ANTÔNIO TADEU AZEVEDO GOMES

UM FRAMEWORK PARA PROVISÃO DE QOS EM AMBIENTES GENÉRICOS DE PROCESSAMENTO E COMUNICAÇÃO

DISSERTAÇÃO DE MESTRADO

Departamento de Informática

Rio de Janeiro, 31 de Maio de 1999.

Pontifícia Universidade Católica do Rio de Janeiro

ANTÔNIO TADEU AZEVEDO GOMES

UM FRAMEWORK PARA PROVISÃO DE QoS EM AMBIENTES GENÉRICOS DE PROCESSAMENTO E COMUNICAÇÃO

Dissertação de Mestrado apresentada ao Departamento de Informática da PUC-Rio, como parte dos requisitos para obtenção do título de Mestre em Informática: Ciência da Computação.
Orientador: Luiz Fernando Gomes Soares.

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 31 de Maio de 1999.

Este trabalho é dedicado

A Luciana Figueirinha, pelo amor, carinho e compreensão sempre presentes em nosso convívio.

A meus pais Sebastião Davel Gomes e Maria Antonieta Azevedo Gomes, por todo carinho, atenção e ensinamento que sempre me deram.

E a Deus, pela Luz que Ele representa ao guiar todos os passos da minha caminhada.

Agradecimentos

Primeiramente, agradeço ao Professor Luiz Fernando Gomes Soares. Como orientador, contribuiu intensivamente, com sua dedicação, experiência e, sobretudo, perseverança, na elaboração deste trabalho. Como amigo, nunca permitiu, com seu bom humor, que o meu moral ficasse em baixa, mesmo nos momentos de maior dificuldade.

Aos colegas do Laboratório TeleMídia, agradeço pelos muitos momentos agradáveis que compartilhamos. Em especial, gostaria de agradecer aos amigos Marcus Antonio Rodrigues e Sérgio Colcher, pelo empenho e trabalho fundamentais para a realização desta dissertação, e a Rogério Rodrigues e Débora Muchaluat, sempre dispostos a colaborar.

Agradeço também a todos os professores e funcionários do Departamento de Informática da PUC-Rio, que colaboraram, de uma forma ou de outra, para a conclusão deste trabalho.

Aos meus tios Izael e Enyse Azevedo, e aos amigos Marcus Antonio Rodrigues, Bruno Muniz, Rômulo Martins e Gabriel Paillard, agradeço pela acolhida, sempre que precisei de uma “segunda casa”.

Ao CNPq e a Embratel, agradeço pela viabilização financeira deste trabalho.

Resumo

Nos últimos anos, tem-se observado uma crescente demanda por sistemas de processamento e comunicação multimídia. Nestes sistemas, a noção de *qualidade de serviço* (QoS) tem evoluído rapidamente, tanto como forma de especificação dos requisitos dos usuários, quanto como referência para a configuração, operação e manutenção dos mecanismos de compartilhamento e orquestração de recursos. Este trabalho de dissertação é motivado pela crença de que arquiteturas para fornecimento de serviços específicos não são flexíveis o bastante para dar suporte adequado a novas técnicas de tratamento de mídias. Como uma alternativa, o trabalho apresenta um framework que organiza a provisão de QoS em um domínio genérico o suficiente para incluir quaisquer ambientes de processamento e comunicação, desde subsistemas específicos até sistemas multimídia distribuídos. Com estas características, o framework permite a construção de sistemas configuráveis, no que se refere à introdução de novos serviços, e facilita a implementação dos mecanismos de compartilhamento e orquestração de recursos presentes nesses sistemas.

Palavras-chave: Qualidade de Serviço, Sistemas Multimídia, Orientação a Objetos.

Abstract

Over the past few years, an emerging demand for multimedia communication and processing systems has been observed. On these systems, the notion of *quality of service* (QoS) has evolved rapidly, either as a way for user requirements specification, or as a reference for the configuration, operation and maintenance of the resource sharing and orchestrating mechanisms. This thesis is motivated by the belief that architectures for the provision of specific services are not flexible enough to support new media treatment technics. As an alternative, the thesis presents a framework that organizes QoS provision in a domain which is sufficiently generic to include any communication and processing environment, encompassing from specific subsystems to distributed multimedia systems. With these characteristics, the framework allows the construction of configurable systems, with regard to the introduction of new services, and eases the implementation of resource sharing and orchestrating mechanisms.

Palavras-chave: Quality of Service, Multimedia Systems, Object Orientation.

Índice Analítico

Índice de Figuras

Capítulo 1

Introdução

Nos últimos anos, tem-se observado uma crescente demanda pelo suporte ao uso de diferentes mídias de representação (áudio, vídeo, imagens, textos, etc.) nos diversos sistemas de processamento[1] e comunicação existentes. Tal demanda por *sistemas de processamento e comunicação multimídia* provêm do número cada vez maior de *aplicações multimídia* sendo implementadas e utilizadas, como a videofonia, videoconferência, transferência de documentos, correio eletrônico multimídia, vídeo sob demanda, entre outras.

A diversidade de requisitos de processamento e comunicação impostos pelas diferentes mídias a esses sistemas apresenta-se como um dos grandes problemas em voga nas áreas de sistemas operacionais, redes de computadores e computação distribuída. A título de exemplo, enquanto a natureza isócrona das mídias contínuas (áudio e vídeo) impõe requisitos relacionados ao retardo de processamento e comunicação de dados, a natureza de rajadas das mídias discretas (imagens gráficas não animadas, texto, etc.) impõe outros relacionados à tolerância a perdas de dados [SOARES95].

Seria relativamente fácil para um sistema oferecer a *qualidade* de processamento e comunicação requerida pelos dados de uma determinada mídia se somente recursos dedicados fossem usados. Isto é o que acontece por exemplo nos sistemas de comunicação telefônica atuais. Entretanto, devido a restrições gerenciais

[1] Neste trabalho, o termo *processamento* abrange as atividades de tratamento (codificação, compressão, etc.) e armazenamento de dados.

(espaço, mão-de-obra, etc.) e de custo, há a necessidade cada vez maior dos sistemas possuírem recursos eficientemente compartilhados pelo processamento e comunicação de dados de diferentes mídias. Ou seja, um mesmo sistema deve ser capaz, simultaneamente, de fornecer diferentes *serviços* de processamento e comunicação às aplicações multimídia (de acordo com as diferentes mídias de interesse), e de aumentar ao máximo a utilização efetiva de seus recursos, permitindo que ele (sistema) ofereça acesso ao maior número possível de aplicações [LU96].

É importante notar que os requisitos impostos pelas diferentes mídias, principalmente as contínuas, são *fim-a-fim*, ou seja, envolvem todo o sistema, desde os *produtores* (câmeras, microfones, servidores de arquivos, etc.) até os *consumidores* (visores, alto-falantes, clientes, etc.) dos dados. Especificamente, os *subsistemas* mais importantes que podem potencialmente afetar os requisitos de processamento e comunicação de uma mídia, e que portanto serão o alvo principal deste trabalho, são:

- As *redes de distribuição* e
- Os *sistemas operacionais*.

Redes de Distribuição

Graças ao desenvolvimento da tecnologia de redes de alta velocidade e de integração de serviços, em especial do *Modo de Transferência Assíncrono* (*Asynchronous Transfer Mode* – ATM) [ITUT.I.321] [ATM93], já existem algumas soluções para o fornecimento de múltiplos serviços às aplicações multimídia, porém, quando procura-se implementar vários serviços específicos sobre essas redes, a diversidade de requisitos de comunicação permanece um obstáculo à utilização das mesmas. Estratégias como a de organizar serviços com requisitos semelhantes de comunicação em classes genéricas de serviço procuram diminuir tais dificuldades, mas ainda deixam a desejar. Isto porque o desenvolvimento de um único protocolo de comunicação (ou de um conjunto finito de protocolos, divididos por classes de serviço) que atenda a todos os possíveis requisitos de comunicação (existentes ou que ainda estão por vir) torna o protocolo muito complexo, pouco flexível, e ainda assim, pouco genérico.

Sistemas Operacionais

Mesmo com o uso da tecnologia ATM nas redes de distribuição, os requisitos de processamento e comunicação das mídias, em especial as contínuas, podem ser alterados nas estações de origem e destino dos dados. Na maioria das estações finais, os dados são movidos entre produtores, consumidores e a pilha de protocolos de comunicação sob o controle de software. As camadas superiores da maioria das pilhas de protocolos também são implementadas em software. Devido à natureza de compartilhamento do tempo de processamento da maioria dos sistemas operacionais convencionais [SILBERSCHATZ95] [TANENBAUM92], o tempo de execução desses softwares varia dependendo da carga sobre o sistema operacional, podendo causar variações consideráveis no processamento e na comunicação dos dados das diferentes mídias.

1.1 O Conceito de Qualidade de Serviço

Um sistema multimídia como um todo deve agir semelhante a um pipeline. Quando algum segmento do pipeline não funciona corretamente, os requisitos fim-a-fim de processamento e comunicação de uma mídia não podem ser garantidos. Para possibilitar que múltiplos tipos de mídia tenham seus requisitos garantidos em cada um dos segmentos do pipeline, os recursos referentes a cada um desses segmentos devem poder ser eficientemente compartilhados. Adicionalmente, o sistema como um todo deve coordenar a operação desses segmentos, através da *orquestração* de seus recursos, para que os requisitos fim-a-fim possam ser efetivamente garantidos (como ilustra a Figura 1).

Tanto o compartilhamento eficiente de recursos quanto a orquestração dos mesmos só pode ser conseguida se as aplicações indicarem ao sistema a qualidade desejada do serviço, que implica, em última instância, na “quantidade” de recursos necessária para o seu fornecimento nos diferentes subsistemas.

Nesse contexto, o termo *qualidade de serviço* (*Quality of Service* – QoS [ITUT.I.350] [VOGEL95]) denomina o efeito coletivo provocado pelo conjunto das características quantitativas e qualitativas de um serviço oferecido por um sistema multimídia que é necessário para alcançar a funcionalidade requisitada pelas aplicações.

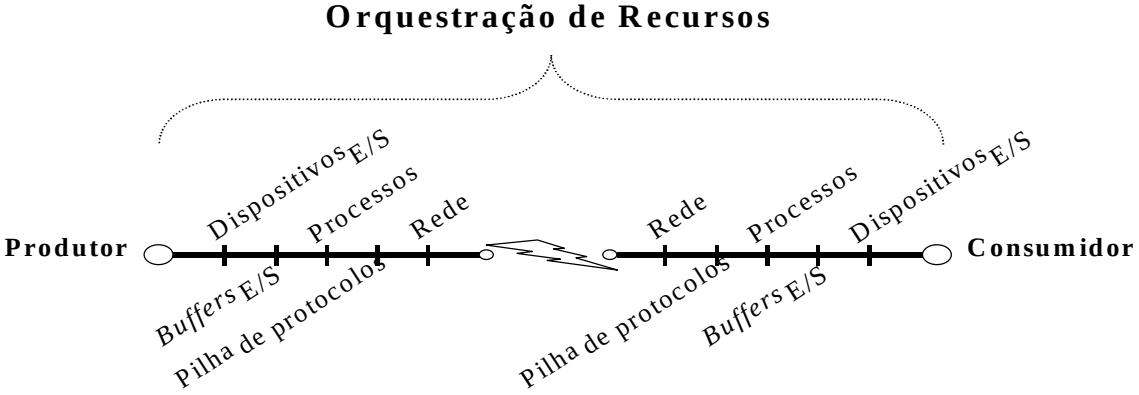


Figura 1: cenário de um sistema multimídia (baseado em [CAMPBELL96]).

1.2 Soluções Presentes

Até recentemente, grande parte do desenvolvimento relacionado ao suporte à provisão de QoS em sistemas multimídia ocorreu no contexto de subsistemas isolados [ZHANG93] [DOERINGER90] [COULSON95] [ITUT.I.121], sem levar em consideração a analogia *sistema multimídia* e *pipeline* exposta anteriormente. Em [HUTCHISON94], a provisão de QoS nesses sistemas é descrita resumidamente como:

- *Incompleta*: as interfaces da maioria dos subsistemas são geralmente não configuráveis no que diz respeito à QoS;
- *Não confiável*: a maioria dos subsistemas não apresenta mecanismos que possam garantir os requisitos de processamento e comunicação das diversas mídias;
- *Não integrada*: quando presentes, tais mecanismos de provisão de QoS não são integrados entre todos os subsistemas; e
- *Pouco genérica*: é necessário o desenvolvimento de frameworks que normalizem o conceito atual de QoS nos diferentes subsistemas.

Em reconhecimento às deficiências acima, vários grupos de trabalho têm proposto novas abordagens de *arquitetura de sistema* à provisão de QoS nos sistemas

multimídia. Em [CAMPBELL95], tais arquiteturas são denominadas coletivamente de *arquiteturas de QoS*.

O objetivo básico das arquiteturas de QoS é definir conjuntos de interfaces que formalizem o conceito de QoS nos diferentes subsistemas. Esta formalização é semelhante a que é feita pelo *Modelo de Referência para Interconexão de Sistemas Abertos* (*Reference Model of Open Systems Interconnection – RM-OSI* [ISO84]) com relação ao conceito de serviço. Estas interfaces são em geral apresentadas sob a forma de frameworks, linguagens de programação ou bibliotecas desenvolvidos sobre subsistemas com características de QoS predefinidas. Com esta abordagem, as arquiteturas de QoS permitem a um projetista de sistemas integrar os mecanismos de provisão de QoS presentes nos diferentes subsistemas, de certo modo estruturando a provisão de QoS fim-a-fim.

A maior limitação da abordagem acima está relacionada à granulosidade de aplicação dessas arquiteturas, uma vez que elas *uniformizam o tratamento da QoS somente na fronteira entre os subsistemas*. A arquitetura interna dos subsistemas não é definida, o que é indesejável por vários motivos. Um exemplo é que a orquestração dos recursos, tanto interna a um subsistema quanto entre subsistemas, não pode ser modelada de maneira suficientemente genérica, tendo em vista a variedade de implementações específicas de subsistemas já existentes. Assim, o modelo de orquestração de recursos a ser utilizado em um sistema é deixado a critério do projetista, o que dificulta a realização efetiva de uma provisão de QoS fim-a-fim.

Outro obstáculo que um projetista pode encontrar é a diversidade de requisitos existentes. As arquiteturas de QoS apoiam-se em subsistemas com características de QoS predefinidas, nem sempre suficientes para atender a todos os possíveis requisitos de processamento e comunicação. Quando acontece de um subsistema não atender a um determinado conjunto de requisitos, os projetistas de sistemas são geralmente levados a estender estes subsistemas, de modo que estes passem a oferecer as características de QoS desejadas. Porém, raramente tais subsistemas são flexíveis o bastante para serem facilmente estendidos. Além disso, a ausência de uma arquitetura interna impede que determinados mecanismos para provisão de QoS potencialmente presentes em vários sistemas distintos (ou mesmo em subsistemas de um mesmo sistema) possam ser facilmente reutilizados.

Para solucionar tais problemas, uma outra abordagem que tem sido considerada em recentes trabalhos é a do oferecimento de um *único* serviço, configurável de acordo com a necessidade dos usuários [LAZAR95] [COLCHER98]. Tais trabalhos, em grande parte baseados no *Modelo de Referência para Processamento Aberto Distribuído* (*Reference Model of Open Distributed Processing – RM-ODP* [ISO95a]) e na *Arquitetura CORBA* (*Common Object Request Broker Architecture* [OMG96]), têm como principal característica a definição de *ambientes genéricos de processamento e comunicação*, sobre os quais residem pequenos componentes reutilizáveis por intermédio da tecnologia de *orientação a objetos* (*Object Oriented – OO*). Através da interconexão desses componentes (objetos) é possível construir implementações particulares de subsistemas e até de sistemas multimídia completos.

É com base nessa abordagem que o *Framework para Provisão de QoS em Ambientes Genéricos de Processamento e Comunicação* será apresentado. A definição deste framework é apenas um dos aspectos abordados no projeto RAVel[2], cujo objetivo principal é dar suporte integrado à construção de arquiteturas de protocolos de comunicação e de aplicações multimídia. Este projeto envolve também, entre outras atividades, a definição de um *Framework para Provisão de Serviço de Multicast* [RODRIGUES99] e um *Modelo de Referência Unificado* [COLCHER98], de onde foi retirada boa parte da terminologia adotada neste trabalho.

1.3 Objetivos da Dissertação

O objetivo principal deste trabalho é apresentar um framework que, por ser aplicável também na modelagem das arquiteturas internas dos vários subsistemas presentes em um sistema multimídia qualquer, alcance uma granulosidade mais fina do que a das arquiteturas de QoS. O grau de granulosidade alcançado pelo Framework para Provisão de QoS possibilita a projetistas a definição de sistemas multimídia altamente flexíveis no que concerne à provisão de QoS, através da adaptação de alguns poucos mecanismos para o fornecimento de diferentes serviços. Além disso, o framework também oferece a projetistas um modelo genérico de orquestração de recursos entre

[2] Esse projeto é baseado no convênio de pesquisa existente entre a PUC-Rio e a Empresa Brasileira de Telecomunicações (EMBRATEL), cujos objetivos incluem o estudo de novos serviços para redes de banda larga.

subsistemas de um sistema multimídia qualquer, graças à sua estruturação recursiva com relação às diferentes escalas de distribuição e níveis de abstração dos componentes presentes nesses sistemas.

Como subproduto da especificação do Framework para Provisão de QoS, é apresentada também neste trabalho uma terminologia básica, em grande parte fundamentada em padrões definidos por órgãos internacionais ligados às áreas de redes de computadores e sistemas distribuídos, que formaliza o conceito de QoS nas várias escalas de distribuição e níveis de abstração em que o framework pode ser aplicado. Com base nessa terminologia, que inclui um modelo genérico de operação para sistemas multimídia, outros trabalhos relacionados à questão da provisão de QoS nesses sistemas podem ser desenvolvidos.

1.4 Estrutura da Dissertação

A presente dissertação é organizada como se segue. No Capítulo 2, é introduzido um conjunto de termos, princípios e conceitos que formam a base da filosofia sobre a qual o Framework para Provisão de QoS foi desenvolvido. É apresentado também nesse capítulo um modelo genérico de operação de sistemas multimídia, de onde foram retirados os módulos principais constituintes do Framework para Provisão de QoS. Este módulos incluem estruturas de parametrização que capturam os requisitos de processamento e comunicação impostos pelas aplicações, e mecanismos de provisão de QoS que cuidam de oferecer a QoS desejada por elas.

A seguir, no Capítulo 3, é feito um apanhado das principais pesquisas relacionadas à questão da provisão de QoS em sistemas multimídia. Primeiramente, são reportados alguns trabalhos centrados em subsistemas específicos, mais precisamente em sistemas operacionais, redes de distribuição e protocolos de comunicação. Apresenta-se também algumas arquiteturas de sistema e modelos de computação distribuída que consideram QoS fim-a-fim como premissa básica para a implementação de sistemas multimídia. Além disso, há uma seção dedicada exclusivamente à questão da aplicação de patterns e frameworks no desenvolvimento de sistemas multimídia distribuídos.

No Capítulo 4, o Framework para Provisão de QoS é descrito por intermédio de um conjunto de elementos que representam de maneira abstrata as estruturas de parametrização e os mecanismos de provisão de QoS componentes do modelo apresentado no Capítulo 2. Esse capítulo apresenta também, para cada elemento constituinte do Framework para Provisão de QoS, um exemplo de aplicação em um subsistema ou sistema multimídia distribuído específico. Em cada um desses exemplos, são ressaltadas as vantagens, desvantagens, e implicações em se utilizar os elementos do Framework para Provisão de QoS.

No Capítulo 5, são apresentados os detalhes de implementação de um sistema de distribuição de vídeo, arquitetado com base no framework especificado no Capítulo 4. O ambiente de implementação utilizado consta de um conjunto de estações finais controladas pelo sistema operacional Solaris 2.5 e interligadas por uma rede ATM. Nas estações finais, é definida uma arquitetura de escalonamento de processadores com garantias estatísticas de QoS sobre o Solaris. Já sobre a rede ATM, é implementada a simulação de uma rede ATM aberta e configurável com relação aos tipos de serviços passíveis de serem oferecidos por ela. Finalmente, o Capítulo 6 apresenta um conjunto de conclusões gerais a respeito da dissertação, e pontos onde trabalhos futuros são necessários.

Capítulo 2

Terminologia, Princípios e Conceitos

Neste capítulo, será apresentado um conjunto de termos genéricos, originados a partir da abordagem de serviços configuráveis, como descrita em [COLCHER98], relacionados à provisão de QoS em sistemas de processamento e comunicação quaisquer. Por questões de padronização, optou-se também por adotar vários dos conceitos descritos na proposta DIS 9309 [ISO95], candidata a se tornar um padrão internacional pela *International Organization for Standardization* (*ISO*) e pelo *International Telecommunications Union* (*ITU-T*). Grupos de pesquisa em ambos os órgãos têm trabalhado em conjunto na definição de padrões que garantam uma interpretação coerente do termo QoS nas várias áreas de computação.

Ao apresentar uma terminologia genérica, e, em grande parte, padronizada por órgãos como a ISO e o ITU-T, este capítulo visa formalizar a independência do Framework para Provisão de QoS com relação às escalas de distribuição e aos níveis de abstração em que ele pode ser aplicado.

2.1 Distribuição X Abstração

Até o momento, os termos *distribuição* e *abstração* têm sido usados conjuntamente, porém indiscriminadamente, neste trabalho. Entretanto, a diferença entre os dois conceitos, embora por vezes tênue, é importante.

A escala de distribuição refere-se à localização relativa de componentes de um sistema que se comunicam entre si. Como exemplos de escala de distribuição, componentes podem estar em um mesmo processo, em processos diferentes de uma mesma estação, ou em estações diferentes de uma rede de distribuição[3].

Já o nível de abstração refere-se à visão que se tem dos componentes de um sistema. Seres humanos, aplicações, e dispositivos multimídia quaisquer[4] são exemplos de produtores e consumidores de dados em diferentes níveis de abstração.

2.2 Ambientes de Processamento e Comunicação

Define-se um *ambiente de processamento e comunicação* [COLCHER98] como sendo formado por um conjunto de componentes e por uma máquina de objetos, responsável pelo processamento desses componentes, assim como pela comunicação entre eles.

O uso do termo *ambiente*[5], por si só, já sugere diversas escalas de distribuição. Por exemplo, ele pode ser tão simples quanto um ambiente de execução de objetos presente em uma linguagem de programação OO, ou tão complexo quanto um ambiente de objetos distribuídos, formado por um conjunto de ambientes locais de execução de objetos e pela infra-estrutura de comunicação entre esses ambientes.

---

[3] Embora, no presente trabalho, o termo *rede de distribuição* designe qualquer infra-estrutura de transmissão, dá-se ênfase às redes baseadas em nós intermediários, sejam elas redes físicas (como as redes com tecnologia ATM) ou inter-redes (como a Internet).

[4] No presente trabalho, o termo *dispositivo multimídia* abrange dispositivos de entrada e saída como câmeras, microfones, codecs, visores, alto-falantes, e quaisquer outros dispositivos produtores ou consumidores de dados de mídias contínuas.

[5] Por vezes, o uso do termo *ambiente* pode gerar confusão, devido as inúmeras situações em que ele pode ser empregado. Por isso, no presente trabalho, sempre que for considerado necessário o uso deste termo no contexto de um ambiente específico, o mesmo será acompanhado de um qualificador, como por exemplo, *ambiente de execução, ambiente distribuído,* etc.

2.3 Fornecedores de Serviços

Sob o ponto de vista dos *usuários* do ambiente, este possui um *fornecedor de serviços*, cuja fronteira de atuação é definida por um subconjunto dos componentes do ambiente responsável pelo fornecimento de um ou mais serviços a estes usuários.

Assim como quaisquer outros elementos presentes em um ambiente, os conceitos de *fornecedor de serviços* e *usuário* dependem da escala de distribuição e do nível de abstração em questão. Para citar exemplos, enquanto um fornecedor de serviços pode ser um sistema operacional de uma estação, um sistema de comunicação de uma rede ou um sistema multimídia distribuído completo (incluindo as aplicações ou parte delas), um usuário pode ser um simples objeto, uma aplicação multimídia ou uma pessoa interagindo com o sistema.

Além de definirmos a fronteira de atuação do fornecedor de serviços, também é importante que definamos as *interfaces* entre ele e os usuários do ambiente, ou seja, os dados provenientes dos usuários que são passados ao fornecedor e os dados que este produz como saídas para serem passadas aos usuários. Fornecedores produzem *eventos de saída* quase sempre como resposta a estímulos (ou *eventos de entrada*) do ambiente; porém, a este trabalho não interessarão todos os eventos gerados no ambiente, mas sim aqueles gerados pelos usuários ou que exigem resposta do fornecedor.

Pode-se representar um *serviço* provido por um fornecedor através de relações entre eventos de entrada gerados pelos usuários e eventos de saída produzidos pelo fornecedor de serviços. Na maioria das vezes, a definição de tais relações depende também do estado interno atual dos componentes do fornecedor (que em conjunto definem o próprio estado interno do fornecedor).

Além dos dados a serem trocados entre os usuários e o fornecedor de serviços, eventos de entrada e saída sempre possuem um instante de início e um local de ocorrência, que pode ser qualquer ponto na interface entre o usuário e o fornecedor, ou seja, qualquer componente do fornecedor que se comunica diretamente com o usuário. Um serviço é fornecido a um usuário por intermédio de eventos que ocorrem nesses pontos da interface, denominados de *pontos de interação* (PI). Exemplos de PIs são as chamadas às funções de um sistema operacional (*system calls* [SILBERSCHATZ95]) e os

pontos de acesso a serviços (*Service Access Points* – SAPs [SOARES95]) de uma camada de um sistema de comunicação.

