programming Environment for Distributed Heterogeneus Systems". University of Illinois, 2001.

[45] Mplayer – The Movie Player. Disponível em: http://www.mplayerhq.hu. Acessado em 22/07/2004.

[46] The Apache Software Foundation. Disponível em: http://www.apache.org/. Acessado em: 22/07/2004.

[47] http://ffmpeg.sourceforge.net/. Acessado em 22/07/2004.

[48] Silva, L. C. "Um Framework para Adaptação de Aplicações Multimídia em Ambientes de Redes Sem Fio". Dissertação de Mestrado. Universidade Federal de São Carlos, São Carlos, 2002.

[49] http://snad.ncsl.nist.gov/itg/nistnet/. Acessado em 22/07/2004.

[50] Carson, M.; Santay, Darrin. "Nist Net – A Linux-based Network Emulation Tool". National Institute of Standarts and Technology, 2002.

[51] Gu, X.; Xu, D.; Wichadakul, D.; Nahrstedt, K. "QoSTalk: A Visual Quality of Service Programming Environment", Illinois Computer Affiliates Program (ICAP) Workshop, 2000.

Apêndice A – Código do Proxy

Principais classes referentes ao componente Proxy citado no estudo de caso descrito no capítulo 6.

Classe Connection

```
package proxy;

import java.net.*;
import java.io.*;

public class Connection extends Thread {
  //--------------------------------------------------------------------------
  private Socket s;
  private Socket sserver;
  private Server server;
  private DataInputStream is;
  private DataOutputStream os;
  private DataInputStream serveris;
  private ProxyTunnel server2client;
  private ProxyTunnel server2filter;
  private ProxyTunnel filter2client;
  private Process process;
  private OutputStream filteros;
  private InputStream filteris;
  private InputStream filtererr;
  //--------------------------------------------------------------------------
  public Connection(Socket client, Server server) throws Exception {
    this.s = client;
    this.server = server;
    this.is = new DataInputStream( s.getInputStream() );
```

```
    this.os = new DataOutputStream( s.getOutputStream() );
    this.start();
  }
  //--------------------------------------------------------------------------
  public void run() {
    DataOutputStream serveros = null;
    try {
      byte bb[] = new byte[65536];
      while ( s.isConnected() ) {
        // Reading client http Header
        int total = is.read(bb);
        if (total<0) throw new Exception("Fim cliente!");
        String header = new String(bb, 0, total, "ISO-8859-1");;
        // Finding host to connect to
        int hi = header.indexOf("Host") + 6;
        int he = header.indexOf("\r", hi);
        String host = header.substring(hi, he);
        // Connecting to the host to do the proxy
        sserver = new Socket(host, 80);
        serveros = new DataOutputStream(sserver.getOutputStream());
        serveris = new DataInputStream(sserver.getInputStream());
        header = header.replaceAll("Proxy-Connection: keep-alive", "Connection: close");
        System.out.println("Pedido de " + s.getInetAddress() + " para " +
sserver.getInetAddress() +
                           " com tamanho " + total + "(" + header.length() + ")");
        serveros.writeBytes(header);

        //direto
        server2client = new ProxyTunnel(1, "direto", serveris, os);

        // Separar e enviar cabeçalho sem filtrar
/*
        int c1=0,c2=0,c3=0,c4=0;
        String h = "";
        while (c1!='\r' || c2!='\n' || c3!='\r' || c4!='\n') {
          c1 = c2;
          c2 = c3;
          c3 = c4;
          c4 = serveris.read();
          h += (char)c4;
        }
        os.writeBytes(h);
        System.out.println(h);
*/
        while(isActive()) {
          Thread.sleep(500);
          server2client.join();
        }
```

```
        server.removeConnection(this);
      }
    } catch (Exception e) { e.printStackTrace(); }
  }

//============================================================
===
  public boolean isActive() {
    return !sserver.isClosed();
  }

//============================================================
===
  public void setFilter(String filter) throws java.io.IOException {
    if ( server2filter != null && filter2client != null ) {
      server2filter.interrupt();
      filter2client.interrupt();
      filteris.close();
      filteros.close();
      filtererr.close();
      process.destroy();
    } else
      server2client.interrupt();
    process = Runtime.getRuntime().exec(filter);
    filteros = process.getOutputStream();
    filteris = process.getInputStream();
    filtererr = process.getErrorStream();
    server2filter = new ProxyTunnel(1, "1", serveris, filteros);
    filter2client = new ProxyTunnel(1, "2", filteris, os);
    server2client = server2filter;
    //consumir err
    new ProxyTunnel(1, "3", filtererr, new ByteArrayOutputStream(1024));
  }

//============================================================
===
  public String toString() {
    return s.getInetAddress().getHostAddress();
  }

//============================================================
===
}
```

Classe ProxyTunnel

```
package proxy;
```

```
import java.net.Socket;
import java.io.InputStream;
import java.io.OutputStream;

public class ProxyTunnel extends Thread {
 //---------------------------------------------------------------------------
 private Socket proxy;
 private OutputStream clientos;
 private InputStream proxyis;
 private boolean adapt = false;
 private String name;
 private int timeToSleep;
 //---------------------------------------------------------------------------
 public ProxyTunnel(int timeToSleep, String name, InputStream proxyis, OutputStream
os) {
   this.timeToSleep = timeToSleep;
   this.name = name;
   this.proxyis = proxyis;
   this.clientos = os;
   this.start();
 }
 //---------------------------------------------------------------------------
 public void run() {
   try {
     byte bb[] = new byte[65536];
     while (true) {
       int iii = proxyis.read(bb);
       if (iii < 0)
         throw new Exception("Server acabou!");
         clientos.write(bb, 0, iii);
         System.out.println(name + " > enviou (" + iii + ")");
       clientos.flush();
       Thread.sleep(timeToSleep);
     }
   } catch (InterruptedException e) {
     System.out.println("Parei " + name);
   } catch (Exception e) {
     e.printStackTrace();
     if ( !this.isInterrupted() ) {
       try{ clientos.close(); } catch (Exception e1) { e.printStackTrace(); }
       try{ proxyis.close(); } catch (Exception e1) { e.printStackTrace(); }
     }
   } finally {
   }
 }
 //---------------------------------------------------------------------------
}
```

Classe Server

```
package proxy;

import java.net.*;
//import java.io.*;
import java.util.Vector;
import java.util.Enumeration;

public class Server {

//============================================================================
  private Vector connections = new Vector();

//============================================================================
  public static void main(String args[]) {
    try {
      if (args.length != 1) {
        System.out.println("Usage: Server port");
        return;
      }
      Server server = new Server();
      ServerSocket ss = new ServerSocket(Integer.parseInt(args[0]));
      System.out.println("Server activated at port " + ss.getLocalPort());
      ControlServer cs = new ControlServer(ss.getLocalPort()+1, server);
      while (true) {
        try {
          server.addConnection(  new Connection( ss.accept(), server )  );
        } catch (Exception e) { e.printStackTrace(); }
      }
    } catch (Exception e) { e.printStackTrace(); }
  }

//============================================================================
  public Vector getConnections() {
    return connections;
  }
  //--------------------------------------------------------------------------
  public void addConnection(Connection connection) {
    connections.add(connection);
  }
  //--------------------------------------------------------------------------
  public void removeConnection(Connection connection) {
    connections.remove(connection);
```

```
  }

//=================================================================
===
}
```

Classe ControlServer

```
package proxy;

import java.net.ServerSocket;
import java.net.Socket;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Vector;
import java.util.Enumeration;

public class ControlServer extends Thread {

//=================================================================
===
  private ServerSocket ss;
  private Server server;

//=================================================================
===
  public ControlServer( int port, Server server ) throws java.io.IOException {
    this.server = server;
    this.ss = new ServerSocket(port);
    System.out.println("Control Server activated at port " + ss.getLocalPort());
    this.start();
  }

//=================================================================
===
  public void run() {
    while(true) {
      try {
        Socket s = ss.accept();
        System.out.println("Controller connected.");
        ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
        ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
        while (true) {
          System.out.println("receber commando");
          String command = ois.readUTF();
          if (command.equals("getConnections")) {
            System.out.println("getConnections");
            Vector v = new Vector();
```

```
          Enumeration e = server.getConnections().elements();
          while (e.hasMoreElements())
            v.add(e.nextElement().toString());
          oos.writeObject(v);
         } else if (command.equals("setFilter")) {
          System.out.println("setFilter");
          int index = ois.readInt();
          String filter = ois.readUTF();
          Connection c = (Connection) server.getConnections().elementAt(index);
          c.setFilter(filter);
         } else if (command.equals("exit")) {
          break;
         }
        }
      } catch (Exception e) { e.printStackTrace(); }
    }
  }

//=================================================================
===
}
```

Classe Controller

```
package proxy;

import java.net.Socket;
import java.io.*;
import java.util.Vector;

public class Controller {

//=================================================================
===
  public static void main(String[] args) {
    if (args.length != 2) {
      System.out.println("Usage: Controller serverAddress serverPort");
      return;
    }
    try {
      Socket s = new Socket(args[0], Integer.parseInt(args[1]));
      ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
      ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
      BufferedReader br = new BufferedReader( new InputStreamReader(System.in) );
      while(true) {
        System.out.print("Command (h for help): ");
        String command = br.readLine();
        if (command.equals("h")) {
```

```
        System.out.println("l - list connections");
        System.out.println("s - set filter");
        System.out.println("x - exit");
        System.out.println("");
      } else if (command.equals("l")) {
        oos.writeUTF("getConnections");
        oos.flush();
        Vector connections = (Vector)ois.readObject();
        for (int i=0; i<connections.size(); i++)
          System.out.println((i+1) + " - " + connections.get(i));
        System.out.println("");
      } else if (command.equals("s")) {
        System.out.print("Connection index: ");
        int index = Integer.parseInt(br.readLine());
        System.out.print("Filter command: ");
        String filter = br.readLine();
        oos.writeUTF("setFilter");
        oos.writeInt(index-1);
        oos.writeUTF(filter);
        oos.flush();
        System.out.println("");
      } else if (command.equals("x")) {
        oos.writeUTF("exit");
        oos.flush();
        break;
      }
    }
  } catch (Exception e) { e.printStackTrace(); }
 }

//=============================================================
===
}
```