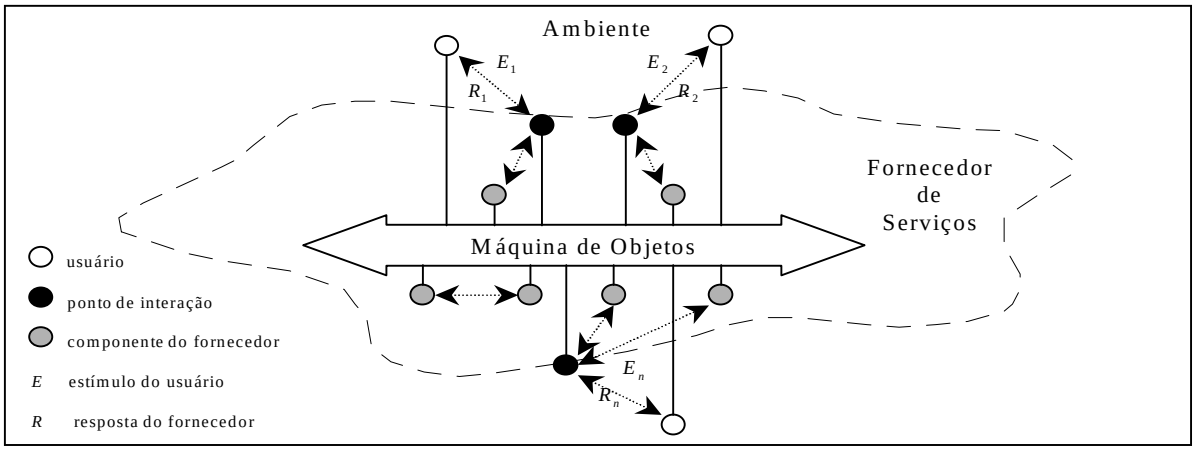


Figura 2: interface entre usuários do ambiente e o fornecedor de serviços.

2.4 Qualidade do Serviço de um Fornecedor

No contexto da terminologia apresentada, o termo *QoS* pode ser definido como o conjunto das características (qualitativas e quantitativas) de processamento e comunicação suportadas por um serviço, que permite a provisão da funcionalidade desejada por usuários do ambiente. Essa funcionalidade pode ser entendida como a capacidade de processamento e comunicação para os dados de uma determinada mídia, por exemplo.

Dessa forma, a QoS pode ser representada através de relações entre os requisitos dos usuários, os eventos que ocorrem na interface entre eles e o fornecedor e, possivelmente, o estado interno atual do fornecedor.

2.5 Princípios Para Provisão de QoS

Apoiando-se na definição de QoS dada acima, percebe-se a existência de dois princípios básicos relacionados à provisão de QoS em um fornecedor:

- A especificação dos requisitos dos usuários e

- A provisão de mecanismos que garantam os requisitos dos usuários através do compartilhamento e da orquestração de recursos.

Em [CAMPBELL95], são enumerados, de modo mais formal, cinco princípios que governam a implementação de fornecedores que provêem QoS a seus usuários:

- *Princípio da integração*: a QoS deve ser configurável, previsível e manutenível em toda a infra-estrutura de suporte aos serviços oferecidos pelo fornecedor, para que a QoS desejada pelo usuário possa ser atendida efetivamente.

- *Princípio da separação*: as atividades relativas ao processamento e comunicação de dados devem ser funcionalmente distintas das atividades de configuração, operação e manutenção da QoS, para que não interfiram no desempenho umas das outras.

- *Princípio da transparência*: os usuários não devem ter ciência da complexidade de configuração, operação e manutenção da QoS oferecida pelo fornecedor.

- *Princípio da gerência assíncrona*: os diversos componentes de configuração, operação e manutenção da QoS oferecida pelo fornecedor, que compõem os mecanismos de provisão de QoS [ISO95], funcionam conjuntamente e paralelamente, mas em diferentes escalas de tempo. O correto funcionamento do fornecedor depende de algoritmos assíncronos que gerenciem esses componentes.

- *Princípio do desempenho*: o fornecedor deve ser projetado de acordo com regras [CLARK90] [SALTZER84] que estruturem os seus componentes da forma mais eficiente possível, como por exemplo, evitando multiplexações de dados de diferentes usuários por um mesmo componente [TENNENHOUSE89], utilizando *hardware* para determinados tipos de processamento e comunicação [ZITTERBART92], etc.

2.6 Fases de Provisão de QoS

Com base em alguns dos princípios enumerados acima, pode-se definir um modelo genérico de operação para fornecedores (baseado no modelo descrito em [ISO95]) dividido nas seguintes fases:

- *Iniciação do fornecedor de serviços;*
- *Requisição de serviços;*
- *Estabelecimento de contratos de serviço e*
- *Manutenção de contratos de serviço.*

2.6.1 Iniciação do Fornecedor de Serviços

Para tornar um fornecedor de serviços operacional, é necessário primeiramente que seja definida a infra-estrutura que dará suporte aos serviços oferecidos e que determinará, juntamente com as *políticas de provisão de QoS* [ISO95] adotadas durante a existência do fornecedor, a forma de descrição do *estado interno* do mesmo. Políticas de provisão de QoS são conjuntos de estratégias, utilizadas pelos mecanismos de provisão de QoS, que regulam a maneira como um fornecedor de serviços controla e gerencia a QoS por ele provida. As políticas de provisão de QoS usadas por um fornecedor estão intimamente relacionadas à sua implementação real.

A definição do estado interno de um fornecedor de serviços inclui informações sobre *recursos* disponíveis no ambiente. Recursos podem ser classificados como *estaticamente* ou *dinamicamente* compartilhados. Enquanto os primeiros permanecem dedicados a um único fornecimento de serviço por vez, os últimos podem ser compartilhados por vários desses fornecimentos simultaneamente. Como exemplos de recursos dinamicamente compartilhados, podem ser citados o processador e a memória de uma estação e os enlaces (físicos ou lógicos) de uma rede. Dispositivos multimídia (câmeras, microfones, etc.) são exemplos de recursos estaticamente compartilhados.

A dinâmica de alteração do estado interno de um fornecedor de serviços é determinada pelo *ciclo de vida* da QoS oferecida aos usuários do ambiente. Em alguns fornecedores, a QoS é especificada somente na etapa de projeto e garantida estaticamente através de decisões de dimensionamento, podendo ser reconfigurada somente através da reiniciação ou reconstrução do fornecedor. Em outros [SILBERSCHATZ95] [COMER95], a QoS é determinada com base na coleta de informações sobre a QoS alcançada em serviços prestados anteriormente. O segundo modelo possui um maior dinamismo do que o primeiro, porém em ambos o usuário não tem a possibilidade de especificar adequadamente a QoS desejada.

Nos últimos anos, tem-se adotado modelos de ciclo de vida de QoS mais dinâmicos dos que os apresentados acima. Em todos eles, o serviço é parametrizável no que concerne à QoS desejada pelo usuário, porém nos modelos mais simples [ATM93] ela só pode ser modificada[6] com a interrupção do serviço, o que não acontece necessariamente em outros modelos de maior complexidade [CAMPBELL96]. Este trabalho leva em consideração esse último modelo na descrição dos mecanismos de provisão de QoS presentes nas outras fases.

2.6.2 Requisição de Serviços

Após a iniciação do fornecedor, usuários podem requisitar[7] a qualquer momento, através de PIs apropriados, serviços ao fornecedor. No que se refere às características de processamento e comunicação desejadas pelo usuário, uma requisição se dá através da:

- Caracterização da carga de entrada a ser gerada, caso o usuário seja um produtor (fonte) de dados, ou de saída, caso ele seja um consumidor (destino), e da

- Especificação da QoS que ele deseja do fornecedor.

[6] A QoS pode ser modificada dinamicamente tanto pelo usuário (caso mais comum) quanto pelo próprio fornecedor, como acontece por exemplo em ambientes tolerantes a falhas.

[7] É importante salientarmos que a própria requisição de serviço também é um evento, assim como qualquer outra troca de dados entre o usuário e o fornecedor. Entretanto, tal evento está associado a atividades de configuração do serviço, e não àquelas relacionadas diretamente ao processamento e comunicação de dados do usuário.

O termo *carga* refere-se à dinâmica do fluxo de dados a ser gerado pelos eventos ocorridos na interface entre os usuários e o fornecedor, e que estão associados ao fornecimento do serviço requisitado. *Fluxo* é outro conceito dependente da escala de distribuição e do nível de abstração em questão. São exemplos de fluxos um conjunto de instruções a serem executadas em um processador, um conjunto de pacotes a serem transmitidos através de um sistema de comunicação, ou um conjunto de quadros de vídeo a serem enviados por um sistema de distribuição de vídeo.

No caso da comunicação de dados entre usuários, o fluxo é representado conjuntamente pela produção e consumo de dados de uma determinada mídia por um ou mais usuários[8]. Nesse caso, fluxos podem ser *ponto a ponto* (*unicast*) ou *multiponto* (*multicast*), dependendo do número de usuários que os consumam.

Tanto a carga quanto a QoS podem ser caracterizadas através do uso de descritores de comportamento do fluxo, que levam em consideração os eventos de entrada e saída, os dados neles contidos e os seus locais de ocorrência. Podemos citar como exemplos de descritores a periodicidade de geração dos eventos, a quantidade média de dados neles contida, etc.

A interface entre o fornecedor e seus usuários deve permitir, a estes, requisições de serviço condizentes com suas *visões de QoS*. Isto significa que a caracterização da carga e a especificação da QoS devem ser facilmente "compreendidas" pelos usuários, ou seja, devem estar de acordo com o nível de abstração no qual os usuários estão inseridos.

Como exemplos de visões de QoS, pessoas preocupam-se com a qualidade de percepção (e o custo associado) em um sistema de distribuição de vídeo, enquanto aplicações multimídia levam mais em consideração a qualidade de transmissão em um sistema de comunicação.

[8] No presente trabalho, assume-se que fluxos de comunicação são sempre unidirecionais. Fluxos bidirecionais são definidos como pares de fluxos em sentidos opostos.

Uma vez que um ambiente pode envolver vários níveis de abstração, cada um desses níveis pode estar associado a uma visão de QoS distinta. Isto implica em traduzir ou mapear tais visões entre os diferentes níveis. Esta função é oferecida pelos mecanismos de *mapeamento de QoS*, nos quais vários mecanismos de provisão de QoS (em especial, os que gerenciam a orquestração de recursos) se apoiam para funcionarem corretamente.

A título de exemplo, a Figura 3 ilustra os níveis de visão de QoS sugeridos em [WADDINGTON97] para sistemas multimídia distribuídos.

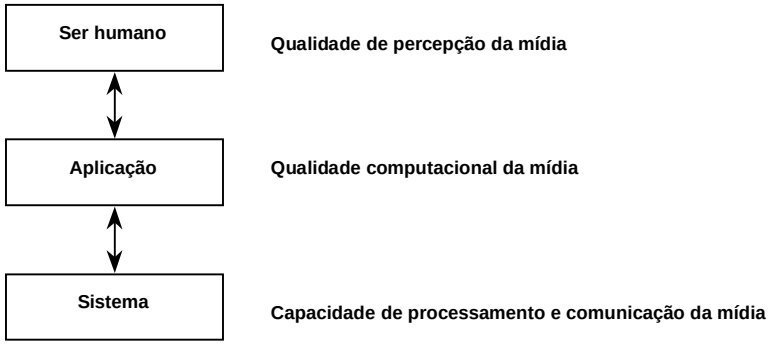


Figura 3: visões de QoS segundo [WADDINGTON97].

Segundo este modelo, o nível de visão de QoS mais alto é o do *ser humano*. Neste nível uma pessoa pode, por exemplo, escolher a qualidade do filme que ela deseja assistir, como TV normal com som analógico ou de alta definição (*High Definition TV – HDTV*) com som digital estéreo.

Já no nível das *aplicações*, uma distribuidora de vídeo pode, por exemplo, escolher a qualidade computacional do áudio e do vídeo, como o número de canais e de bits por amostragem do áudio, e o tamanho de quadro e a taxa de atualização (*refresh*) do vídeo, além do possível relacionamento temporal entre essas duas mídias.

Finalmente, no nível de *sistema* a visão de QoS relaciona-se mais diretamente à capacidade de processamento ou comunicação presente nos recursos do ambiente, como a taxa máxima de transmissão de bits por um enlace ou de execução de instruções por um processador. O nível de sistema pode ser composto por várias visões de QoS diferentes, correspondentes aos diversos recursos presentes no ambiente.

2.6.3 Estabelecimento de Contratos de Serviço

Ao receber uma nova requisição de serviço, o fornecedor determina se o ambiente tem recursos suficientes para manter a QoS desejada pelo usuário. O fornecedor deve tentar otimizar a utilização de recursos do ambiente, para possibilitar a admissão do maior número possível de fluxos de usuários, sem que isto traga algum impacto na QoS de fluxos previamente admitidos. Os mecanismos de *controle de admissão de fluxos* são os responsáveis por esses testes, que levam em conta, além do estado interno atual do fornecedor, a caracterização da carga e a especificação da QoS informadas pelos usuários durante a requisição de serviços.

Se o ambiente tiver recursos suficientes, o fornecedor admitirá o fluxo do usuário e fará a *alocação dos recursos* necessários para servi-lo (levando possivelmente a uma mudança de estado interno), de tal forma que a QoS especificada seja satisfeita. Senão, ele poderá ou rejeitar o fluxo do usuário, ou lhe sugerir uma outra especificação de QoS factível de ser satisfeita.

Nesse último caso, se o usuário aceitar a nova proposta, seu fluxo é admitido, senão é rejeitado, podendo o usuário tentar requisitar o serviço novamente em outra oportunidade. Estes ciclos de requisição e resposta entre os usuários e o fornecedor são providos pelos mecanismos de *negociação da QoS*.

Durante a prestação do serviço, mudanças na QoS podem ser necessárias por várias razões. Em um exemplo típico, uma pessoa inicialmente requer um serviço de vídeo sob demanda de alta definição. Como ela é cobrada com base na QoS oferecida, ela pode posteriormente decidir reduzir a qualidade de definição do vídeo. Logo, mecanismos de *renegociação da QoS* também são necessários.

Mecanismos de negociação e renegociação da QoS são responsáveis também pela parte da orquestração de recursos que ocorre durante a fase de estabelecimento, no caso de fornecedores que:

- Gerenciam mais de um tipo de recurso em um mesmo subsistema, sendo necessária então a “divisão” da responsabilidade pela provisão da QoS fim-a-fim entre esses recursos, ou que

- Envolvem vários subsistemas, e a divisão dessa responsabilidade ocorre entre os diferentes subsistemas, ou então que

- Provêem serviços de comunicação (ponto a ponto ou multiponto) e as requisições de serviço dos usuários participantes são diferentes.

Exemplos típicos de fornecedores que gerenciam mais de um tipo de recurso são os sistemas operacionais. Seria de pouca utilidade definir algoritmos apropriados de escalonamento de processadores para aplicações multimídia e de tempo real, como os apresentados em [LIU73], se o processamento referente a estas aplicações estivesse continuamente sujeito a atrasos imprevisíveis graças a paginações ou *swappings* arbitrários de memória [CAMPBELL96]. Daí a necessidade de se orquestrar ambos os recursos, processador e memória, para que a QoS requisitada pelas aplicações possa ser verdadeiramente provida.

2.6.4 Manutenção de Contratos de Serviço

Após o fluxo do usuário ser admitido pelo fornecedor, este último deve garantir que o usuário atenha-se à carga caracterizada, caso ele seja uma fonte, e que a QoS negociada seja mantida durante todo o tempo de uso do serviço.

Em um *contrato de serviço*: como em todo contrato, ambas as partes interessadas devem colaborar, ou seja, o usuário deve ater-se à carga caracterizada e o fornecedor deve manter a QoS negociada. A quebra do contrato por uma das partes pode ocasionar desde uma mera notificação ao usuário até a interrupção do serviço para aquele usuário.

Dentre os mecanismos presentes nessa fase, estão incluídos os de *escalonamento de recursos, monitorização de fluxos* e *sintonização da QoS*.

Os mecanismos de escalonamento permitem o compartilhamento dinâmico de recursos entre diversos fluxos de dados presentes em um ambiente, de maneira tal que a QoS negociada para cada fluxo possa ser mantida, e que a utilização dos recursos seja a mais otimizada possível. Cada fluxo “enxerga” a sua parcela de utilização de um determinado recurso (necessária para a provisão da QoS desejada) como um *recurso virtual* ao qual o fluxo tem acesso exclusivo. Os paginadores de memória e escalonadores de processadores dos sistemas operacionais [SILBERSCHATZ95], assim como os comutadores de circuitos virtuais presentes em alguns sistemas de comunicação [ITUT.I.321], são exemplos de mecanismos de

escalonamento de recursos. Os mecanismos de alocação de recursos podem também operar com base no conceito de recurso virtual, se o recurso em questão puder ser dinamicamente compartilhado.

Já os mecanismos de monitorização de fluxos permitem o registro da carga efetivamente gerada pelos usuários e da QoS realmente oferecida pelo fornecedor. Em caso de violação de um contrato de serviço, estes mecanismos podem simplesmente ter seus registros repassados aos usuários de interesse sob a forma de *alertas* [ISO95], ou, então, tanto os mecanismos de renegociação[9] quanto os de sintonização da QoS podem ser acionados, dependendo do grau de depreciação da QoS anteriormente negociada.

Os mecanismos de sintonização da QoS são responsáveis pela manutenção da orquestração de recursos definida durante a negociação da QoS, sem que haja a necessidade de interrupção (isto é, renegociação) do fornecimento do serviço. Quando um fornecedor gerencia vários recursos, se um deles não puder suportar sua parcela de responsabilidade pela provisão da QoS fim-a-fim (graças a uma sobrecarga, por exemplo), caberá ao mecanismo de sintonização efetuar operações de ajuste nos outros recursos (através dos mecanismos de escalonamento), ou, em casos mais drásticos, escolher recursos alternativos, a fim de manter a QoS previamente negociada.

Um bom exemplo da atuação dos mecanismos de sintonização está na correção de variações no retardo de trânsito de dados em um sistema de comunicação, que são especialmente críticas em serviços conversacionais[10]. Caso o mecanismo de monitorização detecte, em um determinado fluxo, a presença de variações no retardo de trânsito que estejam fora das especificações do contrato estabelecido, o mecanismo de sintonização da QoS pode ser acionado para, por exemplo, ajustar os mecanismos de escalonamento de pacotes tanto nas estações finais quanto nos nós intermediários da rede de distribuição, para melhor delimitar as variações no retardo de trânsito dos dados desse fluxo.

[9] Neste caso, a renegociação da QoS não é iniciada pelo usuário, mas pelo fornecedor.
[10] Segundo a classificação do ITU-T para aplicações multimídia [ITUT.I.211], *serviços conversacionais* provêem meios para a comunicação bidirecional em tempo real entre usuários. Entre as aplicações que se enquadram nesses serviços, podemos citar videofonia e videoconferência.

2.7 Grandezas e Dimensões de QoS

Qualquer que seja o tipo de ambiente considerado (multimídia, tolerante a falhas, etc.), há nele um conjunto de *grandezas* suscetíveis de medida, como quantidade de dados e tempo, por exemplo, que podem ser identificadas. A quantificação de uma destas grandezas envolve a comparação com um valor unitário da grandeza. Para quantificarmos o tempo decorrido entre dois eventos, por exemplo, precisamos de uma unidade padrão, como o segundo, o minuto, etc.

Muitas unidades de outras grandezas podem ser derivadas em termos de um pequeno número de unidades fundamentais. A grandeza *capacidade,* por exemplo, pode ser quantificada pela simples divisão da quantidade total de dados enviados por um PI (em bits ou bytes) pelo tempo decorrido entre o início e fim de envio (em segundos ou minutos). Uma unidade de quantificação da capacidade neste exemplo poderia ser bits/segundo ou bytes/minuto. A escolha de quais unidades são fundamentais é um tanto quanto arbitrária, e fora do escopo deste trabalho.

Pelo fato da capacidade poder ser quantificada como a divisão da quantidade de dados pelo tempo, pode-se dizer que ela tem as *dimensões* de quantidade de dados dividida por tempo. Para os fornecedores de serviços, algumas dimensões representam grandezas no ambiente que podem ser não só identificadas e quantificadas, mas também medidas para efeito de provisão da QoS. Elas são denominadas de *dimensões de QoS* [HUTCHISON94]. Dimensões de QoS são definidas independente da forma pela qual as grandezas correspondentes são simbolizadas ou controladas pelo fornecedor. Em um exemplo típico, é comum medir-se a QoS, em ambientes multimídia, utilizando-se dimensões como *quantidade de informação*, *tempo*, *coerência* e *integridade,* entre outras.

2.8 Parametrização de Serviços

Em todas as fases de provisão de QoS anteriormente apresentadas, a adaptação dos mecanismos de provisão de QoS a serviços e escalas de distribuição específicos é essencialmente guiada segundo a forma de descrição do comportamento

dos fluxos dos usuários e da infra-estrutura existente no ambiente. Todas essas informações podem ser estruturadas através do uso de *parâmetros de caracterização do serviço*. Um parâmetro é a imagem de uma propriedade (relacionada a uma grandeza qualquer) existente ou desejável em um fornecedor, dotada de um descritor que a nomeia e associada a um valor de um tipo qualquer (inteiro, ponto flutuante, lista de outros valores, etc.).

A partir das formas possíveis de utilização de parâmetros, podemos definir três tipos de parâmetros diferentes:

- *Parâmetros de desempenho do fornecedor*, que descrevem o próprio estado interno atual do fornecedor, ou seja, a habilidade do fornecedor em prover os serviços requisitados pelos usuários. *Vazão disponível (em um recurso)* e *retardo de trânsito instantâneo*[11] *(entre dois PIs)* são exemplos de parâmetros de desempenho do fornecedor.

- *Parâmetros de caracterização de carga*, que descrevem a dinâmica dos fluxos gerados pelos usuários. A *vazão máxima (em um PI)* é um exemplo de parâmetro de caracterização de carga.

- *Parâmetros de especificação da QoS*, que descrevem os requisitos de processamento e comunicação desejados pelos usuários. O *retardo de trânsito máximo* é um exemplo de parâmetro de especificação da QoS.

Quaisquer tipos de parâmetros de caracterização de serviços podem ser armazenados ou trocados entre componentes de um fornecedor. Enquanto parâmetros de caracterização de carga e de especificação da QoS são, em primeira instância, originados a partir de requisitos dos usuários, parâmetros de desempenho do fornecedor podem indicar a disponibilidade de recursos no ambiente, registrar medições relativas à real QoS sendo oferecida aos usuários (registros estes obtidos pelos mecanismos de monitorização de fluxos), ou ser resultados de análises de outros parâmetros.

É importante ser salientada a diferença de significado entre os termos *dimensão* e *parâmetro*. Enquanto dimensões descrevem grandezas a serem

[11] Neste trabalho, o termo *instantâneo* é usado para caracterizar parâmetros que são a própria medição da grandeza associada a eles em um dado instante de tempo. Para simplificar a nomenclatura usada, o termo instantâneo será omitido sempre que forem facilmente diferenciáveis os conceitos de grandeza e parâmetro.

quantificadas e medidas pelo fornecedor de serviços, parâmetros descrevem propriedades existentes ou desejáveis nesse fornecedor, que podem ser expressas em termos das dimensões.

O retardo de trânsito, por exemplo, é uma grandeza que corresponde ao tempo real que ocorre entre os instantes de passagem de um dado por dois PIs. A dimensão correspondente é o tempo. Os parâmetros retardo de trânsito e retardo de trânsito máximo relacionam-se a essa grandeza, descrevendo, respectivamente, propriedades existentes e desejáveis no fornecedor.

2.8.1 Qualificadores de Parâmetros

Parâmetros podem ser definidos através do uso de vários tipos de *qualificadores*, que podem expressar tanto a semântica desses parâmetros com relação à grandeza associada, quanto a semântica da própria grandeza. Em [ISO95], são apresentados os qualificadores de evento e de localização. Os *qualificadores de evento* permitem aos parâmetros descrever que eventos são considerados na definição da grandeza associada, e os *qualificadores de localização* quais os PIs onde os eventos de entrada e saída ocorrem. Como exemplo, podemos descrever a grandeza retardo de trânsito, de modo mais genérico, como o tempo decorrido entre dois eventos (no caso específico, os instantes de passagem de um dado) que ocorrem em dois PIs diferentes.

Parâmetros (em especial, os de desempenho do fornecedor) podem descrever grandezas também como funções de outras, através da definição de *qualificadores de dependência funcional* entre parâmetros. Por exemplo, a partir do parâmetro vazão pode-se definir vazão média, vazão máxima, variância da vazão, etc. As funções mais comuns são [ISO95]:

- Mínimo e máximo;
- Média;
- Variância e desvio padrão;
- Probabilidade $n, 0 \leq n \leq 1$.

Outros tipos de funções também podem ser definidos. Em particular, é possível definir parâmetros que descrevem grandezas como funções de mais de uma grandeza. Um exemplo é a *disponibilidade*, que é uma função da *manutenibilidade* e da *confiabilidade*[12]. Dependências funcionais são especialmente úteis para retratar mapeamentos entre diferentes níveis de visões de QoS e registros de medições relativas à real QoS sendo provida aos usuários.

Apesar de parâmetros poderem descrever grandezas por intermédio de eventos, PIs e dependências funcionais, a definição de um parâmetro, por si só, não deve apontar a forma de sinalização do evento, de endereçamento do PI ou de cálculo da função. Essas são tarefas dos diversos componentes de configuração e operação que constituem um fornecedor, incluindo aqueles que compõem os mecanismos de provisão de QoS. Todavia, parâmetros devem ser descritos adequadamente, ou seja, terem uma estrutura básica, para serem de alguma utilidade.