Classe ControllerGUI

```
package controller;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class ControllerGUI extends JFrame {
 ObjectOutputStream oos;
 ObjectInputStream ois;
```

```
  JPanel jPanel1 = new JPanel();
  JScrollPane jScrollPane1 = new JScrollPane();
  JList jlConnections = new JList();
  JButton jbRefresh = new JButton();
  JButton jbExit = new JButton();
  JButton jbFilter = new JButton();
  JComboBox jcbFilter = new JComboBox();
  public ControllerGUI(String host, int port) {
    try {
      Socket s = new Socket(host, port);
      oos = new ObjectOutputStream(s.getOutputStream());
      ois = new ObjectInputStream(s.getInputStream());
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    if (args.length != 2) {
      System.out.println("Usage: Controller serverAddress serverPort");
      return;
    }
    ControllerGUI controllerGUI = new ControllerGUI(args[0],
Integer.parseInt(args[1]));
    controllerGUI.setSize(600,300);
    controllerGUI.setVisible(true);
  }

  private void jbInit() throws Exception {
    this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    jbRefresh.setText("Refresh");
    jbRefresh.addActionListener(new ControllerGUI_jbRefresh_actionAdapter(this));
    jbExit.setText("Exit");
    jbExit.addActionListener(new ControllerGUI_jbExit_actionAdapter(this));
    jbFilter.setText("Filter");
    jbFilter.addActionListener(new ControllerGUI_jbFilter_actionAdapter(this));
    this.getContentPane().add(jPanel1, BorderLayout.NORTH);
    jPanel1.add(jbRefresh, null);
    jPanel1.add(jbExit, null);
    jPanel1.add(jbExit, null);
    this.getContentPane().add(jScrollPane1, BorderLayout.CENTER);
    jScrollPane1.getViewport().add(jlConnections, null);
    jcbFilter.addItem("ffmpeg -y -qscale 5 -ab 96 -f mpeg -i - -f mpeg -");
    jcbFilter.addItem("ffmpeg -y -qscale 5 -ab 64 -f mpeg -i - -f mpeg -");
    jcbFilter.addItem("ffmpeg -y -qscale 5 -ab 32 -f mpeg -i - -f mpeg -");
```

```
  jcbFilter.addItem("ffmpeg -y -qscale 6 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 5 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 6 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 7 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 6 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 7 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 8 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 6 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 7 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 9 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 8 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 10 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 7 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 9 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 8 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 10 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 9 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 8 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 10 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 9 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 15 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 10 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 15 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 20 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 25 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 15 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 20 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 30 -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 25 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 15 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 20 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 30 -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 25 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 20 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 30 -ab 32 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 25 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -qscale 30 -an -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -vn -ab 96 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -vn -ab 64 -f mpeg -i - -f mpeg -");
  jcbFilter.addItem("ffmpeg -y -vn -ab 32 -f mpeg -i - -f mpeg -");
  jPanel1.add(jcbFilter, null);
  jPanel1.add(jbFilter, null);
  new ListUpdater(this);
 }
```

```
  void jbRefresh_actionPerformed(ActionEvent e) {
   try {
     int sel = jlConnections.getSelectedIndex();
     oos.writeUTF("getConnections");
     oos.flush();
     Vector connections = (Vector) ois.readObject();
     jlConnections.setListData(connections);
     jlConnections.setSelectedIndex(sel);
   } catch(Exception error) {
     error.printStackTrace();
   }
  }

  void jbExit_actionPerformed(ActionEvent e) {
   try {
     oos.writeUTF("exit");
     oos.flush();
     this.dispose();
   } catch(Exception error) {
     error.printStackTrace();
   }
  }

  void jbFilter_actionPerformed(ActionEvent e) {

    if (jlConnections.getSelectedIndex()<0)
      JOptionPane.showMessageDialog(null, "You may choose a connection in the list.",
"Alert", JOptionPane.PLAIN_MESSAGE);
    else {
      try{
        oos.writeUTF("setFilter");
        oos.writeInt(jlConnections.getSelectedIndex());
        oos.writeUTF(jcbFilter.getSelectedItem().toString());
        oos.flush();
      } catch(Exception error) {
        error.printStackTrace();
      }
    }
  }
}

class ListUpdater implements Runnable {
  ControllerGUI adaptee;

  public ListUpdater(ControllerGUI adaptee) {
    this.adaptee = adaptee;
    Thread thread = new Thread(this);
    thread.setDaemon(true);
```

```
    thread.start();
  }
  public void run() {
    while (true) {
      try { Thread.sleep(1000); } catch (Exception e) {}
      adaptee.jbRefresh_actionPerformed(null);
    }
  }
}

class ControllerGUI_jbRefresh_actionAdapter implements
java.awt.event.ActionListener {
  ControllerGUI adaptee;

  ControllerGUI_jbRefresh_actionAdapter(ControllerGUI adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jbRefresh_actionPerformed(e);
  }
}

class ControllerGUI_jbExit_actionAdapter implements java.awt.event.ActionListener {
  ControllerGUI adaptee;

  ControllerGUI_jbExit_actionAdapter(ControllerGUI adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jbExit_actionPerformed(e);
  }
}

class ControllerGUI_jbFilter_actionAdapter implements java.awt.event.ActionListener
{
  ControllerGUI adaptee;

  ControllerGUI_jbFilter_actionAdapter(ControllerGUI adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.jbFilter_actionPerformed(e);
  }
}
```

Apêndice B – Conjunto Completo de Regras de Produção da ConfigG

$\alpha \in$ {AudioServer, VideoServer, MediaServer, VoiceMailRecorder, TrackingServer, t}

$\beta \in$ {Prefetch, Tracker, Transcoder, FrameDropper, FrameFilter, Synchronization, Broadcast, t}

$\delta \in$ {AudioPlayer, VideoPlayer, MediaPlayer, MediaRecorder, AudioRecorder, VideoRecorder, VoiceMailRecorder, Prefetch, Tracker, RemoteControl, t}

$\gamma \in$ {AudioRecorder, VideoRecorder, MediaRecorder, VoiceMailRecorder, AudioPlayer, VideoPlayer, MediaPlayer, Syncronization, t}.

SP: Produções de símbolos

1. $S^0 \rightarrow <\{SC^2, CC^2\}, \{FixedLink(SC^2, CC^2)\}>$
2. $S^0 \rightarrow <\{SC^2, CC^2\}, \{mhl(SC^2, CC^2)\}>$
3. $S^0 \rightarrow <\{SC^2, CC^2\}, \{mul(SC^2, CC^2)\}>$
4. $S^0 \rightarrow <\{SC^2, GWC^2, CC^2\}, \{fl(SC^2, GWC^2), fl(GWC^2, CC^2)\}>$
5. $S^0 \rightarrow <\{SC^2, GWC^2, CC^2\}, \{fl(SC^2, GWC^2), mul(GWC^2, CC^2)\}>$
6. $S^0 \rightarrow <\{SC^2, GWC^2, CC^2\}, \{fl(SC^2, GWC^2), mhl(GWC^2, CC^2)\}$

7. $S^0 \rightarrow <\{GC^2, GC^3\}, \{fl(GC^2, GC^3), fl(GC^3, GC^2)\}>$
8. $S^0 \rightarrow <\{GC^2, GC^3\}, \{mul(GC^2, GC^3), mul(GC^3, GC^2)\}>$
9. $S^0 \rightarrow <\{GC^2, GC^3\}, \{mhl(GC^2, GC^3), mhl(GC^3, GC^2)\}>$
10. $S^0 \rightarrow <\{GC^2, GC^2\}, \{fl(GC^2, GC^2), fl(GWC^2, GC^2)\}>$
11. $S^0 \rightarrow <\{GC^2, GC^2\}, \{mul(GC^2, GC^2), mul(GWC^2, GC^2)\}>$
12. $S^0 \rightarrow <\{GC^2, GC^2\}, \{mhl(GC^2, GC^2), mhl(GWC^2, GC^2)\}>$
13. $SC^0 \rightarrow <\{SC^2, SC^3\}, 0\}>$
14. $SC^0 \rightarrow <\{SC^2, A^2\}, \{fl(A^2, SC^2)\}>$
15. $SC^0 \rightarrow <\{SC^2, A^2\}, \{fl(SC^2, A^2)\}>$
16. $SC^0 \rightarrow <\{SC^2, A^2\}, \{fl(SC^2, A^2), fl(A^2, SC^2)\}>$
17. $SC^0 \rightarrow <\{A^2\}, 0>$
18. $GWC^0 \rightarrow <\{GWC^2, GWC^3\}, 0>$
19. $GWC^0 \rightarrow <\{GWC^2, B^2\}, \{fl(B^2, GWC^2)\}>$
20. $GWC^0 \rightarrow <\{GWC^2, B^2\}, \{fl(GWC^2, B^2)\}>$
21. $GWC^0 \rightarrow <\{GWC^2, B^2\}, \{fl(GWC^2, B^2), fl(B^2, GWC^2)\}>$
22. $GWC^0 \rightarrow <\{B^2\}, 0>$
23. $CC^0 \rightarrow <\{CC^2, CC^3\}, 0>$
24. $CC^0 \rightarrow <\{C^2, CC^2\}, fl(C^2, CC^2)\}>$
25. $CC^0 \rightarrow <\{C^2, CC^2\}, mhl(C^2, CC^2)\}>$
26. $CC^0 \rightarrow <\{C^2, CC^2\}, mul(C^2, CC^2)\}>$
27. $CC^0 \rightarrow <\{C^2, CC^2\}, fl(CC^2, C^2)\}>$
28. $CC^0 \rightarrow <\{C^2, CC^2\}, fl(CC^2, C^2), fl(C^2, CC^2)\}>$
29. $CC^0 \rightarrow <\{C^2, CC^2\}, mul(CC^2, C^2)\ \}>$
30. $CC^0 \rightarrow <\{C^2, CC^2\}, mul(CC^2, C^2), mhl(C^2, CC^2)\}>$