Várias são as notações existentes para a descrição de parâmetros de caracterização de serviços. Em [WADDINGTON97], são sumariadas algumas descrições de parâmetros de caracterização de carga e de especificação da QoS. Em todas elas, parâmetros estão associados a um ou mais valores, podendo incluir também:

- a semântica de cada um desses valores com relação à grandeza associada ao parâmetro, semântica essa normalmente associada a uma regra de estabelecimento ou manutenção de contratos de serviço definida pelas políticas de provisão de QoS adotadas, como por exemplo:
  - um alvo que deve ser aproximado o máximo possível;
  - um limite mínimo ou máximo que não deve ser ultrapassado;
  - um limiar inferior ou superior associado a ações específicas do fornecedor. O alvo deve pertencer ao intervalo formado pelos limiares, que por sua vez, deve estar contido no intervalo formado pelos limites;

[12] Em um ambiente simples que não possui tolerância a falhas ou redundância, a disponibilidade é definida como $MTBF/(MTBF + MTTR)$, onde $MTBF$ é o tempo médio entre falhas (*Mean Time Between Failures*) e $MTTR$ é o tempo médio de reparo (*Mean Time To Repair*).

- as ações a serem executadas quando limiares e limites são atingidos, dentre elas:
  - adaptar o modo de operação do fornecedor;
  - armazenar informações para futura referência;
  - notificar o usuário;
  - interromper o serviço para aquele usuário;

Adicionalmente, parâmetros podem conter informações relativas ao *nível de serviço* desejado pelo usuário. O nível de serviço expressa o grau de certeza de que a QoS será mantida pelo fornecedor, e sua adoção também é normalmente dependente das políticas de provisão de QoS definidas pelo fornecedor.

Como exemplos bastante comuns de níveis de serviço, temos [ISO95]:

- *Melhor esforço (best effort)*: o fornecedor não dá nenhuma garantia de que a QoS manter-se-á dentro dos limites.
- *Compulsório*: o fornecedor deve periodicamente monitorar o serviço, interrompendo-o caso a QoS não possa ser mantida dentro dos limites.
- *Garantido*: a manutenção da QoS dentro dos limites deve ser garantida de qualquer forma, o que implica que o usuário só será aceito se a QoS desejada puder ser sempre mantida.

Todas essas informações podem também ser organizadas através do uso de qualificadores, se associarmos, por exemplo, valores de parâmetros de caracterização de carga e de especificação da QoS a regras de estabelecimento e manutenção de contratos ou a níveis de serviço.

2.9 Taxonomia de Serviços

Embora os requisitos dos usuários possam variar em detalhes entre fluxos distintos a serem providos por um mesmo serviço (como veremos no Capítulo 4), os

parâmetros de interesse para o fornecedor e, consequentemente, as políticas de provisão de QoS adotadas, serão normalmente estáveis, e determinados pelo conjunto de serviços oferecidos pelo fornecedor.

Não se espera que todo fornecedor ofereça todos os serviços existentes. Geralmente, a definição do conjunto de serviços oferecidos pelo fornecedor depende em grande parte do tipo de ambiente considerado, ou seja, das grandezas envolvidas. Logo, fornecedores devem ser projetados e configurados de acordo com um conjunto de parâmetros predeterminados, que indicarão que políticas de provisão de QoS poderão ser usadas.

Os parâmetros de caracterização de serviços podem ser divididos em subconjuntos (não necessariamente disjuntos) que definem *categorias de serviço* que um fornecedor pode oferecer. Numa visão geral, uma categoria de serviço relaciona um conjunto de parâmetros a um determinado tipo de ambiente. Assim, a definição das categorias de serviço oferecidas pelo fornecedor leva à escolha dos parâmetros a serem gerenciados por ele. Em [ISO95], algumas categorias genéricas de serviço são sugeridas, entre elas:

- *Serviços multimídia*, para aqueles usuários que requerem que os dados de diversas mídias sejam eficientemente processados e comunicados no ambiente;

- *Serviços seguros* para aqueles usuários que requerem do ambiente garantias de proteção contra o acesso ou manipulação, intencional ou não, de seus dados por usuários ou componentes não autorizados;

- *Serviços confiáveis* para aqueles usuários que requerem do ambiente componentes altamente confiáveis e tolerantes a falhas;

A lista completa apresentada em [ISO95] não é exaustiva, pois teoricamente a quantidade de possíveis tipos de ambiente é infinita. Nada impede, e é bastante desejável, por exemplo, a existência de fornecedores de serviços multimídia seguros e altamente confiáveis. No presente trabalho, porém, somente a categoria de serviços multimídia será abordada de modo mais abrangente.

2.10 Sumário

Neste capítulo, a terminologia básica a ser usada na descrição do Framework Para Provisão de QoS foi introduzida. Também foi apresentado, a partir de um conjunto de princípios relacionados à provisão de QoS, um modelo genérico de operação para sistemas multimídia, de cujas fases foram extraídos alguns conceitos centrais, dentre os quais os de *parâmetros de caracterização de serviços* e de *mecanismos de provisão de QoS*.

Principalmente no que concerne a esses dois conceitos, as diferentes fases de provisão de QoS em um fornecedor apresentam padrões de estrutura e de comportamento que independem de quaisquer escalas de distribuição, níveis de abstração ou serviços específicos. É levando em consideração essa característica que o Framework para Provisão de QoS será descrito no Capítulo 4. Anteriormente, porém, o Capítulo 3 apresentará alguns trabalhos relacionados à questão do suporte à provisão de QoS em vários ambientes, distintos quanto à escala de distribuição e ao nível de abstração de seus componentes. As abordagens sugeridas por estes trabalhos servirão como fonte de inspiração para melhor esmiuçar os padrões de estrutura e de comportamento a serem modelados pelo Framework para Provisão da QoS.

Capítulo 3

Trabalhos Relacionados

Há nos centros de pesquisa atuantes nas áreas de sistemas operacionais, redes de comunicação, computação distribuída e afins, inúmeros trabalhos que tratam do problema da provisão de QoS. Devido à grande quantidade de abordagens existentes, este capítulo apresentará somente aquelas que tiveram maior influência na definição do Framework para Provisão da QoS, seja no que se refere à sua modelagem propriamente dita, ou no que concerne a possíveis casos de uso do mesmo. A maioria dos trabalhos apresentados neste capítulo insere-se no contexto de padrões definidos por órgãos internacionais[13], como a ISO, o ITU-T, o *Internet Engineering Task Force* (*IETF*) e o *Object Management Group* (*OMG*). Considerações a respeito desses padrões serão também colocadas, conforme necessário, ao longo do capítulo.

Além de trabalhos relacionados à provisão de QoS em sistemas operacionais, redes e protocolos de comunicação, arquiteturas de sistemas multimídia e modelos de computação distribuída, é apresentada também uma seção que toca na questão da aplicação de frameworks e design patterns no desenvolvimento de sistemas orientados a objetos, mais especificamente nos sistemas multimídia.

[13] Apesar do IETF e do OMG não serem órgãos internacionais legalmente constituídos para o desenvolvimento de padrões *de jure* (como a ISO e o ITU-T), o uso maciço das arquiteturas por eles definidas faz com que as soluções propostas por estes dois órgãos sejam largamente aceitas, tornando-as padrões *de facto*. O IETF define padrões para a Internet, enquanto o OMG descreve padrões de computação distribuída orientada a objetos.

3.1 Sistemas Operacionais

Na área de sistemas operacionais, a maior parte dos trabalhos relacionados à provisão de QoS têm focalizado os mecanismos de escalonamento de recursos nas estações finais. Dentre tais recursos, os processadores são os que têm recebido um tratamento mais amplo. O compartilhamento de outros recursos tem sido estudado em contextos mais específicos. Por exemplo, pesquisas e implementações de esquemas de multiplexação de acessos a disco, como as relatadas em [PAN95] e [YU93], têm caminhado quase que exclusivamente na direção de serviços de recuperação[14], especialmente os de *multimídia sob demanda* (*Multimedia-On-Demand – MOD*). Já com relação ao problema de se prover garantias de QoS durante o remapeamento de páginas virtuais de memória (armazenadas em disco) em espaços de endereçamento reais da memória (*page frames*), ele tem sido solucionado através da definição de mecanismos de alocação estática desses page frames, como descrito na proposta POSIX.4[15] [GALLMEISTER95], candidata a se tornar padrão pela ISO. A adoção desta solução deve-se, em grande parte, ao fato do procedimento de paginação entre memória e disco consumir muito tempo de processamento, o que torna muito complexa a implementação de mecanismos mais dinâmicos.

3.1.1 Estratégias de Escalonamento de Processadores

No que se refere ao escalonamento de processadores, os sistemas operacionais tradicionais adotam estratégias fixas, normalmente baseadas em prioridades definidas pelas aplicações [SILBERSCHATZ95]. Entretanto, requisitos de processamento podem variar de aplicação para aplicação, e descrever esses requisitos somente por intermédio de prioridades inviabiliza uma provisão adequada de QoS. Por exemplo, aplicações que manipulam fluxos de dados de mídias contínuas normalmente se valem de threads periódicas[16] durante o processamento dos mesmos, logo é bastante

[14] Segundo a classificação do ITU-T para aplicações multimídia [ITUT.I.211], *serviços de recuperação* fornecem a facilidade de recuperação de informação armazenada remotamente. Entre as aplicações que se enquadram nesses serviços, podemos citar videotexto, livrarias eletrônicas e vídeo sob demanda.

[15] POSIX (*Portable Operating System Interface*) é um conjunto de padrões e propostas que visam a especificação de interfaces que permitam a implementação de aplicações portáveis entre sistemas operacionais distintos. A proposta POSIX.4 aborda questões relacionadas à provisão de serviços de tempo real a essas aplicações.

[16] *Threads periódicas* podem ser definidas como threads que executam um mesmo conjunto de instruções continuamente, e em intervalos regulares.

desejável que o parâmetro *período* seja oferecido pelo sistema operacional. A adoção de parâmetros mais específicos permite que estratégias adequadas de escalonamento possam ser utilizadas pelo sistema operacional. Como exemplo, a estratégia de escalonamento *Earliest Deadline First* (*EDF*) utiliza, entre outros parâmetros, o período para escalonar corretamente um processador entre threads periódicas. Além do EDF, outras estratégias de escalonamento adequadas a threads periódicas (que utilizam diferentes conjuntos de parâmetros) são apontadas em [LIU73]. A Figura 4 ilustra o relacionamento entre os parâmetros utilizados pelo EDF.

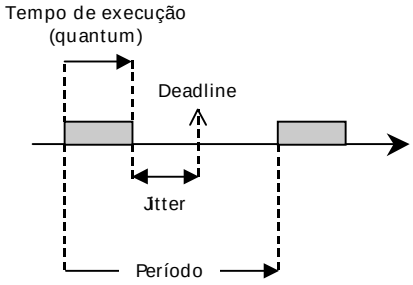


Figura 4: relação entre parâmetros usados na estratégia de escalonamento EDF.

O EDF define que, a cada ação de escalonamento[17], a próxima thread a ser selecionada (dentre as threads ainda não selecionadas nos seus respectivos períodos correntes) deve ser aquela cujo deadline esteja mais próximo. O parâmetro deadline tem correspondência direta com a variação máxima da periodicidade (*jitter*) permitida a uma thread. A Figura 5 mostra um exemplo de funcionamento de um mecanismo de escalonamento que utiliza o EDF.

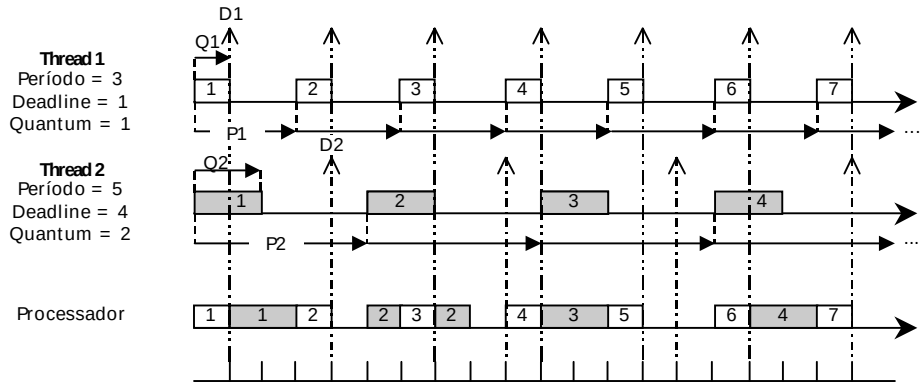


Figura 5: exemplo de escalonamento através da estratégia EDF.

[17] Uma ação de escalonamento é disparada sempre que se inicia um novo período de uma thread, ou que a thread atualmente selecionada tenha terminado a execução referente ao seu período corrente.

O EDF é uma das estratégias de escalonamento mais atraentes para threads periódicas, uma vez que ele sempre encontra um escalonamento válido (isto é, que honra os deadlines de todas as threads), se a capacidade do processador não for excedida. Em [LIU73], é apresentada também uma estratégia de admissão associada ao EDF que permite que essa capacidade seja sempre respeitada. Porém, uma das limitações impostas pelas estratégias de admissão e escalonamento apresentadas em [LIU73] é assumir que o deadline de uma thread não selecionada no seu período corrente é sempre igual ao final deste período, o que, por vezes, proíbe uma melhor utilização da capacidade do processador. Em [MAUTHE96], são apresentadas variações dessas estratégias que visam um maior aproveitamento da capacidade dos processadores, através da permissão do uso de deadlines mais restritos.

3.1.2 Escalonamento Hierárquico de Processadores

A necessidade de dar suporte ao processamento integrado de diferentes mídias requer que o sistema operacional permita o uso de diferentes conjuntos de parâmetros pelas aplicações e, consequentemente, que diferentes estratégias de escalonamento possam atuar concorrentemente sobre um mesmo processador. Neste sentido, uma abordagem interessante é a do *escalonamento hierárquico* do processador, utilizado em trabalhos como o *Hierarchical CPU Scheduler* [GOYAL96] e o *CPU Inheritance Scheduling* [FORD96]. Esta abordagem permite a um sistema operacional multiplexar a utilização do processador entre diferentes categorias de processamento, através de uma determinada estratégia de escalonamento. Estas categorias, por sua vez, podem ter suas parcelas de utilização do processador multiplexadas (possivelmente usando uma estratégia diferente de escalonamento) entre subcategorias, de modo recursivo. A Figura 6 mostra um exemplo de um processador sendo escalonado de maneira hierárquica.

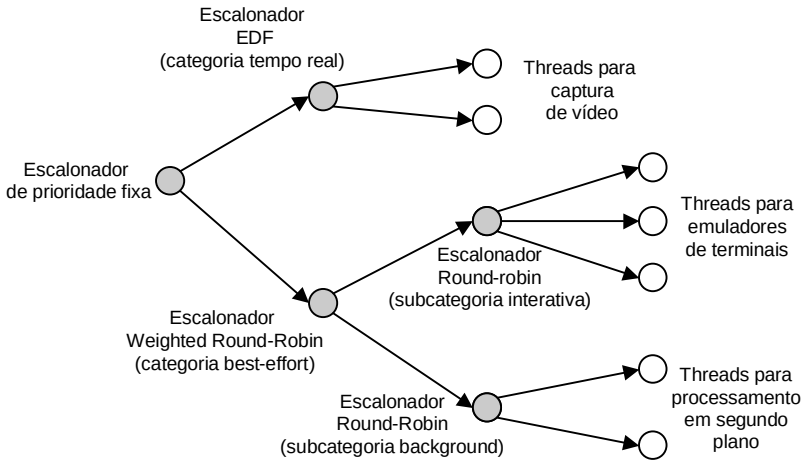


Figura 6: exemplo de escalonamento hierárquico de processadores.

A abordagem de escalonamento hierárquico é suficientemente flexível para permitir a introdução de novas categorias de processamento, através da inserção de estratégias de escalonamento em pontos adequados da hierarquia. Devido a esta característica, no presente trabalho a abordagem de escalonamento hierárquico é generalizada para o caso do compartilhamento de recursos quaisquer.

3.2 Redes de Distribuição

Com relação às redes de distribuição, a maior parte dos trabalhos que tratam do problema da provisão de QoS tem girado em torno da utilização da tecnologia ATM. Esta é a tecnologia adotada como o modo de transferência padrão a ser usado nas *Redes Digitais de Serviços Integrados de Faixa Larga* (*Broadband Integrated Service Digital Networks – B-ISDN*), segundo a recomendação I.121 [ITUT.I.121] existente na padronização definida pelo ITU-T[18].

Na tecnologia ATM, o termo *assíncrono* designa a maneira como parcelas de utilização da capacidade dos vários enlaces que interligam os nós de uma rede são alocadas entre usuários do ambiente de comunicação. A capacidade dos enlaces é dividida em fatias de tempo de tamanho fixo, denominadas de *células*, que são alocadas dinamicamente aos usuários, conforme a necessidade dos mesmos. A técnica

[18] Outro órgão importante para a definição de padrões relativos à tecnologia ATM, que trabalha em cooperação com o ITU-T, é o *ATM Forum*. Fundado em 1991, o ATM Forum é um consórcio internacional de empresas interessadas em acelerar o desenvolvimento e implantação da tecnologia ATM. Alguns dos trabalhos desenvolvidos junto ao ATM Forum serão apresentados no decorrer deste capítulo.

de alocação dinâmica da capacidade dos enlaces, utilizada em tecnologias tradicionais de comutação de pacotes e explorada ao máximo pela tecnologia ATM[19], dá às redes de distribuição um potencial ganho estatístico de utilização da capacidade dos enlaces, além de possibilitar que diferentes serviços de comunicação sejam providos em cima destas redes.

Por outro lado, o ganho estatístico em uma rede de distribuição tem como efeito colateral a possibilidade de sobrecarga de recursos localizados nos vários nós da rede, principalmente buffers. Isto pode gerar desde variações no retardo de transição de dados entre usuários do ambiente de comunicação, até perdas desses dados, dependendo do grau de sobrecarga. Por isso, várias pesquisas têm sido feitas (boa parte delas acerca da tecnologia ATM) visando projetar redes de distribuição que sejam consistentes com as requisições de serviço dos usuários, sob o ponto de vista de provisão de QoS, em face do comportamento estatístico dessas redes. Estas pesquisas têm seguido, em geral, nas seguintes direções:

- A definição de parâmetros de caracterização de carga e de especificação de QoS propícios às várias categorias de serviço;
- A esquematização de estratégias de escalonamento da capacidade dos enlaces que atendam satisfatoriamente aos serviços requisitados; e
- O desenvolvimento de mecanismos de controle de admissão e de alocação de recursos que aceitem ou rejeitem novas requisições, para que sobrecargas na rede sejam evitadas.

3.2.1 Caracterização de Tráfego

Com relação à parametrização de serviços nas redes de distribuição, a maior parte dos trabalhos tem focalizado a definição de modelos de *caracterização de tráfego*. Diversos modelos têm sido identificados, sendo divididos, na literatura, em dois tipos básicos: os modelos *estocásticos*, como o *Markov-modulado* [ANICK82] e o *Auto-*

[19] A tecnologia ATM pode ser descrita como uma técnica de comutação rápida de pacotes baseada na transferência de unidades de informação de tamanho fixo (células). Grande parte das suas características (inclusive a da alocação dinâmica da capacidade dos enlaces) é herdada das tecnologias tradicionais de comutação de pacotes, por isto, neste capítulo, referências ao termo *rede de distribuição* englobarão ambas as tecnologias, a não ser quando for necessário explicitar as diferenças entre elas.

similar [GARRETT94]; e os modelos *determinísticos*, como o *Leaky-bucket* [TURNER86] e o *{PCR, SCR, BT}* [ATM93], este último proposto pelo ATM Forum[20].

Devido ao fato desses modelos serem definidos e aplicados de forma a abranger categorias genéricas de tráfego, eles apresentam, como principal deficiência, a pouca precisão em capturar a natureza do tráfego gerado por fluxos de dados de determinadas mídias. Outros modelos mais recentes abrangem categorias específicas, como o *D-BIND* [KNIGHTLY97], por exemplo, definido especificamente para capturar as propriedades de geração de tráfego de vídeos com compressão e compactação. A adoção de modelos de caracterização de tráfego específicos (juntamente com parâmetros adequados de especificação de QoS) para determinadas categorias de serviço é o primeiro passo a ser dado durante o projeto de uma rede de distribuição que visa, ao mesmo tempo, dar suporte integrado a diferentes mídias, e ser flexível o suficiente para permitir a introdução de novos serviços. O presente trabalho segue esta linha de raciocínio. O Framework para Provisão da QoS possibilita aos projetistas de sistemas estruturar a parametrização das redes de distribuição (no que se refere à provisão de QoS), dando margem assim à flexibilização dos serviços a serem oferecidos por elas.

3.2.2 Esquemas de Repasse

O principal objetivo dos *esquemas de repasse* atuantes nos diversos nós que compõem uma rede de distribuição é prover aos fluxos de dados dos usuários as garantias de desempenho por eles requisitadas, explorando o ganho estatístico decorrente da alocação dinâmica da capacidade dos enlaces dessa rede. Isto é conseguido, em cada nó, através do escalonamento dos enlaces de saída entre as células que aguardam a vez de serem transmitidas.

Em [KUROSE93], é apresentada uma classificação das abordagens mais freqüentemente usadas na definição de esquemas de repasse. Estas abordagens são quase sempre baseadas na simples multiplexação de células, sem considerar a diversidade de serviços oferecidos aos usuários do ambiente de comunicação. Em

[20] A recomendação I.350 [ITUT.I.350] do ITU-T descreve as noções de *QoS* e *descritores de tráfego* em redes B-ISDN. Com base nessa recomendação, o ATM Forum definiu alguns modelos de caracterização de tráfego para redes com tecnologia ATM, sendo um deles o {PCR, SCR, BT}, que utiliza os parâmetros *taxa de pico de geração de células* (*Peak Cell Rate – PCR*), *taxa média de transferência de células* (*Sustainable Cell Rate – SCR*) e *tolerância à explosividade* (*Burst Tolerance – BT*).

trabalhos mais recentes, a analogia *{thread, processador}* e *{célula, enlace}* tem sido bastante explorada na direção da definição de redes de distribuição cujo suporte a múltiplos serviços ocorre explicitamente em seus nós intermediários, através do escalonamento hierárquico dos enlaces. Esta abordagem é formalmente introduzida em [GOYAL96a], que utiliza basicamente a mesma idéia apresentada no *Hierarchical CPU Scheduler*.

3.2.3 Protocolos de Configuração de Fluxos

Apesar da importância da caracterização de tráfego e dos esquemas de repasse, é através dos protocolos de configuração de fluxos que se consegue fundamentalmente prover uma QoS fim-a-fim nas redes de distribuição. São eles os responsáveis por configurar, através dos mecanismos de controle de admissão e de alocação de recursos, os nós por onde passarão os fluxos de dados requisitados pelos usuários. Dentre as características mais desejáveis desses protocolos estão a facilidade e eficiência em acomodar fluxos ponto-a-multiponto, e a robustez e flexibilidade da configuração, incluindo esquemas que lidem com ocasionais falhas na rede.

Vários foram os trabalhos desenvolvidos nos últimos anos que tratam da questão da configuração de fluxos em redes de distribuição. Boa parte deles originou-se dos grupos de pesquisa do IETF. Dois dos trabalhos mais importantes gerados por esses grupos de pesquisa foram o desenvolvimento do *STream Protocol Version 2* (*ST-II*) [TOPOLCIC90] e do *Resource ReSerVation Protocol* (*RSVP*) [ZHANG93].

O ST-II é um conjunto de protocolos (vide Figura 7) projetado para a configuração (SCMP) e transmissão (ST) de fluxos de áudio e vídeo através da Internet. Ele é parte da família de protocolos IP [COMER95] e serve como adendo ao mesmo, podendo ambos coexistirem nos mesmos nós. O ST-II dá suporte à alocação de recursos iniciada a partir das estações de onde se originam os fluxos, oferecendo a maioria das características desejáveis em um protocolo de configuração. Porém, para dar suporte a todas essas características, o ST-II tornou-se dificílimo de ser implementado, além de exigir das estações de onde se originam os fluxos uma capacidade de processamento que muitas vezes elas não possuem, pelo fato de toda a modificação de fluxos ter que partir destas estações.

Em contraste com o ST-II, o RSVP dá suporte à alocação de recursos iniciada a partir das estações destinatárias dos fluxos, distribuindo de modo mais eficiente a responsabilidade pela configuração destes fluxos entre os outros nós da rede. O RSVP funciona basicamente como um protocolo de controle para o IP (da mesma forma que o SCMP com relação ao ST), e também oferece a maioria das características desejáveis em um protocolo de configuração de fluxos. Entretanto, uma vez que o RSVP possui um mecanismo de alocação orientado aos destinatários dos fluxos, ele se adapta pior ao modelo de comutação de células do que o ST-II, pelo fato da maioria das tecnologias que se utilizam de células (incluindo ATM) assumir que a alocação de recursos é estabelecida a partir das estações que são origem dos fluxos.

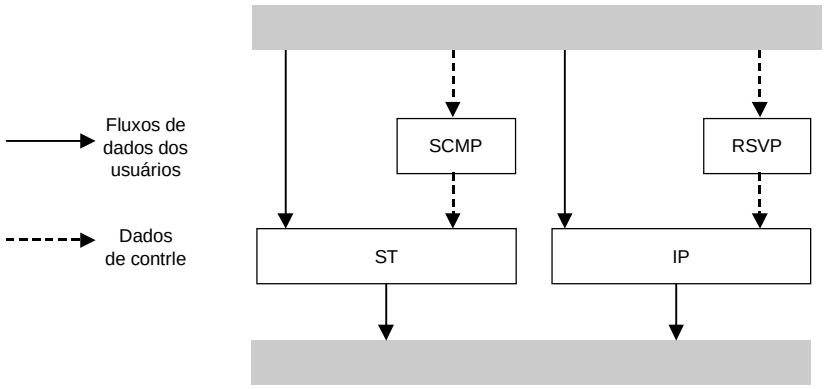


Figura 7: relacionamento entre o ST-II, o RSVP e o IP.