31. $CC^0 \rightarrow <\{C^2, CC^2\}, mhl(CC^2, CC^3)\}>$

32. $CC^0 \rightarrow <\{C^2, CC^2\}, mhl(CC^2, C^2), mhl(C^2, CC^2)\}>$

33. $CC^0 \rightarrow <\{C^2\}, 0\}>$

34. $GC^0 \rightarrow <\{GC^2, GC^3\}, 0\}>$

35. $GC^0 \rightarrow <\{D^2, GC^2\}, \{fl(D^2, GC^2)\}>$

36. $GC^0 \rightarrow <\{D^2, GC^2\}, mhl(D^2, GC^2)\}>$

37. $GC^0 \rightarrow <\{D^2, GC^2\}, mul(D^2, GC^2)\}>$

38. $GC^0 \rightarrow <\{GC^2, D^2\}, mhl(D^2, GC^2)\}>$

39. $GC^0 \rightarrow <\{GC^2, D^2\}, mul(GC^2, D^2)\}>$

40. $GC^0 \rightarrow <\{GC^2, D^2\}, \{fl(GC^2, D^2), fl(D^2, GC^2)\}>$

41. $GC^0 \rightarrow <\{GC^2, D^2\}, \{mul(GC^2, D^2), mul(D^2, GC^2)\}>$

42. $GC^0 \rightarrow <\{GC^2, D^2\}, \{mhl(GC^2, D^2), mhl(D^2, GC^2)\}>$

43. $GC^0 \rightarrow <\{D^2\}, 0\}>$

44. $A^0 \rightarrow <\{AT^2\}, 0\}>$

45. $AT^0 \rightarrow <\{AT^2, AT^3\}, 0\}>$

46. $AT^0 \rightarrow <\{AT^2, \alpha^2\}, 0\}>$

47. $AT^0 \rightarrow <\{AT^2, \alpha^2\}, \{l(\alpha^2, AT^2)\}>$

48. $AT^0 \rightarrow <\{AT^2, \alpha^2\}, \{l(AT^2, \alpha^2)\}>$

49. $AT^0 \rightarrow <\{\alpha^2\}, 0\}>$

50. $B^0 \rightarrow <\{BT^2\}, 0\}>$

51. $BT^0 \rightarrow <\{BT^2, BT^3\}, 0\}>$

52. $BT^0 \rightarrow <\{BT^2, \beta^2\}, \{l(\beta^2, BT^2)\}>$

53. $BT^0 \rightarrow <\{BT^2, \beta^2\}, \{l(BT^2, \beta^2)\}>$

54. $BT^0 \rightarrow <\{BT^2, \beta^2\}, \{l(BT^2, \beta^2), l(\beta^2, BT^2)\}>$

55. $BT^0 \rightarrow <\{\beta^2\}, 0\}>$

56. $C^0 \rightarrow <\{CT^2\}, 0\}>$

57. $CT^0 \rightarrow <\{CT^2, CT^3\}, 0\}>$

58. $CT^0 \rightarrow <\{CT^2, \gamma^2\}, \{l(\gamma^2, CT^2)\}>$

59. $CT^0 \rightarrow <\{CT^2, \gamma^2\}, \{l(CT^2, \gamma^2)\}>$

60. $CT^0 \rightarrow <\{CT^2, \gamma^2\}, \{l(CT^2, \gamma^2), l(\gamma^2, CT)\}>$

61. $CT^0 \rightarrow <\{\gamma^2\}, 0\}>$

62. $D^0 \rightarrow <\{DT^2\}, 0\}>$

63. $DT^0 \rightarrow <\{DT^2, DT^3\}, 0\}>$

64. $DT^0 \rightarrow <\{DT^2, \delta^2\}, \{l(\delta^2, DT^2)\}>$

65. $DT^0 \rightarrow <\{DT^2, \delta^2\}, \{l(DT^2, \delta^2)\}>$

66. $DT^0 \rightarrow <\{DT^2, \delta^2\}, \{l(DT^2, \delta^2), l(\delta^2, DT^2)\}>$

67. $DT^0 \rightarrow <\{\delta^2\}, 0\}>$

RP: Produções de relação

R1. $fl(SC^0, CC^1) \rightarrow [13]\{fl(SC^2, CC^1), fl(SC^3, CC^1)\}$

R2. $mul(SC^0, CC^1) \rightarrow [13]\{mul(SC^2, CC^1), mul(SC^3, CC^1)\}$

R3. $mhl(SC^0, CC^1) \rightarrow [13]\{mhl(SC^2, CC^1), mhl(SC^3, CC^1)\}$

R4. $fl(SC^0, CC^1) \rightarrow [14]\{fl(SC^2, CC^1)\}$

R5. $mul(SC^0, CC^1) \rightarrow [14]\{mul(SC^2, CC^1)\}$

R6. $mhl(SC^0, CC^1) \rightarrow [14]\{mhl(SC^2, CC^1)\}$

R7. $fl(SC^0, CC^1) \rightarrow [15, 16, 17]\{fl(A^2, CC^1)\}$

R8. $mul(SC^0, CC^1) \rightarrow [15, 16, 17]\{mul(A^2, CC^1)\}$

R9. $mhl(SC^0, CC^1) \rightarrow [15, 16, 17]\{mhl(A^2, CC^1)\}$

R10. $fl(SC^0, GWC^1) \rightarrow [13]\{fl(SC^2, GWC^1), fl(SC^3, GWC^1)\}$

R11. $fl(SC^0, GWC^1) \rightarrow [14]\{fl(SC^2, GWC^1)\}$

R12. $fl(SC^0, GWC^1) \rightarrow [15, 16, 17]\{fl(A^2, GWC^1)\}$

R13. $fl(SC^0, A^1) \rightarrow [13]\{fl(SC^2, A^1), fl(SC^3, A^1)\}$

R14. $fl(SC^0, A^1) \rightarrow [14]\{fl(SC^2, A^1)\}$

R15. $fl(SC^0, A^1) \rightarrow [15, 16, 17]\{fl(A^2, A^1)\}$

R16. $fl(A^0, SC^1) \rightarrow [13]\{fl(A^1, SC^2), fl(A^1, SC^3)\}$

R17. $fl(A^1, SC^0) \rightarrow [14, 17]\{fl(A^1, A^2)\}$

R18. $fl(A^1, SC^0) \rightarrow [15, 16]\{fl(A^1, SC^2)\}$

R19. $fl(GWC^0, CC^1) \rightarrow [18]\{fl(GWC^2, CC^1), fl(GWC^3, CC^1)\}$

R20. $mul(GWC^0, CC^1) \rightarrow [18]\{mul(GWC^2, CC^1), mul(GWC^3, CC^1)\}$

R21. $mhl(GWC^0, CC^1) \rightarrow [18]\{mhl(GWC^2, CC^1), mhl(GWC^3, CC^1)\}$

R22. $fl(GWC^0, CC^1) \rightarrow [19]\{fl(GWC^2, CC^1)\}$

R23. $mhl(GWC^0, CC^1) \rightarrow [19]\{mhl(GWC^2, CC^1)\}$

R24. $mul(GWC^0, CC^1) \rightarrow [19]\{mul(GWC^2, CC^1)\}$

R25. $fl(GWC^0, CC^1) \rightarrow [20, 21, 22]\{fl(B^2, CC^1)\}$

R26. $mhl(GWC^0, CC^1) \rightarrow [20, 21, 22]\{mhl(B^2, CC^1)\}$

R27. $mul(GWC^0, CC^1) \rightarrow [20, 21, 22]\{mul(B^2, CC^1)\}$

R28. $fl(GWC^0, CC^1) \rightarrow [18]\{l(GWC^2, CC^1), l(GWC^3, GC^1)\}$

R29. $mul(GWC^0, CC^1) \rightarrow [18]\{mul(GWC^2, CC^1), mul(GWC^3, GC^1)\}$

R30. $mhl(GWC^0, CC^1) \rightarrow [18]\{mhl(GWC^2, CC^1), mhl(GWC^3, GC^1)\}$

R31. $fl(GWC^0, CC^1) \rightarrow [19]\{fl(GWC^2, GC^1)\}$

R32. $mul(GWC^0, CC^1) \rightarrow [19]\{mul(GWC^2, GC^1)\}$

R33. $mhl(GWC^0, CC^1) \rightarrow [19]\{mhl(GWC^2, GC^1)\}$

R34. $fl(GWC^0, CC^1) \rightarrow [20, 21, 22]\{fl(B^2, GC^1)\}$

R35. $mhl(GWC^0, CC^1) \rightarrow [20, 21, 22]\{mhl(B^2, GC^1)\}$

R36. $mul(GWC^0, CC^1) \rightarrow [20, 21, 22]\{mul(B^2, GC^1)\}$

R37. $fl(GWC^0, B^1) \rightarrow [18]\{fl(GWC^2, B^1), l(GWC^3, B^1)\}$

R38. $fl(GWC^0, B^1) \rightarrow [19]\{fl(B^3, B^1)\}$

R39. $fl(GWC^0, B^1) \rightarrow [20, 21, 22]\{fl(B^2, B^1)\}$

R40. $fl(B^1, GWC^0) \rightarrow [18]\{fl(B^1, GWC^2), l(B^1, GWC^3)\}$

R41. $fl(B^1, GWC^0) \rightarrow [19, 22]\{fl(B^1, B^2)\}$

R42. $fl(B^1, GWC^0) \rightarrow [20, 21]\{fl(B^1, GWC^2)\}$

R43. $fl(CC^0, C^1) \rightarrow [23]\{fl(CC^2, C^1), fl(CC^3, C^1)\}$

R44. $mhl(CC^0, C^1) \rightarrow [23]\{mhl(CC^2, C^1), mhl(CC^3, C^1)\}$

R45. $mul(CC^0, C^1) \rightarrow [23]\{mul(CC^2, C^1), mhl(CC^3, C^1)\}$

R46. $fl(CC^0, C^1) \rightarrow [24, 25, 26]\{fl(CC^2, C^1)\}$

R47. $mul(CC^0, C^1) \rightarrow [24, 25, 26]\{mul(CC^2, C^1)\}$

R48. $mhl(CC^0, C^1) \rightarrow [24, 25, 26]\{mhl(CC^2, C^1)\}$

R49. $fl(CC^0, C^1) \rightarrow [27, 28, 29, 30, 31, 32, 32, 33]\{fl(C^2, C^1)\}$

R50. $mhl(CC^0, C^1) \rightarrow [29, 30, 31, 32, 32, 33, 34, 35]\{mhl(C^2, C^1)\}$