Há uma infinidade de outros protocolos, como o RCAP [BANERJEA91] (componente do serviço de transporte de mídias contínuas Tenet [FERRARI95]), e a especificação Q.93B [ATM93] (publicada pelo ATM Forum[21]), que abordam a configuração de fluxos em redes de distribuição, sob inúmeras óticas diferentes. Em todos estes protocolos, não é explicitado como os gerentes de recursos presentes nos diferentes nós da rede controlam a admissão de novos fluxos e alocam os recursos necessários para a oferecimento dos serviços requisitados. Esta característica é explorada no Framework para Provisão da QoS, através da definição de mecanismos de de controle de admissão e de alocação de recursos independentes do mecanismo de configuração, o que permite às redes de distribuição alcançar uma grande flexibilidade com relação aos serviços a serem oferecidos.

[21] A recomendação Q.2931 [ITU.T.Q.2931] define o protocolo de sinalização (processo utilizado para o estabelecimento, supervisão e rompimento de conexões) das redes B-ISDN na *Interface Usuário-Rede* (*User-Network Interface - UNI*). A especificação Q.93B define um protocolo de sinalização para redes com tecnologia ATM, baseado em uma simplificação da Q.2931.

3.3 Protocolos de Comunicação Fim-a-fim

Ao se falar de protocolos de comunicação, não se pode deixar de citar o RM-OSI [ISO84]. O RM-OSI é um modelo definido pela ISO, que propõe uma estrutura com sete camadas como referência para a definição de arquiteturas de protocolos de comunicação, como mostra a Figura 8.

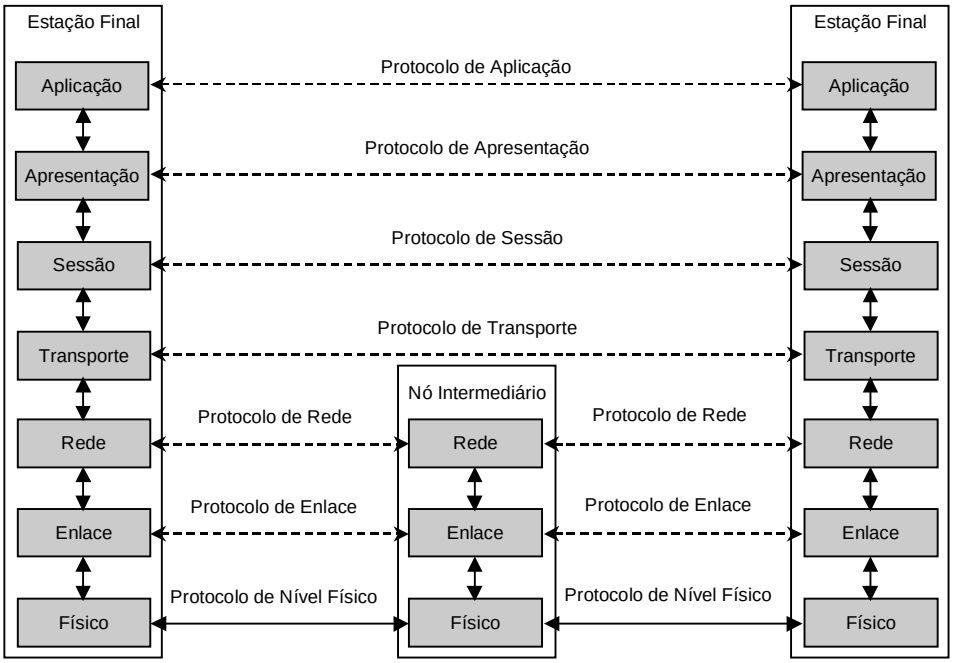


Figura 8: camadas definidas pelo RM-OSI.

Apesar de, no RM-OSI, o conceito de QoS ser introduzido na camada de transporte, o serviço oferecido pela maioria dos protocolos definidos nessa camada (seja pela própria ISO [ISO86], ou por outras organizações, como o IETF [POSTEL81]) provê o que é chamado de *QoS de melhor esforço*, isto é, estes protocolos não dão nenhuma garantia de que a QoS será realmente mantida. Trabalhos mais recentes relacionados à provisão de QoS em protocolos de comunicação fim-a-fim têm sido guiados basicamente na direção da formulação de serviços e protocolos da camada de transporte (segundo a nomenclatura RM-OSI) que sejam configuráveis com relação à QoS, e que permitam a negociação da QoS entre os usuários destes serviços.

3.3.1 Serviços e Protocolos de Transporte com QoS Configurável

Vários trabalhos têm investigado a provisão efetiva de QoS na camada de transporte. Dentre eles, um bastante significativo é o projeto ESPRIT OSI 95 (ligado ao grupo de trabalho SC6 ECFF da ISO), que propõe um serviço de transporte baseado em um conjunto de protocolos denominado de *TPX* [DANTHINE94]. O TPX dá suporte a serviços ponto-a-ponto ou ponto-a-multiponto, orientados ou não à conexão. Todos estes serviços são parametrizáveis no que concerne à vazão, ao retardo, à variação estatística do retardo e à política de tratamento de erros. Além do nível de serviço de melhor esforço, o TPX fornece outros, indicados através de *qualificadores de semântica*. No TPX, podem ser atribuídos vários valores a um parâmetro, sendo um para cada semântica diferente. Dentre os qualificadores definidos, o *compulsório* e o *threshold* são os mais importantes. Quando um valor de um parâmetro é qualificado como compulsório, o protocolo de transporte fica responsável por monitorar o fluxo de dados do usuário, e eventualmente por interrompê-lo, se o valor for violado. Já a qualificação de um valor de um parâmetro como threshold obriga o protocolo a monitorar o fluxo, porém, violações desse valor são simplesmente informadas ao usuário do serviço. Em [DANTHINE94], regras de negociação são estabelecidas no TPX para cada qualificador de semântica de cada parâmetro especificado nas requisições de serviço.

Outros trabalhos, como o CMTP [WOLFINGER91] (componente do Tenet) e o RTP [SCHULZRINNE95] (componente da família de protocolos IP), também definem serviços e protocolos fim-a-fim que dão suporte ao envio de dados de múltiplas mídias e à negociação da QoS. Qualquer que seja o modo como estes protocolos são formulados, os serviços por eles oferecidos são normalmente dependentes do suporte e da flexibilidade oferecidos pela rede de distribuição sobre a qual eles são implementados. Por exemplo, uma rede de distribuição baseada na tecnologia ATM deverá permitir uma gama maior de serviços de transmissão e um transporte mais eficiente das diferentes mídias do que outra baseada em tecnologias tradicionais de comutação de pacotes.

Porém, os serviços e protocolos fim-a-fim citados nesta subseção restringem o conjunto de parâmetros oferecidos aos usuários, potencialmente limitando a flexibilidade oferecida pelas redes de distribuição. A própria padronização do ITU-T para as redes B-ISDN apresenta essa limitação. A Figura 9 ilustra o *Modelo de*

Referência para Protocolos (*Protocol Reference Model - PRM*) definido pelo ITU-T para redes B-ISDN [ITUT.I.321].

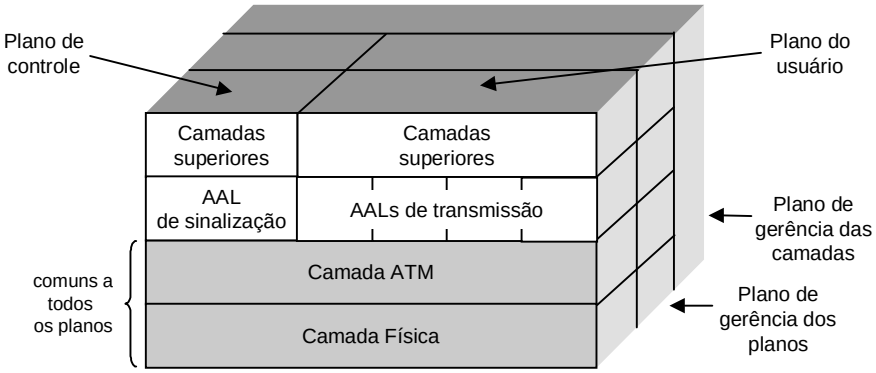


Figura 9: modelo de referência para protocolos de redes B-ISDN.

As diferentes *camadas de adaptação* (*ATM Adaptation Layers* - AALs) definidas pelo ITU-T [ITUT.I.363], em conjunto com as classes de serviço sobre elas especificadas [ITUT.I.362], correspondem a protocolos fim-a-fim que fazem uso da flexibilidade oferecida pela tecnologia ATM. Apesar disso, essa flexibilidade é limitada pelo conjunto de classes de serviço modeladas pelas AALs, como ilustra a Figura 10.

Classes de serviço e AALs relacionadas	AAL 1	AAL 2	AAL 3/4 e AAL 5	
	Classe A	Classe B	Classe C	Classe D
Tempo na fonte e no destino	Relacionado (compensação da variação estatísca do retardo)		Sem relação	
Taxa de geração de bits na origem	Constante (CBR)	Variável (VBR)		
Modo de conexão	Orientado à conexão			Sem conexão

Figura 10: classes de serviço e AALs definidas nas recomendações I.362 e I.363 do ITU-T.

O presente trabalho segue uma linha mais flexível, no sentido de que a definição do Framework para Provisão de QoS fornece uma capacidade de extensão que pode ser utilizada para a configuração de protocolos fim-a-fim voltados especificamente ao serviço desejado, sem a necessidade da definição de um conjunto previamente estabelecido de categorias ou parâmetros que caracterizem o serviço.

3.4 Arquiteturas de QoS

Conforme foi dito na Seção 1.1, coordenar corretamente o compartilhamento de recursos nos diversos subsistemas presentes em um ambiente distribuído é uma tarefa que depende, em grande parte, da indicação da QoS desejada pelos usuários. Contudo, a maioria dos progressos conseguidos na área de provisão de QoS aconteceu isoladamente em subsistemas distintos (como foi apresentado anteriormente neste capítulo) e, por isso, vários desses subsistemas não abordam o nível de visão de QoS dos usuários de ambientes distribuídos. Em face da existência desta lacuna entre os níveis de visão de QoS dos subsistemas e o dos usuários de ambientes distribuídos, novos modelos arquiteturais para provisão de QoS têm sido propostos. Estas arquiteturas de QoS definem, basicamente:

- Interfaces de requisição de serviço condizentes com o nível de visão de QoS compreendido pelos usuários de um ambiente distribuído, e
- Mecanismos responsáveis por traduzir as requisições dos usuários em alocações de recursos nos diversos subsistemas presentes no ambiente distribuído, não deixando, portanto, transparecer aos usuários detalhes acerca, por exemplo, do escalonamento de processadores nas estações e da configuração de fluxos na redes de distribuição.

Dois exemplos bastante relevantes de arquiteturas de QoS encontradas na literatura são apresentadas nas subseções a seguir.

3.4.1 Arquitetura de QoS de Lancaster

Desde 1993, o projeto QoS-A da Universidade de Lancaster [CAMPBELL93] tem desenvolvido uma arquitetura que promove a idéia de *QoS integrada*, abrangendo estações finais e redes de distribuição (em especial, aquelas que adotam tecnologia ATM), e que objetiva, em primeiro plano, dar suporte a fluxos de dados de mídias contínuas em ambientes distribuídos. Em termos funcionais, a arquitetura QoS-A é dividida em camadas e planos, como ilustrado na Figura 11.

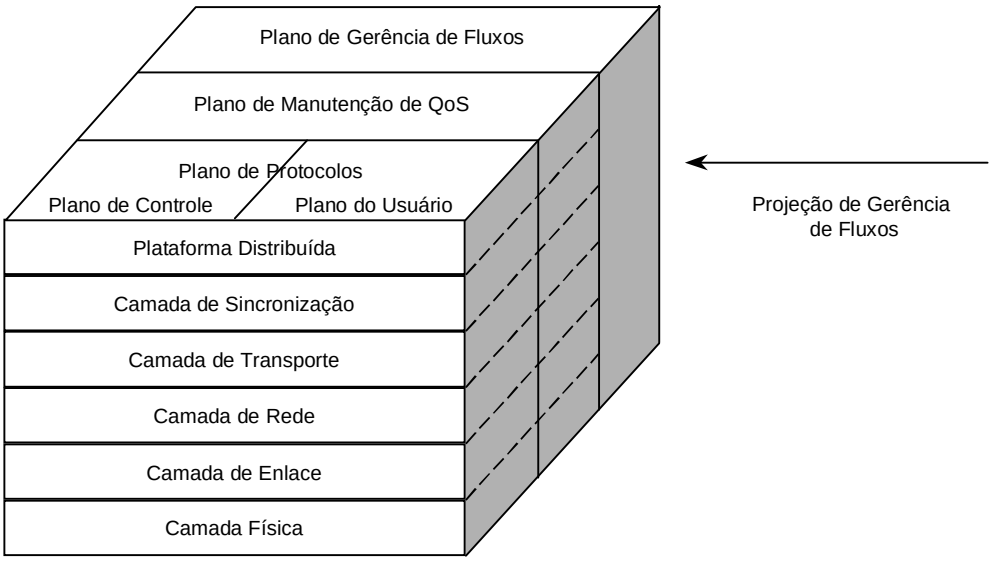


Figura 11: arquitetura QoS-A.

A camada superior define uma *plataforma para aplicações distribuídas* que provê QoS no nível de visão dos componentes dessas aplicações [COULSON93]. Abaixo desta camada está a que provê serviços de *sincronização* entre fluxos [CAMPBELL92]. Provendo os serviços de comunicação fim-a-fim necessários a estas duas camadas, está a camada de *transporte*, que define um conjunto de protocolos configuráveis com relação à QoS [GARCIA93]. Abaixo de todas elas, estão as camadas inferiores, que formam a base da rede de distribuição.

Na arquitetura QoS-A, a gerência da QoS é definida em três planos verticais, que correspondem às escalas de tempo em que a mesma é realizada (como regem os princípios da separação e da gerência assíncrona – Seção 2.5):

- *Plano de protocolos*: consiste em dois subplanos distintos; de *usuários* e de *controle*. A arquitetura QoS-A separa os dois subplanos em face da diferença entre os requisitos de QoS impostos pelos fluxos de dados de mídias contínuas, e os impostos pelos dados de controle (sinalização) associados à configuração desses fluxos.

- *Plano de manutenção de QoS*: define um conjunto de *gerentes de QoS* atuantes em cada uma das camadas, sendo cada um deles responsável pela monitorização e sintonização da QoS na camada correspondente, com base no contrato de serviço estabelecido com o usuário.

- *Plano de gerência de fluxos*: responsável pelo estabelecimento de contratos de serviço, incluindo o controle de admissão e a pré-alocação (*reserva*) de recursos, e pelo mapeamento e sintonização da QoS entre as camadas.

A *projeção de gerência de fluxos* da arquitetura QoS-A (área hachurada da Figura 11) define o relacionamento entre os três planos, que trabalham em conjunto para estabelecer, monitorar e manter a QoS fim-a-fim. Esta projeção é apresentada em mais detalhes na Figura 12, que mostra somente as camadas de transporte e inferiores; as camadas superiores foram omitidas para não "poluir" demais a figura.

Em cada camada, os vários mecanismos de cada plano apresentam interfaces bem definidas com seus parceiros. Assim, o suporte à QoS em uma determinada camada é dependente de interações entre os protocolos de controle e de transmissão, o gerente de QoS, os mecanismos do plano de gerência de fluxos e a camada inferior, bem como com os usuários do serviço provido pela camada. Por exemplo, na camada de transporte são definidas as seguintes interfaces (vide Figura 12):

- *F-TSAP*: esta interface, oferecida pelo plano de gerência de fluxos aos usuários da camada de transporte, é responsável por receber requisições de serviço. Uma requisição descreve *cláusulas* do contrato de serviço que um usuário deseja estabelecer, incluindo aí a especificação da QoS (em termos de parâmetros como vazão, retardo, variação estatística do retardo e tolerância a perdas), o nível de serviço e as ações a serem tomadas em decorrência de violações no contrato. Se, dentre estas ações, for definida a notificação ao usuário, esta interface é responsável também por alertá-lo sobre possíveis degradações na QoS oferecida.

- *D-TSAP*: interface oferecida pelo protocolo de transmissão aos usuários da camada de transporte. Ela permite aos usuários produzir e consumir os fluxos de dados estabelecidos através da interface F-TSAP.

- *Interfaces de Gerência de Recursos e de Monitorização (R e M)*: interfaces oferecidas pelo protocolo de transmissão ao gerente de QoS de transporte. A interface de gerência de recursos permite que um gerente faça a alocação dos recursos usados pelo protocolo de transmissão, e que

seja informado em caso de sobrecarga dos mesmos, enquanto a interface de monitorização permite que o gerente de QoS configure e controle a monitorização de fluxos no protocolo de transmissão. Os protocolos de transmissão usados na arquitetura QoS-A devem ser dotados de mecanismos internos responsáveis pela alocação dos recursos e pela monitorização dos fluxos, para poderem oferecer os serviços de alocação e monitorização requisitados pelos gerentes de QoS.

- *Interface de Manutenção de QoS (Q…)*: interface oferecida pelo gerente de QoS de transporte aos mecanismos presentes no plano de gerência de fluxos. Ela permite que estes mecanismos sejam informados de uma eventual degradação da QoS que não pôde ser tratada isoladamente pelo gerente de QoS de transporte. Desta forma, o mecanismo de sintonização presente no plano de gerência de fluxos pode ajustar a alocação de recursos em outras camadas, através da interface de manutenção de QoS dos gerentes de QoS das outras camadas, a fim de compensar a degradação da QoS fim-a-fim, ou então notificar o usuário através da interface F-TSAP. Em ambos os casos, o papel do mecanismo de mapeamento é fundamental, na medida em que tanto os gerentes de QoS de cada camada quanto os usuários do serviço de transporte possuem as suas próprias visões de QoS.

- *S-TSAP*: interface oferecida pelo protocolo de controle da camada de transporte aos mecanismos presentes no plano de gerência de fluxos. Esta interface permite que os mecanismos de controle de admissão, de reserva de recursos e de sintonização da QoS presentes no ambiente distribuído possam se comunicar entre si através do protocolo de controle[22]. Dessa forma, ao ser recebida uma requisição de serviço em uma estação final, um fluxo condizente com esta requisição pode ser configurado, e posteriormente monitorado, em todos os nós (sejam eles estações finais ou nós intermediários na rede distribuição) que participarão do oferecimento do serviço.

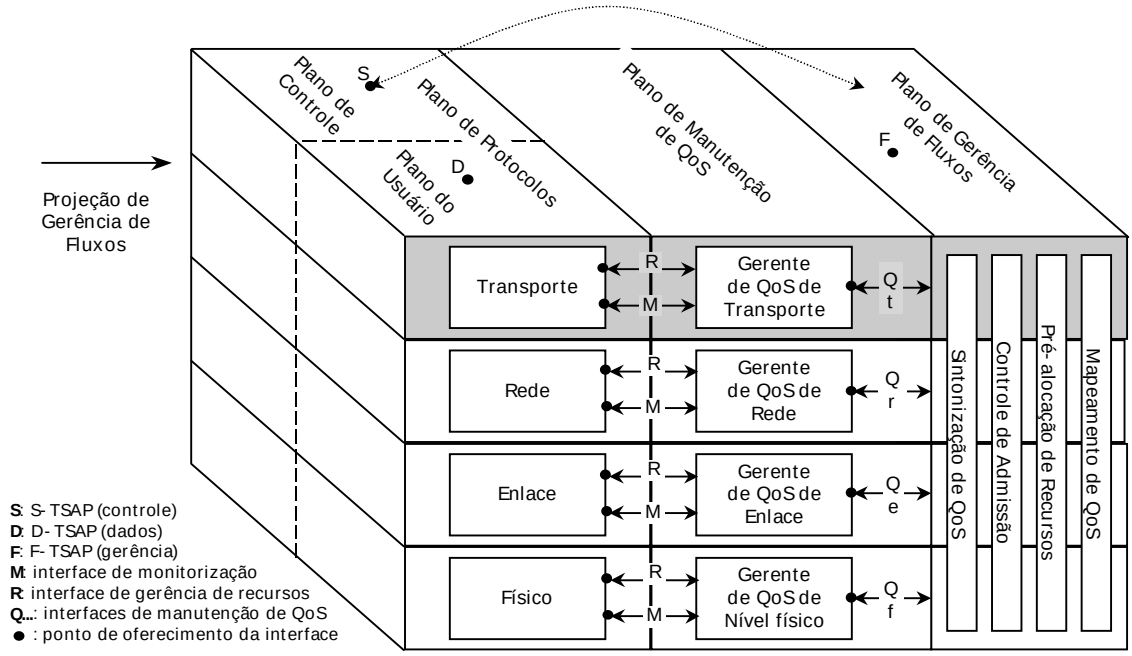


Figura 12: projeção de gerência de fluxos da arquitetura QoS-A.

Objetivando uma maior eficiência na gerência de fluxos, a arquitetura QoS-A define um esquema em que os recursos do ambiente distribuído são logicamente particionados em *domínios de gerência de fluxos*. Em cada domínio, os mecanismos presentes no plano de gerência de fluxos são implementados por um único *servidor de recursos*, localizado em algum nó contido no seu domínio. Cada servidor é responsável por consultar os gerentes de QoS presentes nos vários nós contidos no seu domínio, viabilizando o processo de configuração de fluxos através da integração do controle de admissão e da alocação de recursos nesses nós. Dessa forma, permite-se que determinados nós (em especial, os nós intermediários da rede de distribuição) não implementem o plano de gerência de fluxos, reduzindo assim a carga gerada pelo protocolo de controle, que passa a interligar somente os nós onde estão localizados os servidores de recursos do ambiente distribuído.

[22] Apesar da camada de transporte prover um serviço de comunicação fim-a-fim através do protocolo de transmissão, há entidades do protocolo de controle agindo também nos nós intermediários que compõem a rede de distribuição. Esta característica é semelhante a da AAL de sinalização, sobre a qual opera o protocolo de sinalização definido pela recomendação Q.2931.

3.4.2 Arquitetura OMEGA

Paralelamente ao projeto QoS-A, tem sido desenvolvida, na Universidade de Pennsylvania, a arquitetura OMEGA [NAHRSTEDT95a]. Esta arquitetura é centrada em um modelo de comunicação para estações finais, ilustrado na Figura 13. A arquitetura OMEGA pressupõe a existência de uma camada de rede que provê serviços de comunicação configuráveis com relação a QoS, e de sistemas operacionais capazes de prover QoS de processamento. A essência do modelo de comunicação reside na alocação e na gerência de recursos nas estações finais. O modelo de comunicação é dividido em duas camadas:

- *Camada de aplicação*: define o *Protocolo para Aplicações de Tempo-Real* (*Real-Time Application Protocol – RTAP*), que provê as funções de gerência conjunta de fluxos relacionados (*chamadas*), de gerência de dispositivos multimídia (câmeras, microfones, etc.), bem como de sincronização entre fluxos de uma mesma chamada.

- *Camada de transporte*: define o *Protocolo de Rede de Tempo-Real* (*Real-Time Network Protocol – RTNP*), que provê as funções de gerência de fluxos individuais e de controle da interface com a camada de rede.

Ambas as camadas devem prover serviços com garantias de QoS sobre as chamadas e fluxos criados para os usuários do ambiente distribuído. Para isso, é definido na arquitetura OMEGA um protocolo de configuração de fluxos, responsável por negociar a alocação de recursos para ambas as camadas. Os recursos utilizados por cada uma das camadas (seja na estação final ou internamente à camada de rede) são descritos através de parâmetros armazenados em bases de dados (*perfis*) que são acessíveis pelas entidades de protocolo dessas camadas, e pelas entidades do protocolo de configuração de fluxos. Entidades deste protocolo de configuração são denominadas, na arquitetura, de *QoS Brokers*. Um QoS Broker atua como um gerente de recursos da estação final na qual ele se localiza.

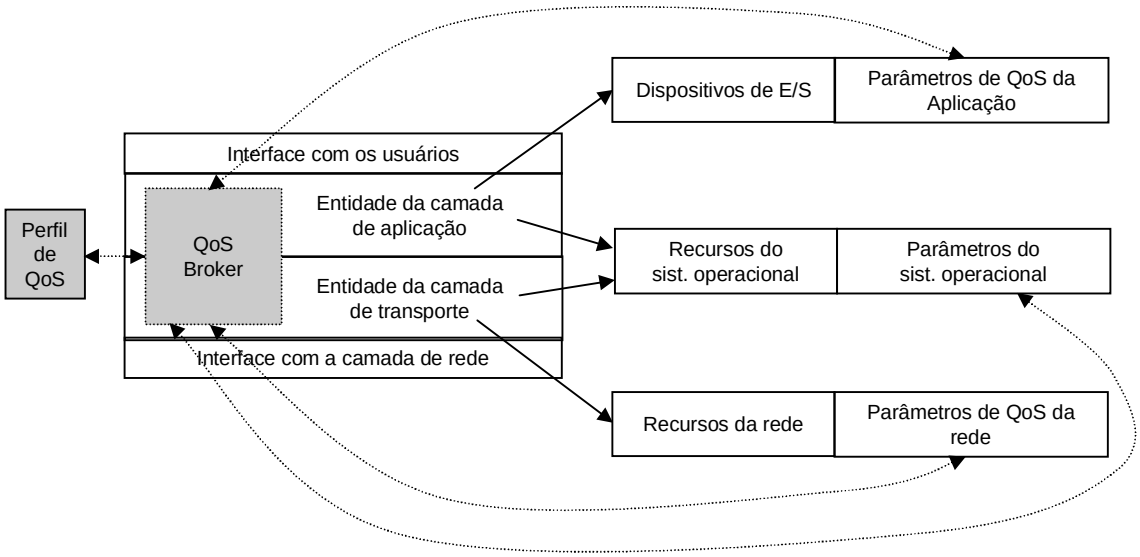


Figura 13: modelo de comunicação OMEGA em uma estação final.