R51. $mul(CC^0, C^1) \rightarrow [27, 28, 29, 30, 31, 32, 33]\{mul(C^2, C^1)\}$

R52. $fl(C^1, CC^0) \rightarrow [23]\{fl(C^1, CC^2), fl(C^1, CC^3)\}$

R53. $mhl(C^1, CC^0) \rightarrow [23]\{mhl(C^1, CC^2), mhl(C^1, CC^3)\}$

R54. $mul(C^1, CC^0) \rightarrow [23]\{mul(C^1, CC^2), mul(C^1, CC^3)\}$

R55. $fl(C^1, CC^0) \rightarrow [24, 25, 26, 33]\{fl(C^1, C^2)\}$

R56. $mul(C^1, CC^0) \rightarrow [24, 25, 26, 33]\{mul(C^1, C^2)\}$

R57. $mhl(C^1, CC^0) \rightarrow [24, 25, 26, 33]\{mhl(C^1, C^2)\}$

R58. $fl(C^1, CC^0) \rightarrow [27, 28, 29, 30 ,31 ,32]\ \{fl(C^1, CC^2)\}$

R59. $mhl(C^1, CC^0) \rightarrow [27, 28, 29, 30, 31, 32]$ $\{mhl(C^1, CC^2)\}$

R60. $mul(C^1, CC^0) \rightarrow [27, 28, 29, 30, 31, 32]$ $\{mul(C^1, CC^2)\}$

R61. $fl(D^1, GC^0) \rightarrow [34]$ $\{fl(D^1, GC^2), fl(D^1, GC^3)\}$

R62. $mhl(D^1, GC^0) \rightarrow [34]$ $\{mhl(D^1, GC^2), mhl(D^1, GC^3)\}$

R63. $mul(D^1, GC^0) \rightarrow [34]$ $\{mul(D^1, GC^2), mul(D^1, GC^3)\}$

R64. $fl(D^1, GC^0) \rightarrow [35, 36, 37, 43]$ $\{fl(D^1, D^2)\}$

R65. $mhl(D^1, GC^0) \rightarrow [35, 36, 37, 43]$ $\{mhl(D^1, D^2)\}$

R66. $mul(D^1, GC^0) \rightarrow [35, 36, 37, 43]$ $\{mul(D^1, D^2)\}$

R67. $fl(D^1, GC^0) \rightarrow [38, 39, 40, 41, 42]$ $\{fl(D^1, GC^2)\}$

R68. $mhl(D^1, GC^0) \rightarrow [38, 39, 40, 41, 42]$ $\{mhl(D^1, GC^2)\}$

R69. $mul(D^1, GC^0) \rightarrow [38, 39, 40, 41, 42]$ $\{mul(D^1, GC^2)\}$

R70. $fl(GC^0, D^1) \rightarrow [34]$ $\{fl(GC^2, D^1), fl(GC^3, D^1)\}$

R71. $mul(GC^0, D^1) \rightarrow [34]$ $\{mul(GC^2, D^1), mul(GC^3, D^1)\}$

R72. $mhl(GC^0, D^1) \rightarrow [34]$ $\{mhl(GC^2, D^1), mhl(GC^3, D^1)\}$

R73. $fl(GC^0, D^1) \rightarrow [35, 36, 37]$ $\{fl(GC^2, D^1)\}$

R74. $mhl(GC^0, D^1) \rightarrow [35, 36, 37]$ $\{mhl(GC^2, D^1)\}$

R75. $mul(GC^0, D^1) \rightarrow [35, 36, 37]$ $\{mul(GC^2, D^1)\}$

R76. $fl(GC^0, D^1) \rightarrow [38, 39, 40, 41, 42, 43]$ $\{fl(D^2, D^1)\}$

R77. $mhl(GC^0, D^1) \rightarrow [38, 39, 40, 41, 42, 43]$ $\{mhl(D^2, D^1)\}$

R78. $mul(GC^0, D^1) \rightarrow [38, 39, 40, 41, 42, 43]$ $\{mul(D^2, D^1)\}$

R79. $fl(GC^0, GWC^1) \rightarrow [34]$ $\{fl(GC^2, GWC^1), fl(GC^3, GWC^1)\}$

R80. $mul(GC^0, GWC^1) \rightarrow [34]$ $\{mul(GC^2, GWC^1), mul(GC^3, GWC^1)\}$

R81. $mhl(GC^0, GWC^1) \rightarrow [34]$ $\{mhl(GC^2, GWC^1), mhl(GC^3, GWC^1)\}$

R82. $fl(GC^0, GWC^1) \rightarrow [35, 36, 37]$ $\{fl(GC^2, GWC^1)\}$

R83. $mhl(GC^0, GWC^1) \rightarrow [35, 36, 37] \{mhl(GC^2, GWC^1)\}$

R84. $mul(GC^0, GWC^1) \rightarrow [35, 36, 37] \{mul(GC^2, GWC^1)\}$

R85. $mul(GC^0, GWC^1) \rightarrow [38, 39, 40, 41, 42, 43] \{fl(D^2, GWC^1)\}$

R86. $mhl(GC^0, GWC^1) \rightarrow [38, 39, 40, 41, 42, 43] \{mhl(D^2, GWC^1)\}$

R87. $mul(GC^0, GWC^1) \rightarrow [38, 39, 40, 41, 42, 43] \{mul(D^2, GWC^1)\}$

R88. $l(AT^0, \alpha^1) \rightarrow [45] \{mul(AT^2, \alpha^1), l(AT^3, \alpha^1)\}$

R89. $l(AT^0, \alpha^1) \rightarrow [46] \{l(AT^2, \alpha^1)\}$

R90. $l(AT^0, \alpha^1) \rightarrow [47, 48, 49] \{l(\alpha^2, \alpha^1)\}$

R91. $l(\alpha^0, AT^0) \rightarrow [45] \{l(\alpha^1, AT^2), l(\alpha^1, AT^3)\}$

R92. $l(\alpha^0, AT^0) \rightarrow [46, 49] \{l(\alpha^1, \alpha^2)\}$

R93. $l(\alpha^0, AT^0) \rightarrow [47, 48] \{l(\alpha^1, AT^2)\}$

R94. $l(BT^0, \beta^1) \rightarrow [51] \{l(BT^1, \beta^2), l(BT^3, \beta^1)\}$

R95. $l(BT^0, \beta^1) \rightarrow [55] \{l(BT^2, \beta^1)\}$

R96. $l(BT^0, \beta^1) \rightarrow [53, 54, 55] \{l(\beta^2, \beta^1)\}$

R97. $l(\beta^1, BT^0) \rightarrow [51] \{l(\beta^1, BT^2), l(\beta^1, BT^3)\}$

R98. $l(\beta^1, BT^0) \rightarrow [52, 55] \{l(\beta^1, \beta^2)\}$

R99. $l(\beta^1, BT^0) \rightarrow [53, 54] \{l(\beta^1, BT^2)\}$

R100. $l(CT^0, \gamma^1) \rightarrow [57] \{l(CT^2, \gamma^1), l(CT^3, \gamma^1)\}$

R101. $l(CT^0, \gamma^1) \rightarrow [58] \{l(CT^2, \gamma^1)\}$

R102. $l(CT^0, \gamma^1) \rightarrow [59, 60, 61] \{l(\gamma^2, \gamma^1)\}$

R103. $l(\gamma^1, CT^0) \rightarrow [57] \{l(\gamma^1, CT^2), l(\gamma^1, CT^3)\}$

R104. $l(\gamma^1, CT^0) \rightarrow [58, 61] \{l(\gamma^1, \gamma^2)\}$

R105. $l(\gamma^1, CT^0) \rightarrow [59, 60] \{l(\gamma^1, CT^2)\}$

R106. $ml(DT^0, \delta^1) \rightarrow [63] \{l(DT^2, \delta^1), l(DT^3, \delta^1)\}$

R107. $l(DT^0, \delta^1) \rightarrow [64]\ \{l(DT^2, \delta^1)\}$

R108. $l(DT^0, \delta^1) \rightarrow [65, 66, 67]\ \{l(\delta^2, \delta^1)\}$

R109. $l(\delta^1, DT^0) \rightarrow [63]\ \{l(\delta^1, DT^2), l(\delta^1, DT^3)\}$

R110. $l(\delta^1, DT^0) \rightarrow [64, 67]\ \{l(\delta^1, \delta^2)\}$

R111. $l(\delta^1, DT^0) \rightarrow [65, 66]\ \{l(\delta^1, DT^2)\}$

Apêndice C – Código HQML especificado no Estudo de Caso

```
<HQML>
 <User Preferences>
   <Application Type>
   Video On Demand
   </Application Type>
   <Desired QoS>
   High
   </Desired QoS>
 </User Preferences>
 <Application Name>
 Case Study
 </Application Name>
  <ServerGroup>
    <Server>
      <Name>
      Hardware Server
      </Name>
      <Category>
      Server
      </Category>
      <NodeLabel>
      0
      </NodeLabel>
      <HardwareEnvironment>
      PC PentiumIII
      </HardwareEnvironment>
      <SoftwareEnvironment>
      Windows 2000
      </SoftwareEnvironment>
      <Memory>
      512000 KB
```

```
      </Memory>
      <Disk>
      80000 MB
      </Disk>
      <Bandwidth>
      100 Mbits
      </Bandwidth>
       <AtomicComponent>
         <Name>
         Software Server
         </Name>
         <Category>
         Server
         </Category>
         <NodeLabel>
         0
         </NodeLabel>
         <Memory>
         256000KB
         </Memory>
         <Disk>
         40000MB
         </Disk>
         <Bandwidth>
         100Mbits
         </Bandwidth>
       </AtomicComponent>
    </Server>
  </ServerGroup>
  <ClientGroup>
    <Client>
      <Name>
      Hardware Client
      </Name>
      <Category>
      Client
      </Category>
      <NodeLabel>
      1
      </NodeLabel>
      <HardwareEnvironment>
      PC PentiumIII
      </HardwareEnvironment>
      <SoftwareEnvironment>
      Windows 2000
      </SoftwareEnvironment>
      <Memory>
      512000KB
```

```
</Memory>
<Disk>
50000MB
</Disk>
<Bandwidth>
100Mbits
</Bandwidth>
 <AtomicComponent>
  <Name>
  Software Client
  </Name>
  <Category>
  Client
  </Category>
  <NodeLabel>
  1
  </NodeLabel>
  <Memory>
  256000KB
  </Memory>
  <Disk>
  20000MB
  </Disk>
  <Bandwidth>
  100Mbits
  </Bandwidth>
  <QoSParameters>
    <Delay>
      <Min>
      0
      </Min>
      <Max>
      1
      </Max>
    </Delay>
    <AdaptationRules>
      <Rule>
        <Name>
        Rule1
        </Name>
        <Order>
        101
        </Order>
      </Rule>
      <Rule>
        <Name>
        Rule2
        </Name>
```

```
    <Order>
    102
    </Order>
  </Rule>
</AdaptationRules>
<Jitter>
  <Min>
  0
  </Min>
  <Max>
  1
  </Max>
</Jitter>
<AdaptationRules>
  <Rule>
    <Name>
    Rule1
    </Name>
    <Order>
    201
    </Order>
  </Rule>
  <Rule>
    <Name>
    Rule3
    </Name>
    <Order>
    202
    </Order>
  </Rule>
</AdaptationRules>
<Loss>
  <Min>
  0
  </Min>
  <Max>
  1
  </Max>
</Loss>
<AdaptationRules>
  <Rule>
    <Name>
    Rule4
    </Name>
    <Order>
    301
    </Order>
  </Rule>
```

```
            <Rule>
              <Name>
              Rule6
              </Name>
              <Order>
              302
              </Order>
            </Rule>
          </AdaptationRules>
        </QoSParameters>
      </AtomicComponent>
    </Client>
  </ClientGroup>
</HQML>
```

Apêndice D – Tabela 6