À face de uma requisição de serviço, o QoS Broker é o responsável por:

- Orquestrar os recursos utilizados por ambas as camadas na sua estação final,
- Negociar com o mecanismo responsável pela gerência de recursos na camada de rede a QoS a ser por ela provida, e
- Negociar com os QoS Brokers localizados nas estações remotas que participarão do oferecimento do serviço a QoS que cada uma dessas estações proverá.

Contratos estabelecidos pelo protocolo de configuração também são armazenados em uma base de dados, denominada de *perfil de QoS*. Esta base de dados permite que sejam definidos mecanismos de sintonização que respondam pela manutenção dos contratos de serviço. Porém, tais mecanismos não são explicitamente modelados na arquitetura OMEGA.

Devido ao fato do protocolo de configuração de fluxos atravessar a fronteira entre as camadas de aplicação e transporte, pode-se incorporar a ele os seguintes tipos de comunicação entre os elementos da arquitetura OMEGA: entre camadas, entre entidades de uma mesma camada, e entre entidades de uma camada e os sistemas operacionais. Cada tipo de comunicação responde por um serviço diferente oferecido pelos QoS Brokers, como ilustra a Figura 14. A comunicação entre camadas é

provida pelo serviço de mapeamento da QoS. Já a comunicação entre entidades de uma mesma camada corresponde ao serviço de negociação. E a comunicação entre entidades de uma camada e os sistemas operacionais é responsabilidade do serviço de admissão, cuja função consiste em decidir se é viável a uma camada alocar determinados recursos na estação final.

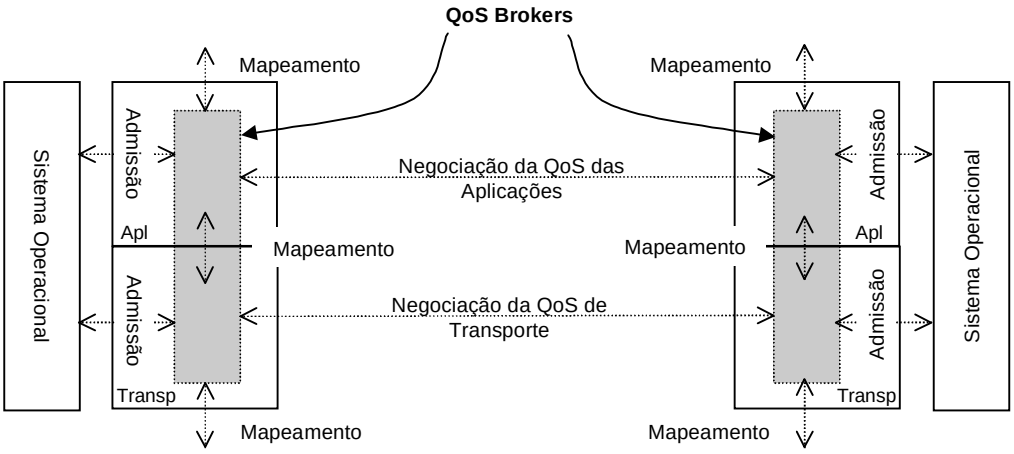


Figura 14: serviços oferecidos pelo QoS Broker.

3.4.3 Considerações a Respeito das Arquiteturas de QoS

Outros trabalhos, como o *Modelo de QoS de Heidelberg* [HERRTWICH92] e a *Arquitetura Fim-a-Fim MASI* [BESSE94], também tratam da integração entre sistemas operacionais com garantias de desempenho e serviços e protocolos de comunicação configuráveis com relação a QoS, todos objetivando prover efetivamente a QoS fim-a-fim desejada pelos usuários. Atualmente, não há um consenso sobre qual é a arquitetura de QoS mais adequada, dado um ambiente distribuído específico. Em [CAMPBELL95], é feito um estudo comparativo entre várias dessas arquiteturas, levando em conta, basicamente, a abrangência das mesmas com relação ao termo *fim-a-fim*.

Arquiteturas de QoS podem ser consideradas como frameworks para a integração da QoS entre os diferentes subsistemas presentes em um ambiente distribuído. Porém, elas são insuficientemente genéricas para abrangerem de maneira satisfatória todas as variedades de configurações que esse tipo de arquitetura deveria cobrir. Inúmeros são os exemplos da falta de flexibilidade dessas arquiteturas. Dentre eles, podemos citar como mais importantes:

- A pressuposição, de antemão, de um conjunto finito de parâmetros, o que, por si só, já limita o conjunto de categorias de serviço que as arquiteturas de QoS podem abranger.

- A adoção de abordagens de orquestração não configuráveis de acordo com a necessidade do ambiente distribuído. A arquitetura QoS-A, por exemplo, adota um abordagem cliente-servidor parcialmente centralizada (implementada pelos servidores de recursos dos domínios de gerência de fluxos). Já a arquitetura OMEGA adota uma abordagem de intermediação (implementada pelos QoS Brokers do protocolo de configuração de fluxos). Outras abordagens mais recentes [HAFID95] [GUEDES97] provêem esquemas mais flexíveis, mas igualmente insuficientes para abranger de maneira adequada todos os tipos de ambientes distribuídos.

- O uso de subsistemas com características de QoS predefinidas, nem sempre suficientes para envolver todas as categorias de serviço necessitadas pelos usuários do ambiente distribuído. Quando acontece de um subsistema não prover uma determinada categoria de serviço, os projetistas de sistemas são geralmente levados a estender estes subsistemas, de modo que estes passem a oferecer as características de QoS desejadas. Porém, raramente tais subsistemas são flexíveis o bastante para serem facilmente estendidos.

Em suma, todas essas limitações encontradas nas arquiteturas de QoS decorrem, em última instância, do fato de que a uniformização da provisão da QoS ocorre somente nas fronteiras entre os subsistemas. O Framework para Provisão de QoS vai de encontro a essas deficiências, provendo aos projetistas de sistemas estruturas suficientemente genéricas para serem aplicadas tanto na definição de arquiteturas de QoS como um todo, quanto na especificação da arquitetura interna dos vários subsistemas.

3.5 Modelos de Objetos com QoS

Uma outra abordagem para o problema da provisão de QoS que tem sido considerada em recentes propostas é a do oferecimento, nos ambientes distribuídos, de um *único* serviço configurável de acordo com os anseios dos usuários. De uma forma geral, a principal característica desses trabalhos é o uso de modelos computacionais OO que permitam a definição de pequenos componentes reutilizáveis (objetos). Como a diversidade de requisitos de processamento e comunicação é oriunda da necessidade dos usuários, futuramente parte da responsabilidade pela configuração do serviço deverá recair sobre os próprios usuários, seja através da conexão entre objetos preexistentes, ou da implementação de novos objetos específicos. Conforme mencionado em [LAZAR95], a "programabilidade" de serviços deverá ser uma característica chave em qualquer ambiente distribuído moderno. O projeto, implementação e configuração desses serviços deverá se tornar tão corriqueiro quanto a implementação das aplicações dos usuários, que passarão a atuar também como projetistas de sistemas, uma vez que ambas as tarefas estarão interligadas e completamente integradas.

Grande parte das propostas que seguem a abordagem de serviços configuráveis é baseada no RM-ODP [ISO95a], resultado de um esforço conjunto de padronização da ISO e do ITU-T, e na arquitetura CORBA [OMG96], criada pelo OMG. O RM-ODP especifica um modelo de computação aberta que dá suporte ao desenvolvimento de outros padrões que permitam que quaisquer tipos de serviços de processamento e comunicação sejam uniformemente oferecidos pelos ambientes distribuídos. O RM-ODP define que a especificação de um fornecedor de serviços em um ambiente distribuído aberto deve ser dividida em cinco visões, sendo duas delas de particular interesse para este trabalho:

- *Visão computacional*: trata da decomposição funcional do fornecedor em objetos que interagem através de interfaces. Estas interfaces podem corresponder a diversos tipos de interações, desde invocações de métodos através do tradicional estilo de requisição e resposta, até mensagens de fluxo contínuo (adequadas para mídias contínuas).

- *Visão de engenharia*: trata dos mecanismos necessários ao suporte a interações distribuídas entre os objetos (incluindo mecanismos de

provisão de QoS), de maneira independente de considerações como, por exemplo, os sistemas operacionais sobre os quais o serviço de processamento é oferecido aos objetos, a linguagem de programação utilizada para implementar esses objetos, etc.

A maior limitação do RM-ODP reside no fato de que ele é por demais abstrato, o que exige dos projetistas de sistemas um considerável esforço de refinamento do modelo, antes que ele possa ser implementado no contexto de um ambiente distribuído específico. Neste ponto, a arquitetura CORBA define de modo mais concreto um modelo de comunicação entre objetos distribuídos. A arquitetura CORBA define um conjunto de mecanismos que permitem a interação entre objetos de forma independente da sua distribuição e das linguagens de programação utilizadas na sua codificação, atuando assim como uma plataforma para aplicações distribuídas. Essa transparência é obtida através da atuação de um elemento central denominado *Object Request Broker* (*ORB*), representado pela seta grande da Figura 15. O núcleo do ORB é o responsável pela comunicação entre os objetos.

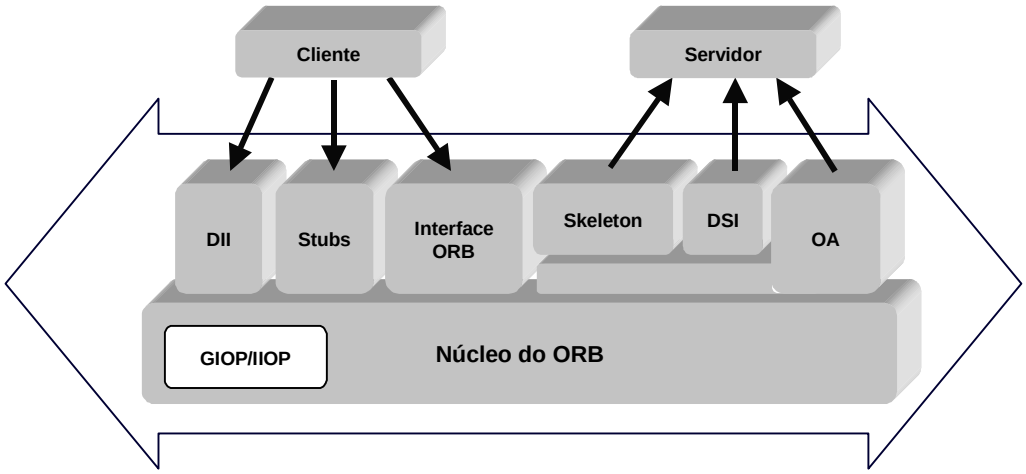


Figura 15: arquitetura CORBA.

Stubs e *skeletons* são os intermediários dessa comunicação. De um lado, eles fornecem aos objetos a abstração de que estão interagindo com objetos locais. Do outro lado, eles transformam as interações dos objetos em mensagens a serem transmitidas através do núcleo do ORB. A maior limitação da arquitetura CORBA, no que se refere a comunicação multimídia, é que ela, ao contrário do RM-ODP, não dá suporte a tipos de interação por mensagens de fluxo contínuo, nem à especificação, operação e manutenção da QoS requerida pelos objetos.

Dois exemplos bem interessantes de modelos computacionais OO, que tentam suprir, ao mesmo tempo, as deficiências do RM-ODP e da arquitetura CORBA, no que se refere à provisão de QoS em ambientes distribuídos, são apresentados nas subseções a seguir.

3.5.1 Modelo XRM

Dentre outros projetos, o grupo COMET da Universidade de Columbia tem desenvolvido o *Extended Integrated Reference Model* (*XRM*), um framework que modela as funções de gerência e controle de *redes multimídia*, abrangendo, segundo a definição dada em [LAZAR94], recursos presentes nas estações finais (processadores, memória, discos, dispositivos multimídia, etc.) e na rede de distribuição (comutadores, enlaces, etc.). Uma representação abstrata do XRM é mostrada na Figura 16. Dando suporte à estrutura do XRM está a pressuposição de que os fundamentos de operação de todos esses recursos são os mesmos. Segundo o XRM, todos esses recursos podem ser modelados como produtores, consumidores ou processadores de dados.

Como mostra a figura, o XRM é organizado em cinco planos distintos (que, assim como na arquitetura QoS-A, refletem os princípios da separação e da gerência assíncrona, apresentados na Seção 2.5):

- *Plano N*: cobre a área de gerência de redes segundo o RM-OSI [ISO90] (gerência de falhas, contabilidade, configuração, desempenho e segurança).

- *Plano M*: responsável pelo controle de recursos, basicamente por intermédio de mecanismos de escalonamento e de controle de admissão.

- *Plano C*: responsável pela gerência de fluxos, através de mecanismos de negociação e renegociação, e de protocolos de configuração de fluxos.

- *Plano U*: modela os protocolos de transmissão de dados dos usuários.

- *Plano D*: representa a *telebase*, uma base que contém informações sobre toda a rede multimídia, incluindo recursos alocados, estatísticas coletadas, detecções de falhas, violações de contrato, etc.

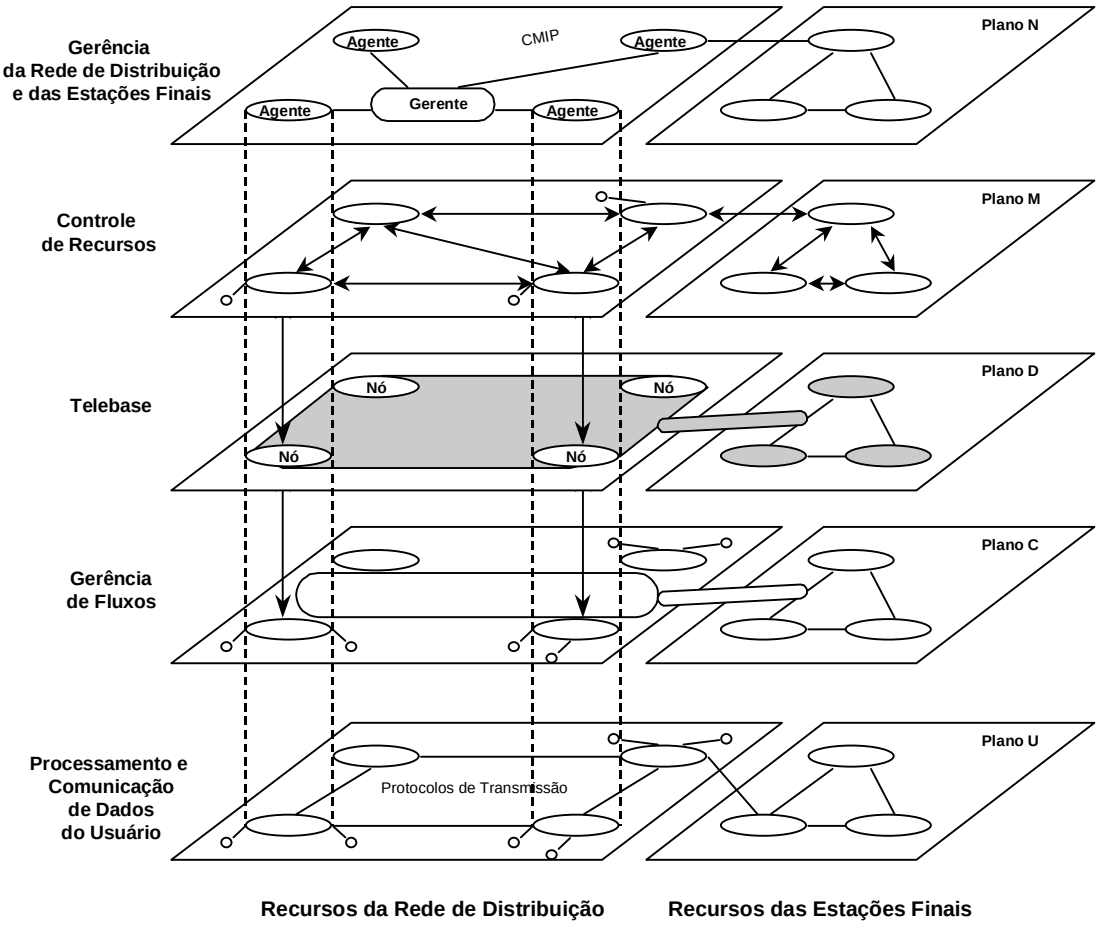


Figura 16: Modelo XRM.

Uma aplicação do XRM tem sido realizada como parte da arquitetura *Binding* [LAZAR95], projeto também desenvolvido pelo grupo COMET. Esta arquitetura adota a abordagem de serviços configuráveis, através da definição de interfaces, métodos e primitivas que permitem a conexão dinâmica entre recursos das estações finais e da rede de distribuição. As interfaces, armazenadas na telebase, abstraem as funcionalidades dos recursos, enquanto os métodos e primitivas permitem que os algoritmos residentes nos planos M e C controlem esses recursos. Estes algoritmos invocam os métodos e primitivas oferecidos pelas interfaces definidas na telebase através de uma plataforma para aplicações distribuídas, como por exemplo a definida pela arquitetura CORBA.

Através dessas interfaces, a arquitetura Binding oferece aos usuários (projetistas) um ambiente de software sobre o qual eles podem implementar, por exemplo, protocolos de configuração de fluxos e mecanismos de escalonamento e de controle de admissão (dentre outros mecanismos pertinentes aos planos M e C) com

características de QoS específicas. Esta arquitetura facilita consideravelmente a implementação de novos serviços, uma vez que eles não precisam mais ser, necessariamente, internos aos sistemas operacionais e de comunicação, podendo ser vistos, de certa forma, como componentes das aplicações dos usuários.

3.5.2 Modelo de Referência Unificado

Dentre outros projetos, o Laboratório TeleMídia da Pontifícia Universidade Católica do Rio de Janeiro tem trabalhado, nos últimos anos, no projeto RAVel, que envolve, entre outras atividades, a definição de um *Modelo de Referência Unificado* [COLCHER98]. Este modelo permite a construção e configuração de ambientes distribuídos flexíveis, no que diz respeito a adequação dos serviços de comunicação à diversidade de modelos de caracterização de carga e de especificação da QoS impostos pelos usuários. Boa parte da terminologia apresentada no Capítulo 2 baseia-se neste modelo.

Atualmente, a definição do Modelo de Referência Unificado concentra-se em duas visões inspiradas nas duas visões do RM-ODP, apresentadas na Seção 3.5.

Visão Computacional

A visão computacional oferece uma descrição abstrata de alto nível, válida em quaisquer níveis de abstração e escalas de distribuição. Segundo esta visão, um ambiente de processamento e comunicação qualquer é composto de uma *máquina de objetos* e vários *componentes*. Uma máquina de objetos fornece um modelo de execução completo para cada componente, agindo como a controladora do ambiente de processamento e comunicação. A Figura 17 ilustra alguns dos conceitos definidos pela visão computacional.

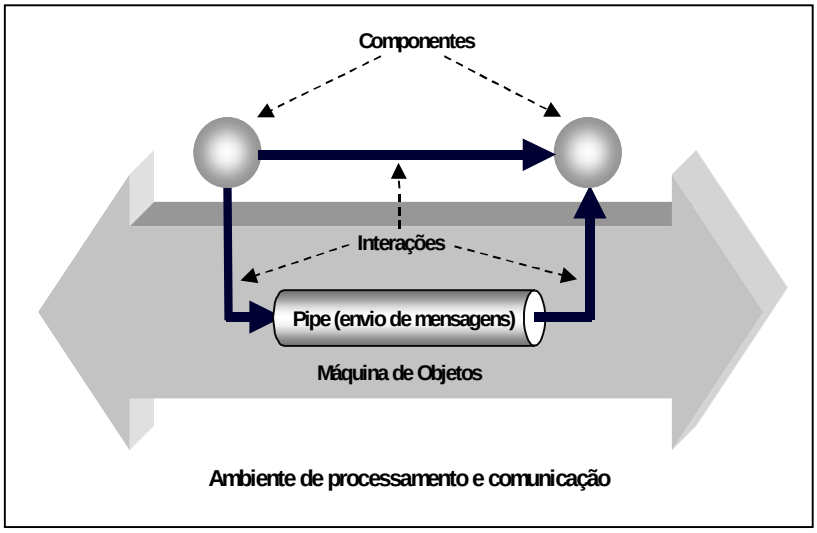


Figura 17: visão computacional do Modelo de Referência Unificado.

Máquinas de objetos específicas podem ser definidas sob condições restritas com relação aos tipos de componentes envolvidos e aos padrões de cooperação entre eles, descrevendo assim ambientes específicos. Há, no Modelo de Referência Unificado, dois padrões básicos de cooperação entre componentes, que definem dois tipos de serviços básicos oferecidos pelas máquinas de objetos. O *serviço de interação* entre componentes corresponde às tradicionais invocações de métodos. Já o *serviço de envio explícito de mensagens* entre componentes utiliza componentes internos da máquina de objetos denominados *pipes* (conforme ilustrado na Figura 17), com o qual os componentes interagem (através do serviço de interação) de forma a solicitar a transferência de mensagens. De forma simplificada, pipes oferecem operações de *read* e *write*, que permitem a troca de mensagens entre componentes. Opcionalmente, pipes podem avisar os componentes da presença de mensagens (*pipes ativos*), funcionando de forma semelhante a tratadores de eventos. Pipes podem também ser associados a especificações e mecanismos de provisão de QoS, sendo denominados, neste caso, de *MediaPipes*.

No caso do serviço explícito de envio de mensagens, não existe a interação direta entre os componentes usuários do serviço, e sim entre estes componentes e a máquina de objetos. Dependendo da escala de distribuição dos

componentes, interações tornam-se, eventualmente[23], solicitações explícitas de envio de mensagens dentro da máquina de objetos. A conversão de invocações de métodos em mensagens, e vice-versa, é realizada implicitamente por componentes também internos à máquina, denominados, assim como na arquitetura CORBA, de stubs e skeletons, como ilustra a Figura 18.

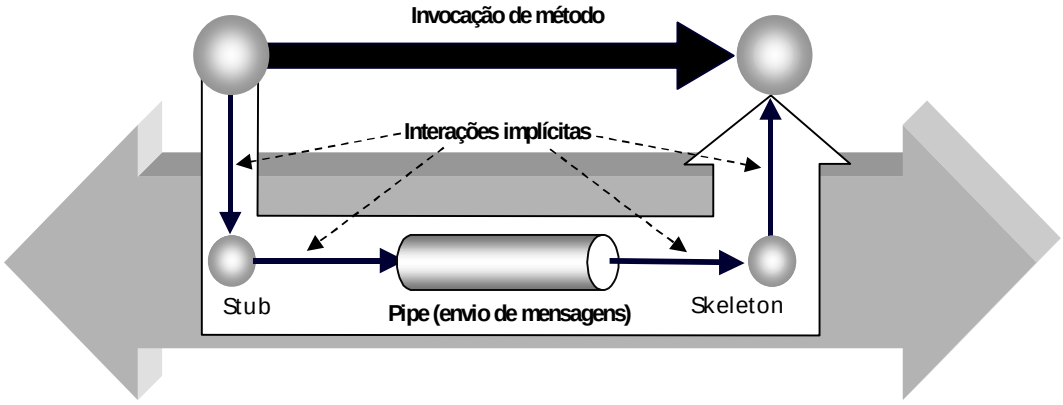


Figura 18: relacionamento entre interações e envios explícitos de mensagens.

Note que interações entre componentes de uma máquina de objetos podem ser implicitamente definidas através de trocas de mensagens que, por sua vez, correspondem, novamente, a interações, só que dessa vez entre os componentes e uma outra máquina de objetos. O relacionamento entre diferentes máquinas de objetos em diferentes níveis de abstração e escalas de distribuição de componentes é assunto da visão de engenharia.

Visão de Engenharia

Segundo o Modelo de Referência Unificado, o conceito de uma máquina de objetos como elemento central de um ambiente de processamento e comunicação é independente do nível de abstração e da escala de distribuição dos componentes. O mesmo modelo pode ser utilizado em diferentes situações como, por exemplo, no tratamento de componentes envolvidos na comunicação local entre entidades de protocolo de camadas adjacentes (comunicação vertical), ou mesmo na modelagem da comunicação entre entidades de protocolo de uma mesma camada (comunicação horizontal), ou ainda na modelagem da interação entre componentes de aplicações, de

[23] Quando a máquina de objetos é primitiva, como por exemplo, no caso do ambiente de execução definido internamente a um processo, chamadas de operação são mecanismos primitivos e não tratados pelo Modelo de Referência Unificado.

maneira transparente em relação à distribuição destes componentes. A visão de engenharia do Modelo de Referência Unificado corresponde a um refinamento da visão de engenharia do RM-ODP, através da definição de patterns de composição e configuração entre componentes nas diversas escalas de distribuição. O conceito de *pattern* é introduzido na Seção 3.6.

Para citar um exemplo de utilização de patterns de composição, o Modelo de Referência Unificado define uma *camada* como sendo um ambiente de processamento e comunicação cujos componentes são usuários de uma mesma camada (ou subcamada) de protocolo. Para que a comunicação entre componentes usuários de uma camada N (N > 1) seja possível, no Modelo de Referência Unificado, pipes e interações devem ser utilizados internamente à camada.

Como mostra a Figura 19, os stubs e skeletons internos à camada N correspondem às entidades de protocolo da mesma, sendo, portanto, usuários de uma mesma camada N-1, que dá suporte à camada N. Estas entidades se comunicam através de pipes e interações internos à camada N-1, e assim sucessivamente, definindo desta forma o *pattern de composição de camadas*. Este pattern de composição envolve também a comunicação entre os componentes usuários de uma camada e os stubs e skeletons dessa camada, que se dá através de máquinas de objetos controladoras dos ambientes locais de execução.

Através da utilização de MediaPipes em todos os níveis de abstração e escalas de distribuição envolvidos em um ambiente distribuído, o Modelo de Referência Unificado oferece uma capacidade de provisão de QoS verdadeiramente fim-a-fim, pois tem-se a garantia de controle da comunicação vertical, assegurada basicamente pelos mecanismos de alocação e escalonamento definidos nos ambientes locais de execução, e da comunicação horizontal, mantida através da implementação de protocolos de configuração e manutenção de fluxos (pipes) e de transmissão.

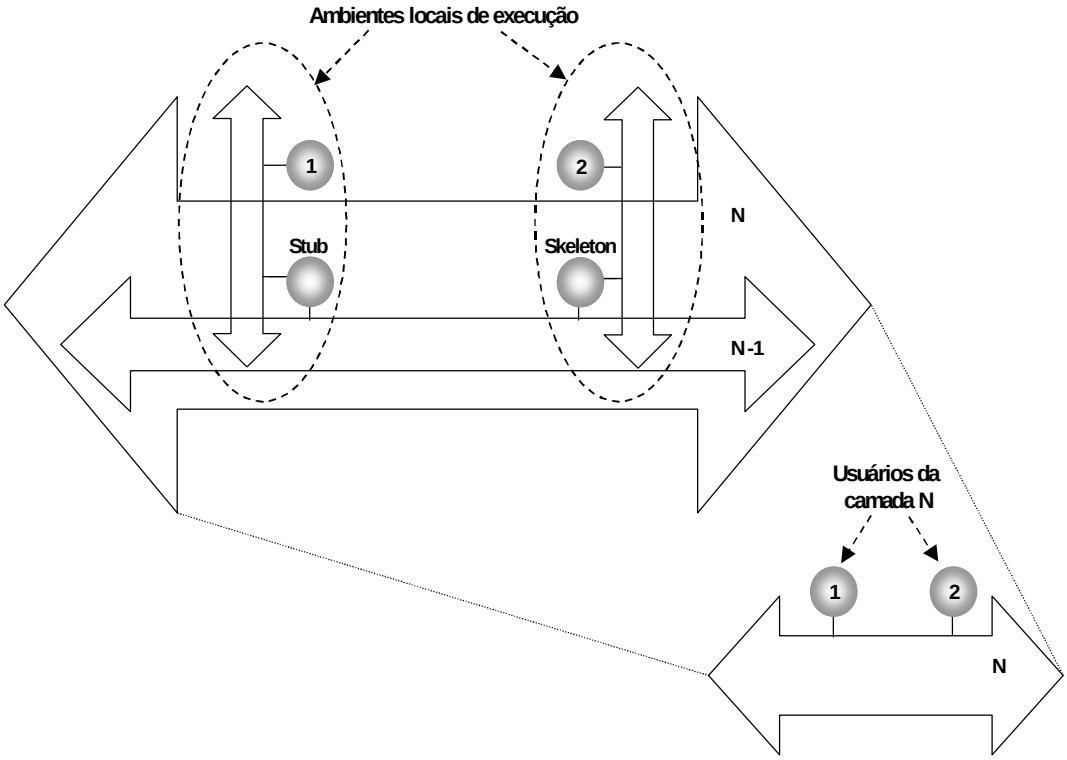


Figura 19: pattern de composição de camadas.

3.5.3 Considerações a Respeito dos Modelos de Objetos com QoS

Modelos de objetos que permitem a especificação de serviços configuráveis com relação à QoS apresentam a mais promissora das abordagens de tratamento de QoS, graças a sua flexibilidade e capacidade de configuração e interoperação, dentre outras vantagens. Existem vários outros modelos que utilizam esta mesma abordagem, podendo ser classificados basicamente em duas categorias, segundo a notação introduzida em [COULSON97]:

- *Modelos não nivelados*: modelos que oferecem aos projetistas de sistemas a capacidade de conectar "*objetos multimídia padrão*" (basicamente, dispositivos multimídia encapsulados por objetos), e de monitorar e controlar fluxos de dados de mídias contínuas que atravessam internamente esses objetos. Nestes modelos, não é permitido o acesso aos aspectos internos desses objetos; eles são oferecidos aos projetistas de sistemas através de bibliotecas. Dentre outros modelos, insere-se nessa categoria o modelo XRM, o modelo *SUMO* [COULSON95a], desenvolvido junto à Universidade de Lancaster, e o

Multimedia Systems Services (*MSS*) [ISO96], que vem sendo padronizado junto a ISO.

- *Modelos nivelados*: modelos que integram tipos de dados de mídias contínuas como tipos de primeira ordem (assim como inteiros, cadeias de caracteres, etc.), como por exemplo, o Modelo de Referência Unificado e a plataforma multimídia definida em [COULSON97].

A desvantagem dos modelos não nivelados com relação aos modelos nivelados decorre de uma certa falta de flexibilidade dos primeiros. Se os projetistas precisarem implementar processamentos específicos em dados de mídias contínuas, eles serão obrigados a atuar por fora do modelo, isto é, agindo diretamente sobre os sistemas operacionais e os protocolos de comunicação, o que é indesejável, visto que isto complica a tarefa desses projetistas. Ao permitirem a definição de interfaces de objetos orientadas a eventos (como as viabilizadas pelos pipes ativos no Modelo de Referência Unificado), os modelos nivelados possibilitam aos projetistas de sistemas manipular diretamente os fluxos de dados de mídias contínuas, de maneira similar aos tipos de dados convencionais.

Independente da forma como os modelos de objetos com QoS são classificados, há uma lacuna não preenchida nos mesmos, que concerne à implementação dos mecanismos de provisão de QoS. Por tentarem ser bastante genéricos, estes modelos acabam não oferecendo aos projetistas de sistemas certas facilidades com relação à implementação desses mecanismos, o que torna a tarefa destes projetistas bastante árdua. O presente trabalho objetiva o preenchimento desta lacuna, no contexto do Modelo de Referência Unificado, através de estruturas que permitam o projeto, implementação e configuração desses mecanismos de provisão de QoS em todos os níveis de abstração e escalas de distribuição envolvidos em um ambiente de processamento e comunicação qualquer.

3.6 Patterns e Frameworks em Sistemas Multimídia

Durante o desenvolvimento de quaisquer sistemas, projetistas tendem a se valer de decisões de projeto tomadas anteriormente em outros sistemas. *Patterns* [PREE95] [GAMMA95] [BUSCHMANN95] podem ser vistos como uma forma de documentar conjuntos de regras que descrevem decisões de projeto recorrentes em vários sistemas, facilitando assim a reutilização de soluções já existentes. Patterns podem cobrir vários níveis de descrição presentes em um sistema, dependendo de quão genéricas são as descrições desses patterns, podendo variar desde diretrizes a respeito da organização de um sistema até práticas de codificação da implementação desse sistema.

Os conceitos oferecidos pela tecnologia OO também têm o potencial de aumentar significativamente a reutilização de soluções existentes, se comparada a outras abordagens de construção de sistemas. Esses conceitos são especialmente úteis na descrição de frameworks, definidos em [PREE95] como arquiteturas semi-prontas reutilizáveis. Frameworks permitem que não só os componentes de um sistema, mas também as decisões de projeto a ele associadas, sejam reutilizados. Neste sentido, o potencial da abordagem de patterns permite a descrição de frameworks de forma mais abstrata do que o código correspondente que os implementaria. Patterns podem ser vistos como uma forma de documentação de alto nível para frameworks, por se constituírem em "mapas" muito úteis para a compreensão dos detalhes de implementação destes.

Em geral, frameworks padronizam sistemas para um domínio específico. Usualmente, vários aspectos de um framework relativos a esse domínio não podem ser antecipados. Essas "partes" do framework, denominadas em [PREE95] de *pontos de flexibilização* (*hot spots*), devem ser suficiente genéricas para que possam ser adaptadas a cada caso de uso. Patterns mostram-se como ferramentas poderosas no delineamento desses pontos de flexibilização durante o processo de definição de um framework.

Além disso, é interessante que certas diretrizes [MEYER88] relacionadas ao uso da tecnologia OO sejam levadas em conta durante o processo de definição de um framework, como os princípios do acoplamento fraco entre objetos e forte internamente a eles, das interfaces não redundantes, etc. O uso de patterns permite a definição de

estruturas que induzem o seguimento dessas diretrizes durante a definição de um framework.

Especificamente na área de ambientes multimídia distribuídos, vários são os trabalhos que utilizam a abordagem de patterns, sob óticas diferentes. Trabalhos como o *Conduits+* [HUNI95], o *x-Kernel* [HUTCHISON91], e o Modelo de Referência Unificado, por exemplo, adotam patterns arquiteturais ao estilo *Pipes e Filtros* [BUSCHMANN95]. O Conduits+ e o x-Kernel modelam núcleos (*kernels*) de sistemas operacionais, fornecendo uma arquitetura para a construção de protocolos de comunicação eficientes. Dentre suas principais características, estão o tratamento da sincronização de acesso à memória entre camadas, e a passagem de mensagens minimizando o número de cópias de memória, segundo rege o princípio do desempenho apresentado na Seção 2.5. *Patterns arquiteturais* são considerados como definições abstratas para a organização estrutural de sistemas. Eles não são necessariamente apresentados sob a forma de classes ou interfaces de objetos relacionados a um problema, mas sim provêem um conjunto de regras que devem ser aplicadas durante o projeto de um sistema.

Outros trabalhos, dentre os quais o *Adaptive Communication Environment* (*ACE*) [SCHMIDT97], desenvolvido na Universidade de Washington, utilizam patterns mais específicos que os patterns arquiteturais. O ACE objetiva simplificar os mecanismos de comunicação entre processos (IPC), além de automatizar a configuração e reconfiguração do software de comunicação. Para isso, o ACE implementa um conjunto de patterns de projeto, dentre eles o *Strategy*, o *Bridge*, o *Abstract Factory* e o *Adapter*, todos definidos em [GAMMA95]. *Patterns de projeto* (*design patterns*) são, em geral, mais específicos que os patterns arquiteturais na medida em que eles apresentam um conjunto de objetos ou classes de objetos que podem ser usados diretamente na solução de um problema particular. Contudo, a definição desses patterns é suficiente genérica para permitir também a modelagem de regras de estruturação de sistemas. O presente trabalho utiliza vários dos patterns de projeto apresentados em [GAMMA95] como solução para a modelagem das estruturas e mecanismos formalizados no Capítulo 2.

3.7 Sumário

Neste capítulo, foram apresentados trabalhos relacionados à questão do suporte à provisão de QoS em vários ambientes, distintos quanto à escala de distribuição e ao nível de abstração de seus componentes. Primeiramente, foram focalizados trabalhos que tratam esta questão sob o ponto de vista de subsistemas isolados, mais especificamente, de sistemas operacionais, redes de distribuição e protocolos de comunicação. Também foram apresentadas algumas arquiteturas e modelos que objetivam a uniformização do tratamento da QoS entre estes subsistemas, e que permitam aos usuários do sistema multimídia como um todo requisitar serviços através de interfaces condizentes com as suas visões de QoS. As abordagens sugeridas por esses trabalhos serviram como base para um maior detalhamento do problema da provisão de QoS em sistemas multimídia, levantado de modo genérico no Capítulo 2. A partir deste detalhamento, e com a aplicação do conceito de patterns também introduzido neste capítulo, serão apresentados, no Capítulo 4, os elementos do Framework para Provisão de QoS.

Capítulo 4

Descrição do Framework

A idéia de se definir o Framework para Provisão de QoS nasceu da percepção de que as estruturas e mecanismos formalizados no Capítulo 2 são recorrentes nas várias escalas de distribuição e níveis de abstração presentes em um ambiente de processamento e comunicação qualquer, e independentes da diversidade de serviços passíveis de serem oferecidos pelo fornecedor de serviços. Este capítulo sugere a utilização da abordagem de patterns para retratar ambas as características de recorrência e independência existentes no framework a ser apresentado e ilustrado nesse capítulo. Esta abordagem torna mais fácil para um projetista de sistemas adaptar o framework às suas necessidades específicas, visto que:

- Oculta detalhes relativos à distribuição dos componentes de um sistema;
- Mostra da maneira mais clara possível como tornar flexível um sistema, com relação ao fornecimento de serviços, e
- Permite também a construção de partes isoladas de um sistema associadas à provisão de QoS, através da aplicação de elementos específicos do Framework para Provisão de QoS. Essas partes podem, eventualmente, se inserir em sistemas já existentes e não necessariamente modelados segundo o framework.

4.1 Elementos do Framework

O Framework para Provisão de QoS é composto pelos seguintes elementos:

- Um Framework para Parametrização de Serviços;
- Frameworks para Compartilhamento de Recursos, que incluem:
  - Um Framework para Escalonamento de Recursos e
  - Um Framework para Alocação de Recursos;
- Frameworks para Orquestração de Recursos, que incluem:
  - Um Framework para Negociação de QoS e
  - Um Framework para Sintonização de QoS.

A divisão do Framework para Provisão de QoS em um conjunto de frameworks menores advém, em primeiro lugar, da diferença entre os níveis de descrição providos pelo Framework para Parametrização de Serviços com relação aos outros frameworks. O Framework para Parametrização fornece um modo de se estruturar dados adequadamente para a representação de parâmetros, podendo ser *diretamente aplicado* através de qualquer linguagem de programação, de especificação ou de definição de dados. O resultado da aplicação deste framework em um sistema é, em última análise, a definição de uma *base de informações de QoS* a qual os mecanismos de provisão de QoS podem ter acesso. Considerações a respeito da distribuição dessa base pelo sistema e de como os mecanismos têm acesso à mesma não são tratadas pelo framework.

Por outro lado, os Frameworks para Compartilhamento e Orquestração de Recursos funcionam como meta-arquiteturas, uma vez que eles *esboçam* os principais mecanismos de provisão de QoS (e os pontos de flexibilização associados) a serem modelados durante o processo de definição de arquiteturas de gerência de recursos para as diferentes escalas de distribuição e níveis de abstração desejados. Estes frameworks não são diretamente aplicáveis: eles documentam, através de classes de objetos e

relacionamentos, conjuntos de regras de estruturação dos mecanismos de provisão de QoS que podem ser usadas na modelagem dessas arquiteturas. Estas classes de objetos são componentes essencialmente conceituais. Para poderem se tornar classes de objetos propriamente ditas, os frameworks aos quais elas pertencem devem ser aplicados no contexto de uma arquitetura específica. Os Frameworks para Compartilhamento e Orquestração de Recursos pressupõem que bases de informações de QoS (modeladas ou não pelo Framework para Parametrização de Serviços) estejam disponíveis para as arquiteturas modeladas a partir desses frameworks, a fim de que as políticas de provisão de QoS possam ser corretamente implementadas.

Com relação à divisão interna dos Frameworks para Compartilhamento e Orquestração de Recursos, objetiva-se, principalmente, separar funcionalidades distintas existentes em sistemas que provêm QoS, a fim de permitir a construção de partes isoladas destes sistemas. Embora estes frameworks estejam bastante integrados, é interessante que eles possuam também uma certa independência entre si, o que possibilita a reutilização de partes já implementadas de sistemas preexistentes, alcançando assim um alto grau de acoplamento com estes sistemas.

4.2 Representação do Framework

Todos os elementos do Framework para Provisão de QoS são descritos por meio de uma notação textual informal, auxiliada de uma notação gráfica com informações relacionadas a questões de implementação. Os diagramas que descrevem graficamente o framework apoiam-se na *Linguagem Unificada de Modelagem OO* (*Unified Modeling Language – UML*) [UML97], apresentada no Apêndice A. Particularmente neste trabalho, foi adotada uma notação diferenciada entre as classes de objetos que compõem a estrutura básica dos elementos do Framework para Provisão de QoS (hachuradas) e as que simbolizam possíveis instanciações dos pontos de flexibilização desses elementos (brancas). Durante a descrição textual, foi utilizada também uma grafia diferenciada para denominar classes e métodos abstratos (em *`itálico`*) e classes e métodos concretos, além de atributos e relacionamentos. Já as dicas de implementação incluem pedaços de código escritos em uma pseudo-notação derivada da linguagem Java. Definiu-se também uma extensão à notação UML especificamente para a apresentação dos exemplos de aplicação dos Frameworks para

Compartilhamento e Orquestração de Recursos. Os nomes das classes e métodos definidos nesses exemplos são ligados aos das classes e métodos que compõem os frameworks através de um descritor envolvido por "<<" e ">>", como ilustra o exemplo da Figura 20.

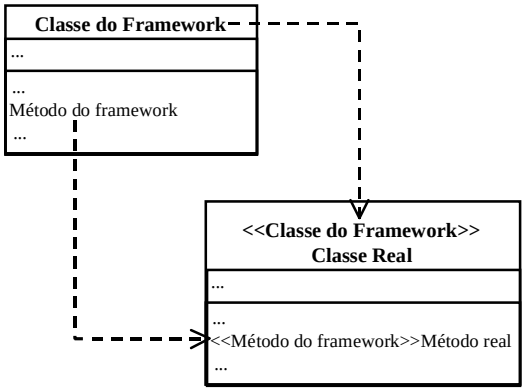


Figura 20: ligação entre uma classe de um framework e uma classe real.

Nesta dissertação, a descrição do Framework para Provisão de QoS leva em consideração somente relacionamentos de dependência, especialização, agrupamento e associação entre componentes associados à provisão de QoS em ambientes genéricos de processamento e comunicação. Outros tipos de relacionamentos, como os de concorrência e sincronização, embora importantes em uma descrição mais precisa, não serão abordados.

4.3 Framework para Parametrização de Serviços

Todos os mecanismos de provisão de QoS presentes em um fornecedor são modelados através do uso de parâmetros de caracterização de serviços. Para que seja possível tornar suficientemente flexível a parametrização de serviços em um fornecedor qualquer, há a necessidade de uma estrutura, correspondente ao núcleo do Framework para Provisão de QoS, responsável por definir um esquema de parametrização que seja independente dos possíveis serviços a serem oferecidos pelo fornecedor.

Parâmetros de caracterização de serviços são a imagem do comportamento de um fornecedor. Por causa da diversidade de serviços e níveis de visão de QoS considerados, não se pode definir um conjunto finito de parâmetros concretos que representem todos eles. Como exemplo, dependendo de onde o parâmetro

retardo de trânsito esteja sendo utilizado, PIs podem ser interpretados como system calls de um sistema operacional ou SAPs de um sistema de comunicação. Mais ainda, em um mesmo sistema de comunicação, esse parâmetro pode ser definido de maneiras diferentes em camadas distintas, se considerarmos a diferença semântica entre os SAPs dessas camadas.

Em casos semelhantes ao exemplo anterior, é importante haver uma definição abstrata para o parâmetro, de maneira que ele possa ser aplicado em todas as circunstâncias particulares através de especializações. Um dos propósitos do Framework de Parametrização de Serviços é esquematizar uma estruturação que permita essa definição abstrata de parâmetros.

O framework utiliza a noção de grandeza como fundamento para a definição de parâmetros abstratos. Através desse framework podemos definir, como exemplificaremos mais à frente, o parâmetro-base *retardo (entre dois eventos)*, que representa abstratamente a grandeza retardo. A partir deste parâmetro podem ser derivados outros parâmetros, dentre os quais um que represente a grandeza retardo de trânsito. Porém, este também é um parâmetro abstrato, logo várias outras especializações podem ser necessárias (criando uma *hierarquia de derivação de parâmetros*), de forma a se conseguir um parâmetro concreto.

Em uma implementação real de um fornecedor de serviços, um parâmetro deve ser sempre concreto, de forma a tornar claro o significado de seu valor em relação aos mecanismos de provisão de QoS. Como exemplo, os vários mecanismos de negociação da QoS presentes em um sistema de comunicação podem ser definidos de forma única, através de sua associação a um conjunto de parâmetros abstratos (incluindo o retardo de trânsito, por exemplo), sendo posteriormente configurado por um projetista de protocolos para diferentes camadas do sistema, conjuntamente com a definição de parâmetros concretos apropriados a partir do conjunto de parâmetros abstratos.

4.3.1 Qualificadores de Parâmetros

O Framework de Parametrização de Serviços se vale do conceito de qualificador para estruturar parâmetros de maneira independente dos possíveis serviços a serem oferecidos pelo fornecedor. Como exemplo, podemos definir, através de

qualificadores de evento, o parâmetro retardo como o tempo decorrido $\Delta T = T_2 - T_1$ entre dois eventos gerais E_1 e E_2, que ocorrem nos tempos T_1 e T_2, respectivamente. A partir daí, podemos derivar o parâmetro retardo de trânsito através:

- Da descrição dos eventos E_1 e E_2 como sendo "*a passagem do primeiro item de informação*" e "*a passagem do último item de informação*", respectivamente, e
- Da utilização de qualificadores de localização para os PIs P_1 e P_2 onde ocorrem os eventos E_1 e E_2, respectivamente.

Para definirmos parâmetros concretos a partir do retardo de trânsito em uma camada de um sistema de comunicação, por exemplo, devemos primeiramente descrever P_1 e P_2 como um par de SAPs da camada. Posteriormente, através de qualificadores de valor, podemos definir retardo de trânsito (instantâneo), retardo de trânsito médio, variação estatística do retardo de trânsito, etc.

4.3.2 Categorias de Serviço

Se, por um lado, as hierarquias de derivação de parâmetros tornam a parametrização de serviços bastante flexível, por outro lado, deixar a especialização adequada de parâmetros totalmente a cargo do projetista, em todos os níveis de visão de QoS presentes em um fornecedor, torna o Framework de Parametrização de Serviços pouco prático. Isto é especialmente percebido quando são implementadas as políticas de provisão de QoS, que são as maiores usuárias dos parâmetros definidos pelo projetista.

Uma vez que as categorias de serviço providas por um fornecedor determinam que políticas de provisão de QoS podem ser usadas, outra estruturação que se torna necessária no Framework para Parametrização de Serviços refere-se às categorias de serviço. Segundo esta estruturação, toda manipulação de parâmetros feita pelos mecanismos de provisão de QoS deve ocorrer no contexto da categoria desejada (que funciona como uma espécie de interface para a base de informações de QoS do fornecedor), com exceção para os mecanismos de monitorização e mapeamento, que com freqüência atuam diretamente sobre a estrutura interna de parâmetros de diferentes níveis de visão de QoS simultaneamente, como veremos adiante.

A descrição de uma categoria de serviço como um simples conjunto de parâmetros relacionados ao tipo de ambiente, como foi proposto no Capítulo 2, é muito vaga. Primeiro, pela própria diversidade de níveis de abstração relacionados ao conceito de ambiente, o que demanda a derivação apropriada de inúmeros parâmetros em cada nível de visão de QoS. E, em segundo lugar, pelo fato das características e requisitos de processamento e comunicação exigidos pelos dados dos usuários poderem variar consideravelmente em categorias genéricas de serviço como as já exemplificadas, principalmente em função dos tipos desses dados.

No caso da categoria dos serviços multimídia, por exemplo, os parâmetros mais importantes são aqueles cujas dimensões de QoS envolvem quantidade de informação, tempo, coerência e integridade, como vazão, retardo, sincronização e perda. Entretanto, dependendo do nível de visão de QoS em questão e do tipo de dado que um usuário queira transmitir, receber ou processar, diferentes derivações dos parâmetros acima podem ser necessárias. E o uso de diferentes derivações de parâmetros deve ser levado em conta ao implementarmos as políticas de provisão de QoS apropriadas.

É interessante então sermos capazes de refinar a taxonomia de um serviço, utilizando para isso propriedades relacionadas ao nível de visão de QoS do usuário e aos tipos de dados provenientes dele. Mais ainda, tais propriedades devem ser descritas por intermédio de derivações apropriadas de parâmetros. Assim, definimos *categorias de serviço*, de forma mais precisa, como conjuntos de parâmetros que possuem determinadas propriedades em comum, relacionadas ao tipo do ambiente, ao nível de visão de QoS e aos tipos de dados do usuário.

Como exemplo, podemos definir, no nível de visão de QoS das aplicações multimídia distribuídas, a categoria de serviço de vídeo refinada da categoria dos serviços multimídia. A propriedade principal da mídia vídeo, que caracteriza a sua categoria de serviço, é a necessidade de reprodução dos dados para o usuário a uma taxa constante. Derivações do parâmetro *retardo*, como *retardo de trânsito máximo* e *variação estatística do retardo de trânsito*, são usadas para descrever essa categoria de serviço.

Pelo processo acima, podemos continuar sucessivamente refinando categorias de serviço, criando assim *hierarquias de categorias de serviço*, se considerarmos, por exemplo, as diversas técnicas existentes de processamento de vídeo.

Assim, podemos refinar da categoria de serviço de vídeo uma subcategoria de serviço de vídeo sem compressão e compactação e outra de serviço de vídeo com compressão e compactação, por exemplo. Uma importante propriedade que difere as duas categorias relaciona-se à natureza da carga gerada pelos dados. Assim, derivações do parâmetro *vazão* podem ser usadas para descrever essas subcategorias. Para vídeos sem compressão e compactação, o parâmetro *vazão máxima* é suficiente; enquanto que para vídeos com compressão e compactação, os parâmetros *vazão média* e *máxima variação da vazão*, entre outros, são os mais utilizados.

4.3.3 Componentes do Framework para Parametrização de Serviços

A Figura 21 mostra as classes e os respectivos relacionamentos que constituem o Framework para Parametrização de Serviços. As classes *ServiceCategory* e *Parameter* simbolizam abstratamente as hierarquias de categorias de serviço e de derivação de parâmetros, respectivamente. O pattern estrutural Bridge [GAMMA95] é utilizado para representar as categorias de serviço como conjuntos de parâmetros (através do relacionamento de agrupamento `parameterList`), possibilitando o desacoplamento entre as hierarquias de categorias de serviço e de derivação de parâmetros, tornando-as independentemente extensíveis. O desacoplamento é conseguido através do método `getParameter()` em *ServiceCategory*, que retorna uma instância de qualquer subclasse de *Parameter* com base no atributo `param_descriptor`.

A classe *Qualifier* modela de maneira abstrata os qualificadores de parâmetros. O pattern estrutural Bridge também é utilizado para representar parâmetros como conjunto de qualificadores (através do relacionamento de agrupamento `qualifierList`), o que oculta dos mecanismos de provisão de QoS certos detalhes a respeito da estruturação dos parâmetros, tornando tais mecanismos mais independentes desses parâmetros. Isto é conseguido através do método `getQualifier()` em *Parameter*, que retorna uma instância de uma subclasse de *Qualifier* com base no atributo `qual_descriptor`.

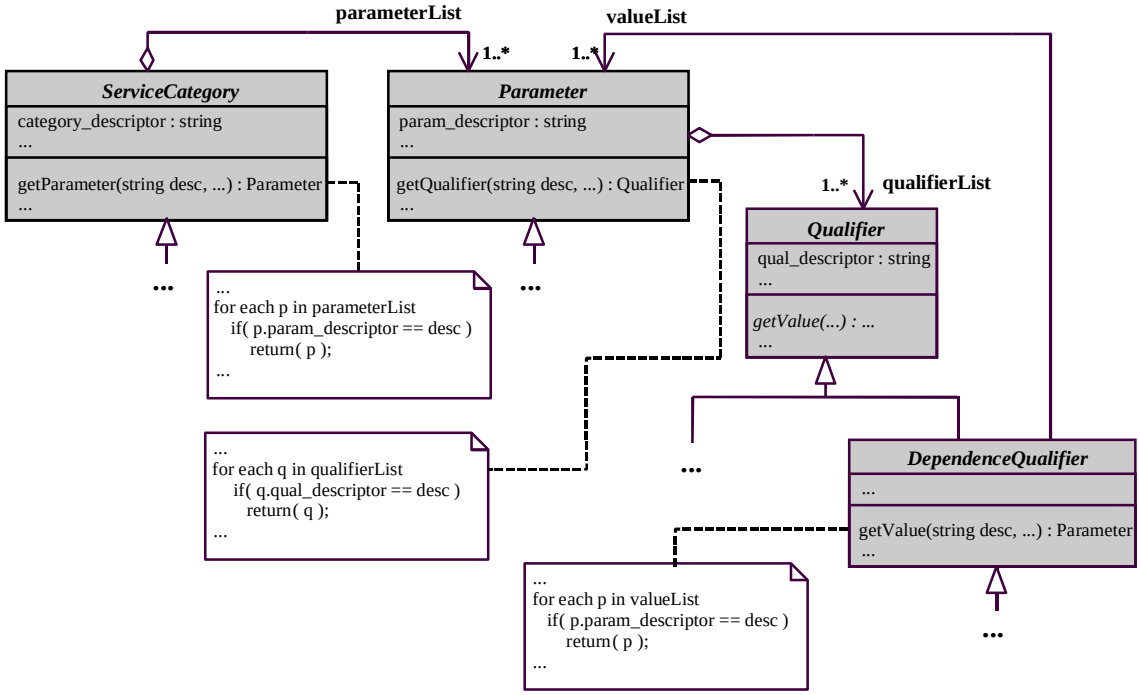


Figura 21: Framework para Parametrização de Serviços.

4.3.4 Exemplo de Aplicação do Framework para Parametrização

O desenho do framework possibilita diferentes formas de estruturação. A Figura 22 apresenta um exemplo simplificado de instanciação do framework, onde todas as subclasses de *`Parameter`* representam parâmetros abstratos, e cada dupla de métodos {`set*Value()`, `get*Value()`}[24] (simbolizada por `set(get)*Value()`) das subclasses de *`Parameter`* implementa um parâmetro concreto derivado do parâmetro abstrato representado pela sua classe. Subclasses de *`Qualifier`* são utilizadas para desempenhar o papel dos qualificadores que armazenam os valores aos quais as duplas de métodos `set(get)*Value()` têm acesso. A classe `ThroughputVariance`, por exemplo, representa o parâmetro abstrato variação estatística da vazão. As duplas de métodos `set(get)CurrValue()` e `set(get)UboundValue()` nela presentes representam os parâmetros variação estatística (instantânea) da vazão e máxima variação da vazão, respectivamente. As classes `CurrentValueQualifier` e `UpperBoundValueQualifier` representam os qualificadores associados a essas duas duplas de métodos.

[24] Na figura, foram omitidos os métodos `set*Value()`.

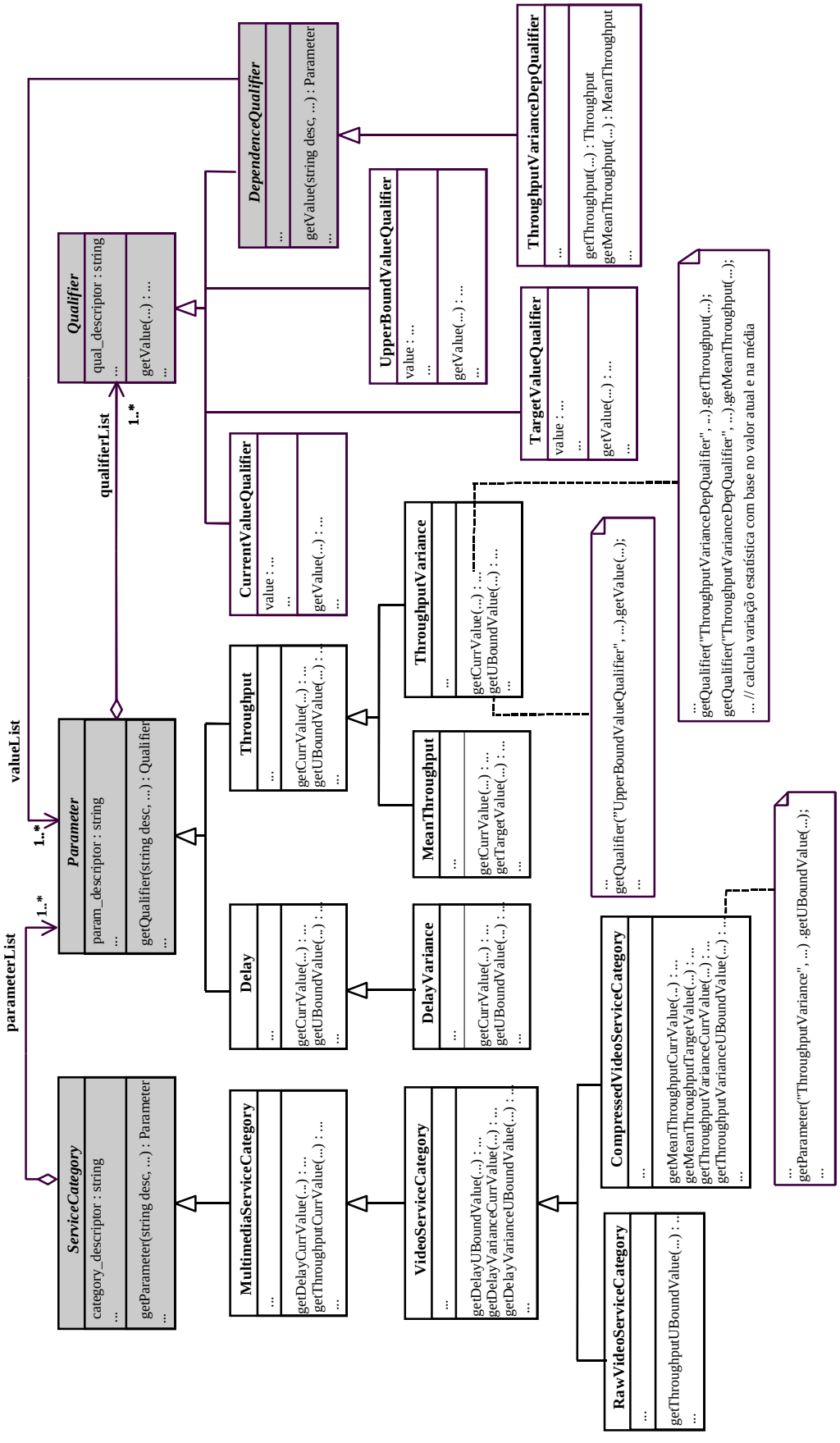


Figura 22: exemplo de instanciação do Framework para Parametrização de Serviços.

Poderíamos também ter definido que somente as subclasses menos especializadas de *Parameter* significariam parâmetros abstratos, enquanto que todos os parâmetros concretos seriam representados por subclasses mais especializadas. Com essa segunda forma de estruturação, teríamos classes como CurrThroughputVariance e UboundThroughputVariance derivadas de *ThroughputVariance* (que nesse caso seria uma classe abstrata), cada uma delas dotada de uma dupla de métodos set(get)Value(), que teriam acesso a instâncias de classes como ValueQualifier, por exemplo. A escolha da primeira forma de estruturação no exemplo da Figura 22 firmou-se meramente na quantidade de classes necessárias para se tentar apresentar sucintamente o exemplo.

O desacoplamento entre *ServiceCategory* e *Parameter* permite, ainda no exemplo da Figura 22, que uma instância de CompressedVideoServiceCategory associe-se dinamicamente a uma instância de uma outra classe que também represente o parâmetro abstrato variação estatística da vazão, ou seja, que tenha a mesma interface (conjunto de duplas set(get)*Value()) e semântica de ThroughputVariance, mas que esteja associada a uma estratégia de monitorização que calcule o valor da variação estatística da vazão de modo mais preciso. A associação dinâmica entre uma categoria de serviço e representações distintas de um mesmo parâmetro pertencente àquela categoria é em geral muito útil, pois a definição de quais estratégias serão adotadas não só pelos mecanismos de monitorização, mas também pelos de mapeamento, influi diretamente na escolha de uma representação interna adequada para os parâmetros por eles manipulados.

Neste sentido, os qualificadores de dependência funcional, modelados na Figura 21 pela classe *DependenceQualifier*, mostram-se muito importantes na tarefa de manter suficientemente flexíveis as estruturas de registro de medições e de mapeamento de parâmetros, para que as estratégias de monitorização e de mapeamento possam ser facilmente substituídas. Especificamente no exemplo da Figura 22, o relacionamento de agrupamento valueList associado a *DependenceQualifier* permite que a estrutura de registro de medição da variação estatística da vazão, formada pelas classes ThroughputVariance, ThroughputVarianceDepQualifier, Throughput e MeanThroughput, possa ser dinamicamente substituída por outra que não envolva esta última classe. Esta mudança de estrutura pode ser necessária, por exemplo, se for adotada uma nova estratégia de monitorização que calcule a variação

estatística da vazão adaptativamente, ao invés de utilizar valores previamente calculados para a vazão média[25].

4.4 Frameworks para Compartilhamento de Recursos

O conceito de *recurso virtual* é primordial para a definição dos elementos do Framework para Provisão de QoS que modelam os mecanismos responsáveis pelo compartilhamento de recursos, cujo escopo abrange as fases de estabelecimento e manutenção de contratos de serviço. Enquanto o Framework para Escalonamento de Recursos modela os mecanismos de escalonamento (que atuam durante a fase de manutenção) com base no conceito de *árvores de recursos virtuais*, o Framework para Alocação de Recursos modela os mecanismos de alocação (que atuam durante a fase de estabelecimento) com base no conceito de *criação de recursos virtuais*.

4.4.1 Árvores de Recursos Virtuais

Uma restrição importante presente na maioria dos fornecedores de serviço atuais é que eles só permitem o estabelecimento e manutenção de contratos relativos a algumas poucas categorias de serviço específicas. Objetivando possibilitar, através do uso de recursos virtuais, a definição de um conjunto amplo e flexível de serviços com características distintas de QoS em um mesmo fornecedor, é introduzido nos Frameworks para Compartilhamento de Recursos o conceito de árvore de recursos virtuais.

A cada recurso real está associada uma árvore de recursos virtuais. Como mostra a Figura 23, em uma árvore qualquer, as folhas são os recursos virtuais pelos quais os fluxos de dados dos usuários têm acesso à sua parcela de utilização do recurso real. A raiz da árvore corresponde a algum componente básico do fornecedor de serviços responsável por gerenciar diretamente o escalonamento do recurso real entre

[25] De fato, análises feitas em [DACHS88] mostram que a representação em ponto flutuante presente na maioria dos sistemas computacionais pode tornar o cálculo da variação estatística de um conjunto de valores muito impreciso, dependendo do algoritmo utilizado. Normalmente, algoritmos adaptativos (onde para cada novo valor usa-se o valor da variação estatística calculada com os valores anteriores e recalcula-se o valor da variação estatística com o novo valor) são mais precisos e menos custosos computacionalmente do que aqueles que baseiam-se na própria fórmula da variação estatística, que envolve o cálculo prévio da média.

recursos virtuais, denominado de *escalonador de recurso real*. Escalonadores de recursos reais podem gerenciar mais de um recurso real de um mesmo tipo simultaneamente, sendo neste caso responsáveis também por permitir que diferentes recursos virtuais de uma mesma árvore possam estar utilizando diferentes recursos reais paralelamente. Podemos citar, como exemplos de escalonadores de recursos reais, as partes do núcleo de um sistema operacional responsáveis pela gerência de baixo nível de processadores e da memória.

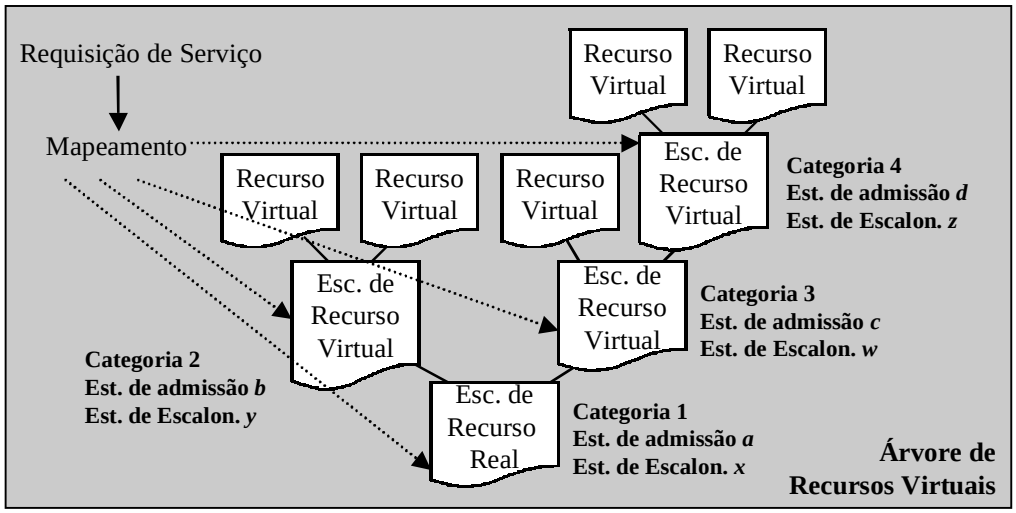


Figura 23: árvores de recursos virtuais.

Os outros nós internos da árvore são recursos virtuais especializados e internos ao fornecedor de serviços, que se ocupam simplesmente de ceder a sua parcela de utilização do recurso real, escalonando-a entre seus recursos virtuais filhos, sendo, portanto, denominados de *escalonadores de recursos virtuais*. Porém, ao contrário dos recursos virtuais, que só podem ter acesso a parcelas de utilização de um único recurso real por vez, os escalonadores de recursos virtuais podem ter acesso simultâneo a mais de um recurso real, para permitir que múltiplos recursos virtuais filhos possam ser atendidos em paralelo. Cada escalonador de recurso virtual, além do próprio escalonador de recurso real, mantém-se associado a uma determinada categoria de serviço e a políticas de provisão de QoS específicas. No contexto do compartilhamento e da orquestração de recursos, essas políticas definem basicamente as estratégias de admissão e escalonamento adotadas. Estas estratégias devem permitir, ao mesmo tempo, que os fluxos de dados dos usuários tenham acesso às suas parcelas desejadas de utilização dos recursos reais, e que estes recursos sejam utilizados de maneira otimizada. Escalonadores de processadores, paginadores de memória e comutadores de

circuitos virtuais são exemplos de mecanismos de escalonamento que podem ser implementados com base no conceito de árvores de recursos virtuais.

4.4.2 Criação de Recursos Virtuais

Considerando-se a estrutura de escalonamento oferecida pelas árvores de recursos virtuais, o processo de criação de recursos virtuais pode ser resumidamente descrito da seguinte forma:

- Como parte do processo de tradução de uma requisição de serviço (efetuado pelos mecanismos de mapeamento, apresentados na Seção 4.5), são definidos os recursos reais aos quais o fluxo de dados do usuário requisitante terá acesso, e que portanto participarão do oferecimento do serviço requisitado.

- Para cada um dos recursos reais envolvidos, a requisição é associada a um escalonador presente na árvore ligada ao recurso real em questão. A escolha apropriada de escalonadores em cada uma das árvores é feita com base nas categorias de serviço a eles associadas. Estas categorias de serviço são obtidas através do mapeamento da categoria de serviço originalmente expressa pela requisição. O mapeamento é necessário na medida em que a categoria expressa pela requisição geralmente pertence a um nível de visão de QoS mais alto do que a das categorias associadas aos recursos, que estão mais relacionadas à capacidade de processamento ou comunicação desses recursos.

- Após o controle de admissão do fluxo desejado pelo usuário (apresentado na Seção 4.5) ter sido efetuado em cada um dos escalonadores escolhidos, os mecanismos de alocação de recursos associados a cada uma das árvores são acionados para criar os recursos virtuais, associá-los a esses escalonadores, e atualizar os parâmetros de desempenho do fornecedor, que serão utilizados pela estratégia de escalonamento interna desses escalonadores.

4.4.3 Componentes do Framework para Escalonamento de Recursos

A Figura 24 mostra as classes e os respectivos relacionamentos que constituem o Framework para Escalonamento de Recursos. As classes *`VirtualResource`*, `RealResourceScheduler` e `VirtualResourceScheduler` representam respectivamente os recursos virtuais, os escalonadores de recursos reais e os escalonadores de recursos virtuais, que em conjunto definem os mecanismos de escalonamento de recursos. Uma variação do pattern estrutural Adapter [GAMMA95] é utilizada para definir os escalonadores de recursos reais e virtuais através de uma única interface de escalonamento genérica, simbolizada pela classe abstrata *`ResourceScheduler`*, que uniformiza esses escalonadores (em geral implementados de maneira bastante distinta entre si). Isto permite, por exemplo, que:

- Recursos virtuais obtenham parcelas de utilização do recurso real diretamente do escalonador de recurso real, ou por intermédio de escalonadores de recursos virtuais, indistintamente, e que
- A infra-estrutura que dá suporte aos serviços oferecidos pelo fornecedor (em termos de recursos reais disponíveis no ambiente) possa ser alterada, sem que isso incorra em uma alteração dos outros mecanismos de provisão de QoS.

No que se refere ao escalonamento, a compatibilização de interfaces entre `RealResourceScheduler` e `VirtualResourceScheduler` é realizada através do método `schedule()` definido em *`ResourceScheduler`*. Este método permite a um escalonador desativar o recurso virtual filho atualmente detentor de um recurso real (através do método `deactivate()`) e ativar o próximo recurso virtual filho a utilizar este recurso (através do método `activate()`). Devido ao fato das árvores de recursos virtuais poderem gerenciar múltiplos recursos reais, em determinados casos de uso do Framework para Escalonamento os métodos `schedule()`, `activate()` e `deactivate()` podem ser dotados de algum argumento que identifique a que recurso real o disparo do método em questão está associado.

Os procedimentos de *ativação* e *desativação* de recursos virtuais dependem, em primeira instância, do tipo de recurso real em questão, ou seja, do

contexto no qual o Framework para Escalonamento de Recursos está sendo aplicado. A ativação e desativação de threads, por exemplo, inclui a troca de contexto parcial ou total de processadores[26], enquanto que a ativação e desativação de páginas de memória inclui o remapeamento de páginas virtuais em page frames.

Os procedimentos de ativação e desativação podem também ser diferentes entre recursos virtuais ligados a um mesmo tipo de recurso real, se considerarmos, por exemplo, as categorias de serviço a que eles se prestam. Este é o caso do processamento de fluxos de dados de mídias contínuas, cuja característica indica uma maior adequação de threads periódicas com relação a outros tipos de threads. O procedimento de ativação de threads periódicas é diferente, por exemplo, do de threads ligadas ao processamento de dados em segundo plano (*background*), na medida em que as primeiras possuem obrigatoriamente um período máximo de execução, seguido geralmente da liberação imediata do recurso real através de desativação voluntária, o que não necessariamente acontece com as últimas. Graças a essa distinção entre tipos de recursos virtuais é que define-se *`VirtualResource`* como uma classe abstrata, a partir da qual é permitida a criação de subclasses que redefinam os métodos abstratos *`activate`*`()` e *`deactivate`*`()`.

Um caso especialmente importante de especialização da classe *`VirtualResource`* é a classe `VirtualResourceScheduler`, que vale-se também de outro relacionamento de especialização com a classe *`ResourceScheduler`* para possibilitar que um escalonador de recursos virtuais seja capaz, ao mesmo, de obter parcelas de utilização de recursos reais (como um recurso virtual), e de escaloná-las entre seus recursos virtuais filhos. Os relacionamentos de especialização supracitados, juntamente com o relacionamento de agregação `virtualResourceList`, modelam a estrutura abstrata das árvores de recursos virtuais.

[26] Em alguns sistemas operacionais, como, por exemplo, o Solaris 2.5 [SOLARIS95], threads são gerenciadas no espaço de endereçamento do processo que as detém (*user-level threads*), sendo desconhecidas pelo núcleo. Logo, o escalonamento de processadores entre threads de um mesmo processo incorre em uma troca de contexto parcial destes processadores. Em outros sistemas, como, por exemplo, o Windows 95 [WINDOWS95], o núcleo está a par da existência de múltiplas threads por processo (*kernel-level threads*), consequentemente o escalonamento de processadores entre threads sempre incorre em uma troca de contexto total destes processadores.

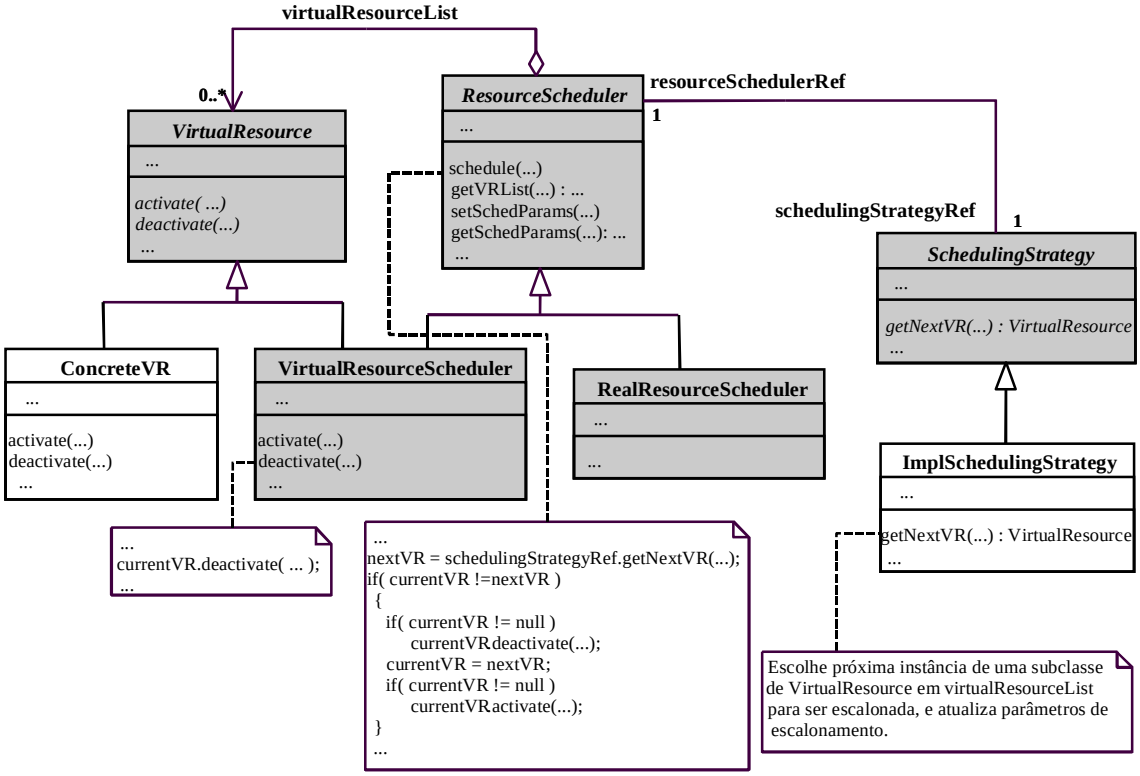


Figura 24: Framework para Escalonamento de Recursos.

A dinâmica de escalonamento associada a uma árvore de recursos virtuais é expressa no Framework para Escalonamento através da seqüência de invocações de métodos entre os componentes do mecanismo de escalonamento, como mostra o exemplo da Figura 25. Esta seqüência é sempre iniciada pela geração de um evento qualquer (associado a um recurso real) no ambiente, que ocasione o disparo do método `schedule()` em algum escalonador presente na árvore e, por conseqüência, a desativação e reativação de recursos virtuais filhos deste escalonador. Exemplos bastante comuns de eventos associados ao escalonamento de recursos são as interrupções (de relógio, de entrada e saída, de desativação voluntária de threads, etc.) captadas pelos sistemas operacionais.

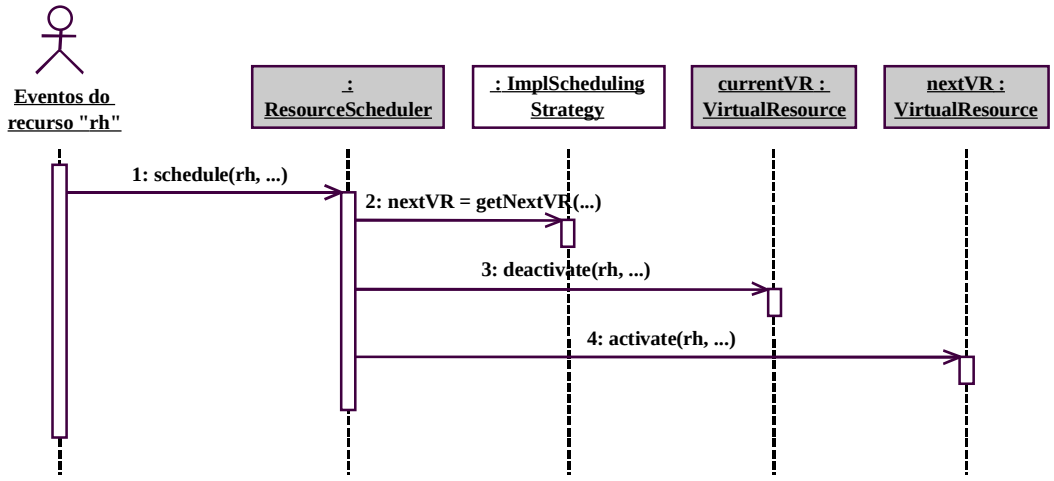


Figura 25: seqüência de invocação de métodos entre componentes de escalonamento.

Caso o recurso virtual filho sendo ativado ou desativado também seja um escalonador (de recurso virtual), os conceitos de ativação e desativação envolvem considerações adicionais. Como mostra o exemplo da Figura 26, a desativação de um escalonador de recurso virtual com relação a um recurso real provoca a desativação imediata de qualquer um de seus recursos virtuais filhos que por ventura esteja de posse da parcela de utilização do recurso real cedida pelo escalonador. Além disso, a geração de eventos associados a este recurso real que ocasionam o disparo de `schedule()` no escalonador é inibida, só sendo restabelecida após a sua reativação. Dessa forma, escalonadores de recurso virtual só podem escalonar parcelas de utilização de um recurso real se eles estiverem realmente de posse do mesmo. É importante atentar para o fato de que, ao contrário do que acontece com os recursos virtuais, que só podem ser ativados e desativados em um único recurso real por vez, a ativação ou desativação de escalonadores de recursos virtuais, com relação a um determinado recurso real, não implica na ativação ou desativação dos mesmos com relação a outros recursos reais gerenciados pela árvore da qual estes escalonadores fazem parte.

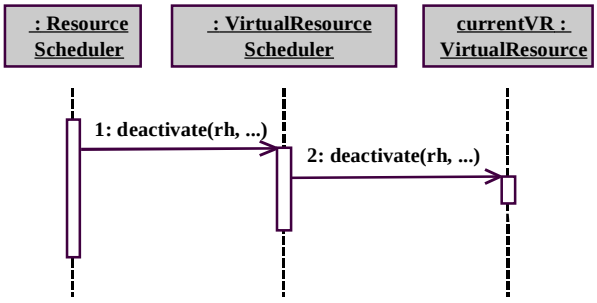


Figura 26: seqüência de desativação de um escalonador de recurso virtual.

Como mostra a Figura 25, quando `schedule()` é disparado em um escalonador, a escolha do próximo recurso virtual a utilizar um recurso real é delegada a uma instância de uma subclasse qualquer da classe abstrata *`SchedulingStrategy`*, através do disparo do método `getNextVR()`. A classe *`SchedulingStrategy`* representa abstratamente as estratégias de escalonamento, sendo as redefinições do método abstrato *`getNextVR`*`()` responsáveis por implementá-las concretamente em cada uma das subclasses de *`SchedulingStrategy`*. O pattern comportamental Strategy [GAMMA95] é utilizado para definir o relacionamento de associação (simbolizado na Figura 24 por `schedulingStrategyRef`) entre os escalonadores e as estratégias de escalonamento correntemente adotadas por eles. Isto possibilita a flexibilidade dos mecanismos de escalonamento com relação às estratégias adotadas, ou até mesmo com relação às categorias de serviço a serem oferecidas por eles. Para que seja possível a um escalonador permitir que `getNextVR()` escolha adequadamente um de seus recursos virtuais filhos, a classe *`ResourceScheduler`* oferece em sua interface os seguintes métodos:

- `getVRList()`: retorna a estrutura da subárvore de recursos virtuais cuja raiz seja o escalonador;
- `getSchedParams()` e `setSchedParams()`: obtém e atualiza parâmetros pertinentes ao escalonamento dos recursos virtuais filhos do escalonador.

O acesso de `getNextVR()` a estes três métodos é representado no framework da Figura 24 através do relacionamento de associação `resourceSchedulerRef` entre *`SchedulingStrategy`* e *`ResourceScheduler`*.

4.4.4 Exemplo de Aplicação do Framework para Escalonamento

A Figura 27 apresenta um caso de uso do Framework para Escalonamento de Recursos, onde é sugerida uma arquitetura para o escalonamento de processadores em sistemas operacionais da família UNIX. Esta arquitetura, baseada no trabalho apresentado em [GOVINDAN91], explora o conceito de *processos leves*

(*Lightweight Processes – LWPs*) já existente nos sistemas operacionais Solaris 2.x[27], acrescentando-lhe algumas características desejáveis com relação ao suporte à provisão de QoS.

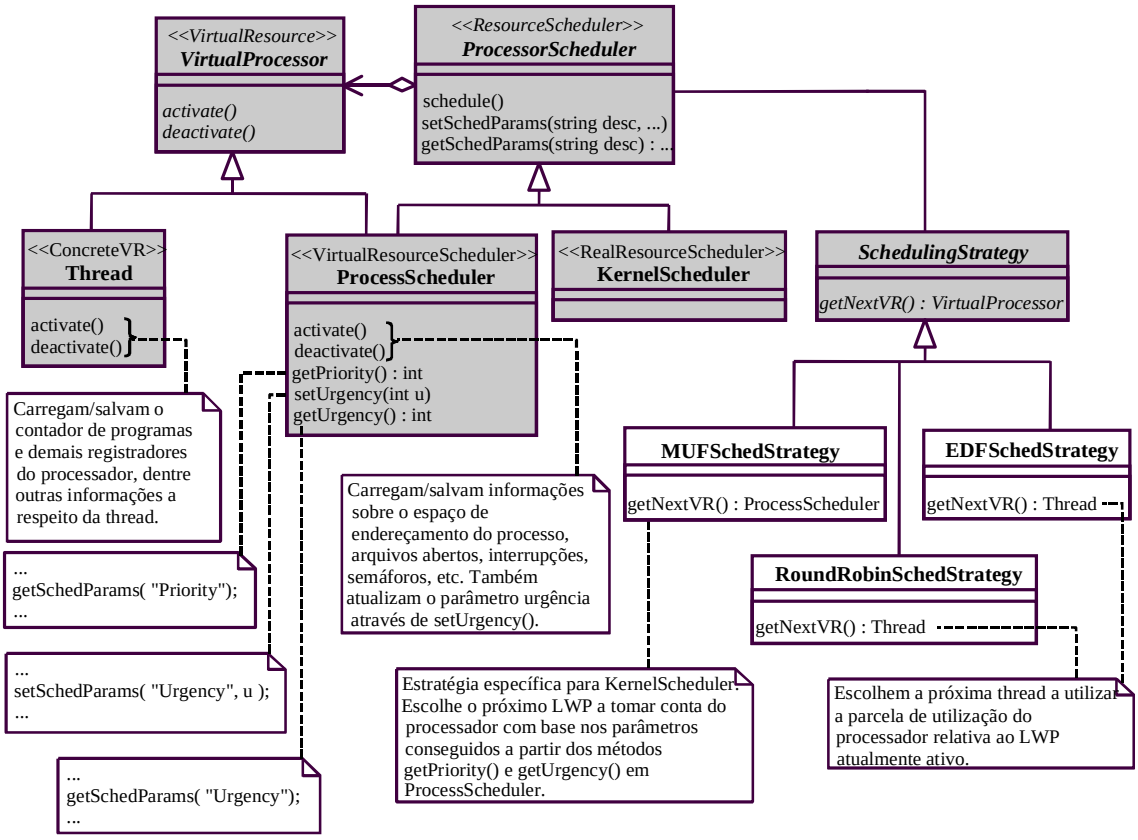


Figura 27: exemplo de aplicação do Framework para Escalonamento de Recursos.

Como mostra a Figura 28, no nível mais baixo da arquitetura encontra-se o *escalonador central* (classe `KernelScheduler`), localizado no espaço de endereçamento do núcleo do sistema operacional, e que é responsável por escalonar o processador entre os LWPs dos vários processos. No nível superior da arquitetura, cada LWP age como um *escalonador de processo* (classe `ProcessScheduler`), responsabilizando-se por escalonar a sua parcela de utilização do processador entre threads (classe `Thread`) que se encontram no espaço de endereçamento do processo que o detém.

Nesta arquitetura, é definida para o escalonador central uma estratégia de escalonamento específica, denominada neste trabalho de *Most Urgent First (MUF* – classe `MUFSchedStrategy`*)*. A grosso modo, esta estratégia define que o LWP a

[27] O exemplo assume algumas simplificações no modelo de processos adotado por esses sistemas, como

possuir o processador será sempre aquele de maior prioridade, dentre todos os LWPs que possuem threads ativáveis no momento. Em caso de serem escolhidos dois ou mais LWPs, o que estiver oferecendo suporte à thread mais urgente, dentre todas as threads controladas por estes LWPs, será o escolhido. O conceito de *urgência* (introduzido em [GOVINDAN91]) depende da estratégia de escalonamento adotada pelos escalonadores de processo. Por exemplo, na estratégia EDF, o que define a urgência das threads é o deadline das mesmas. Já na estratégia Round-Robin, o conceito de urgência é desnecessário, uma vez que esta estratégia não impõe restrições quanto ao retardo de execução das threads.

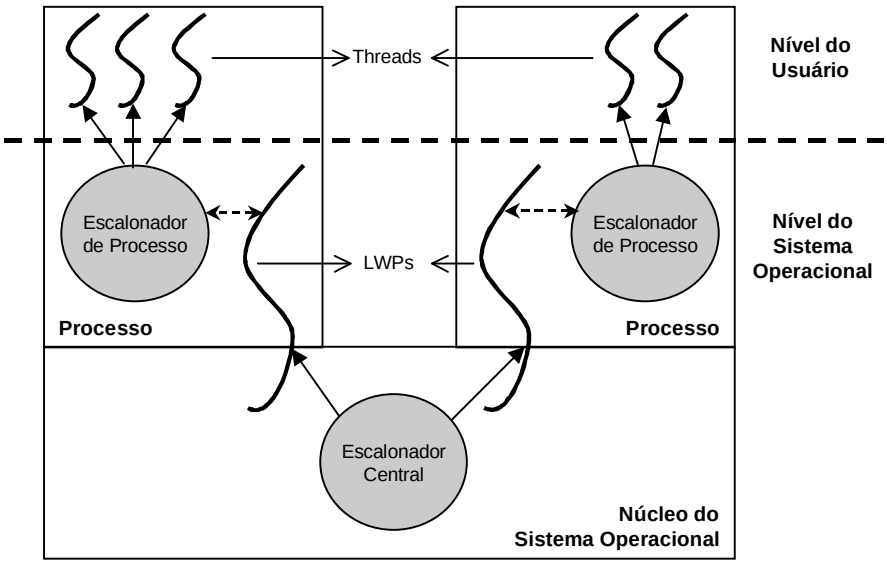


Figura 28: arquitetura para o escalonamento de processadores.

4.4.5 Componentes do Framework para Alocação de Recursos

Na Figura 29, é apresentado o Framework para Alocação de Recursos. A classe `VirtualResourceCreator` representa abstratamente os componentes responsáveis pela criação de recursos virtuais. Estes componentes, em conjunto, definem os mecanismos de alocação de recursos. O pattern de criação Factory Method [GAMMA95] é utilizado para modelar o relacionamento de dependência entre esses componentes e os tipos de recursos virtuais por eles criados. Este relacionamento é simbolizado em `VirtualResourceCreator` pelo método abstrato `createVR()`,

por exemplo, a existência de um único processador por estação.

cujas redefinições em cada uma das subclasses de *VirtualResourceCreator* implementam a instanciação de subclasses específicas de *VirtualResource*.

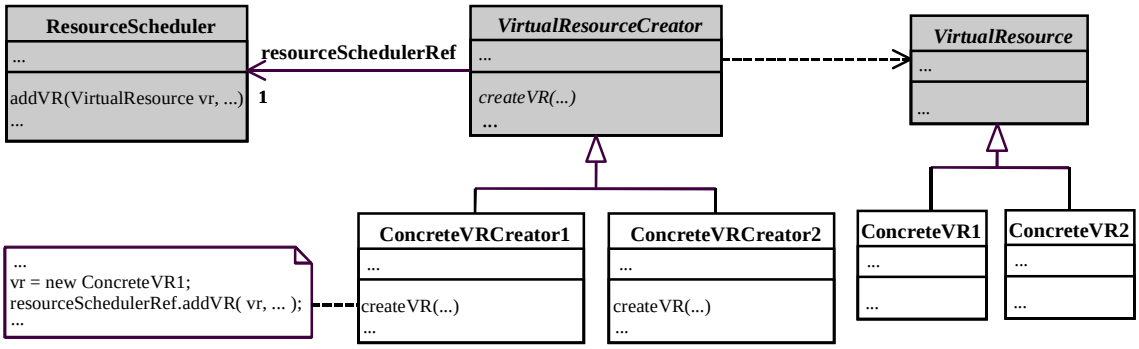


Figura 29: Framework para Alocação de Recursos.

A cada escalonador presente em uma árvore de recursos virtuais está ligado um componente de criação de um determinado tipo de recurso virtual. Esta ligação, representada na Figura 29 através do relacionamento de associação `resourceSchedulerRef` entre as classes *`VirtualResourceCreator`* e `ResourceScheduler`, deve associar tipos específicos de recursos virtuais a estratégias adequadas de escalonamento. Valendo-se novamente do exemplo do processamento de fluxos de dados de mídias contínuas apresentado na Subseção 4.4.3, o uso de threads periódicas só faz sentido quando as estratégias de escalonamento adotadas para essas threads levam em conta o período máximo de execução das mesmas. Como mostra a seqüência de invocações de métodos ilustrada na Figura 30, a ligação entre componentes de criação e escalonadores permite que o método `createVR()` seja implementado de forma que, ao ser disparado em um componente de criação, possa não só criar um recurso virtual, mas também adicioná-lo à lista de recursos virtuais que o escalonador associado possui, e atualizar os parâmetros de escalonamento desse escalonador, por intermédio do método `addVR()` definido em `ResourceScheduler`. Para isto, ambos os métodos `createVR()` e `addVR()` podem ser dotados de argumentos que descrevam os parâmetros de escalonamento relativos aos recursos virtuais a serem criados, dependendo do caso de uso do Framework para Alocação

A definição de um componente de criação de recursos virtuais abstrato permite que novas categorias de serviço possam ser dinamicamente introduzidas em um ambiente, através da definição de componentes concretos (subclasses de *`VirtualResourceCreator`*) ligados à nova categoria, juntamente com a associação

destes componentes a escalonadores que adotem estratégias de escalonamento condizentes com a nova categoria.

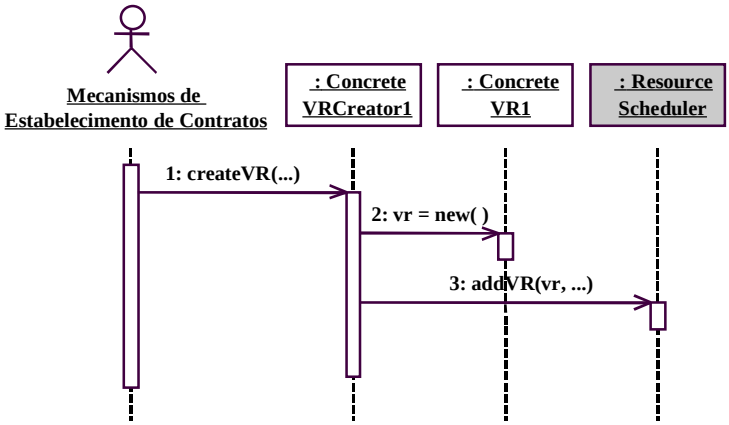


Figura 30: seqüência de criação de um recurso virtual.

4.4.6 Exemplo de Aplicação do Framework para Alocação

A Figura 31 complementa a arquitetura apresentada na Subseção 4.4.4 para melhor estruturar o processo de criação de threads, através do uso de uma variação do Framework para Alocação de Recursos. A classe `ThreadCreator` representa os componentes de criação de threads-padrão (classe `Thread`). O termo *padrão* é usado para designar threads que se preocupam somente em executar, do início ao fim, o código provido pelos usuários[28] do sistema operacional. Ao final do código, a thread é automaticamente destruída. Na arquitetura proposta, threads-padrão são associadas a escalonadores de processo que adotam a estratégia Round-Robin.

Threads periódicas requerem que o código provido pelos usuários seja periodicamente executado. Em sistemas operacionais convencionais, o usuário é obrigado a implementar, internamente ao código, laços de execução indeterminados e esquemas de desativação voluntária, para simular o comportamento de threads periódicas. Para facilitar a tarefa dos usuários com relação ao uso deste tipo de thread, a arquitetura apresentada na Subseção 4.4.4 é estendida na Figura 31 através da definição da classe `RTThread`, derivada de `Thread`. A classe `RTThread` redefine o método `activate()`, para permitir que o código provido pelos usuários possa ser repetitivamente executado. Ao final do código, a thread não é destruída: ela se desativa voluntariamente, e o seu contador de programa é atualizado para o início do código, a

fim de que a thread possa executar novamente em um outro período. Na arquitetura proposta, threads periódicas são associadas a escalonadores de processo que adotam a estratégia EDF.

A classe `RTThreadCreator`, derivada de `ThreadCreator`, representa os componentes de criação de threads periódicas. Segundo a arquitetura proposta, quando um usuário escolhe uma estratégia de escalonamento de threads a ser adotada internamente ao seu processo, deve ser criada neste processo uma instância de `ThreadCreator` ou `RTThreadCreator`, dependendo da estratégia escolhida, para viabilizar, através do método `createThread()`, a criação de threads internamente ao processo, além da correta associação delas com os escalonadores de processo apropriados, através do método `addThread()` em `ProcessScheduler`.

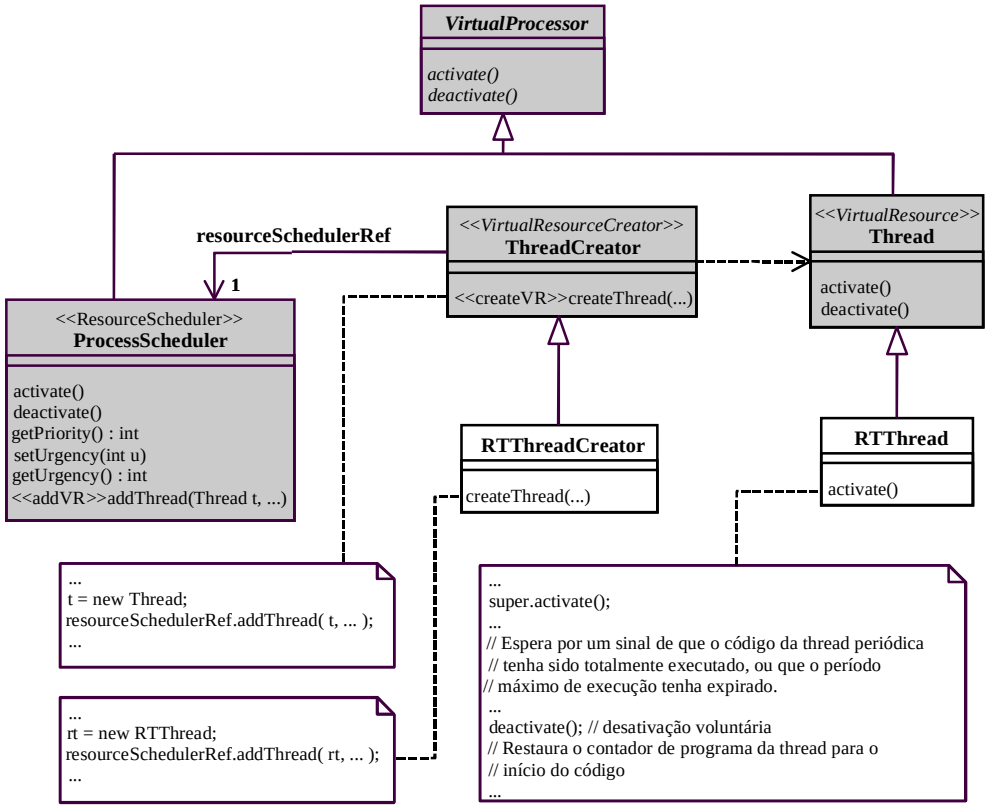


Figura 31: exemplo de aplicação do Framework para Alocação de Recursos.

[28] Neste contexto, usuários são os programadores de aplicações.

4.4.7 Considerações a Respeito dos Frameworks para Compartilhamento de Recursos

A grande vantagem da arquitetura apresentada na Subseção 4.4.4 e estendida na Subseção 4.4.6, no que se refere aos usuários do sistema operacional, decorre da flexibilidade oferecida pelos Frameworks para Compartilhamento com relação às estratégias a serem adotadas pelos LWPs de cada processo. Esta arquitetura viabiliza a definição de mecanismos de configuração de alto nível (por exemplo, através de APIs) que possibilitem aos usuários escolherem uma ou mais estratégias de escalonamento de threads a serem adotadas internamente aos seus processos. Na arquitetura apresentada, somente as estratégias EDF (classe `EDFSchedStrategy`) e Round-Robin (classe `RRSchedStrategy`) são oferecidas. Entretanto, os Frameworks para Compartilhamento não fazem nenhuma restrição quanto ao número e tipos de estratégias a serem oferecidas pela arquitetura. Além disso, a arquitetura poderia ser modelada de maneira a permitir que o administrador do sistema operacional inserisse novas estratégias dinamicamente, contanto que fossem definidos, para cada uma dessas estratégias, a prioridade e o critério de urgência com relação ao escalonador central, além da categoria de serviço a qual a estratégia estaria associada. As restrições feitas à arquitetura proposta foram simplesmente para tentar simplificar o exemplo.

É importante atentarmos para o fato de que as classes *`VirtualResource`* e `ResourceScheduler` presentes no Framework para Alocação são *conceitualmente* idênticas às classes *`VirtualResource`* e *`ResourceScheduler`* apresentadas no Framework para Escalonamento. Em ambos os frameworks, estas classes representam, respectivamente, recursos virtuais e escalonadores. Porém, relembrando o que foi citado no começo deste capítulo, o Framework para Provisão de QoS permite a construção de partes isoladas de um sistema, através da aplicação de frameworks específicos. Portanto, não é obrigatório, embora seja desejável (como foi feito na arquitetura apresentada), que um projetista aplique ambos os Frameworks para Compartilhamento em todos os casos de uso.

No contexto do Framework para Alocação, por exemplo, o que importa é a existência de uma classe que modele recursos virtuais, e outra que modele escalonadores que tenham interfaces que permitam a inserção de novos recursos virtuais em sua estrutura de escalonamento. Não interessa a este framework a interface de

ativação e desativação oferecida pelos recursos virtuais, ou a interface de escalonamento oferecida pelos escalonadores. O uso de nomes semelhantes para classes em frameworks distintos não objetiva confundir, mas sim, reforçar a idéia de que as classes ilustradas nos frameworks apresentados neste capítulo são muito mais conceitos do que classes de objetos propriamente ditas, e que estes conceitos aparecem recorrentemente (quase sempre sob óticas diferentes) nos vários frameworks que compõem o Framework para Provisão de QoS.

4.5 Frameworks para Orquestração de Recursos

Mecanismos de alocação e escalonamento de recursos estão dispersos nos vários subsistemas presentes em um ambiente distribuído. Mesmo no escopo de um único subsistema, a cada tipo de recurso real está associado um mecanismo de alocação e escalonamento diferente. Estes mecanismos devem ser gerenciados de maneira integrada, tanto internamente aos subsistemas quanto entre eles, em todas de fases de provisão de QoS, para viabilizar a orquestração de recursos no ambiente como um todo.

Neste trabalho, a definição dos elementos do Framework para Provisão de QoS que modelam os mecanismos responsáveis pela orquestração de recursos fundamenta-se no conceito de fluxo. O Framework para Negociação da QoS modela os mecanismos de negociação e mapeamento (que atuam durante as fases de requisição e estabelecimento), assim como os de controle de admissão (que atuam durante a fase de estabelecimento), com base no conceito de *concatenação entre fluxos*, enquanto o Framework para Sintonização da QoS modela os mecanismos de sintonização e monitorização (que atuam durante a fase de manutenção) com base no conceito de *balanceamento entre fluxos*.

4.5.1 Fluxos de Dados

Fluxos de dados figuram como a realização dos serviços de processamento e comunicação requisitados pelos usuários. Em ambientes que envolvem múltiplas escalas de distribuição e níveis de abstração, fluxos de dados podem ser recursivamente compostos de outros fluxos. Como mostra o exemplo da Figura 32, um

fluxo de vídeo entre duas pessoas usuárias de um sistema de videoconferência pode ser visto como uma composição de:

- Fluxos que representam a troca de quadros desse vídeo entre as aplicações utilizadas no sistema e as entidades de protocolo da camada mais alta de um sistema de comunicação, e de
- Um fluxo de pacotes (encapsulando os quadros do vídeo) entre as entidades pares do protocolo dessa camada.

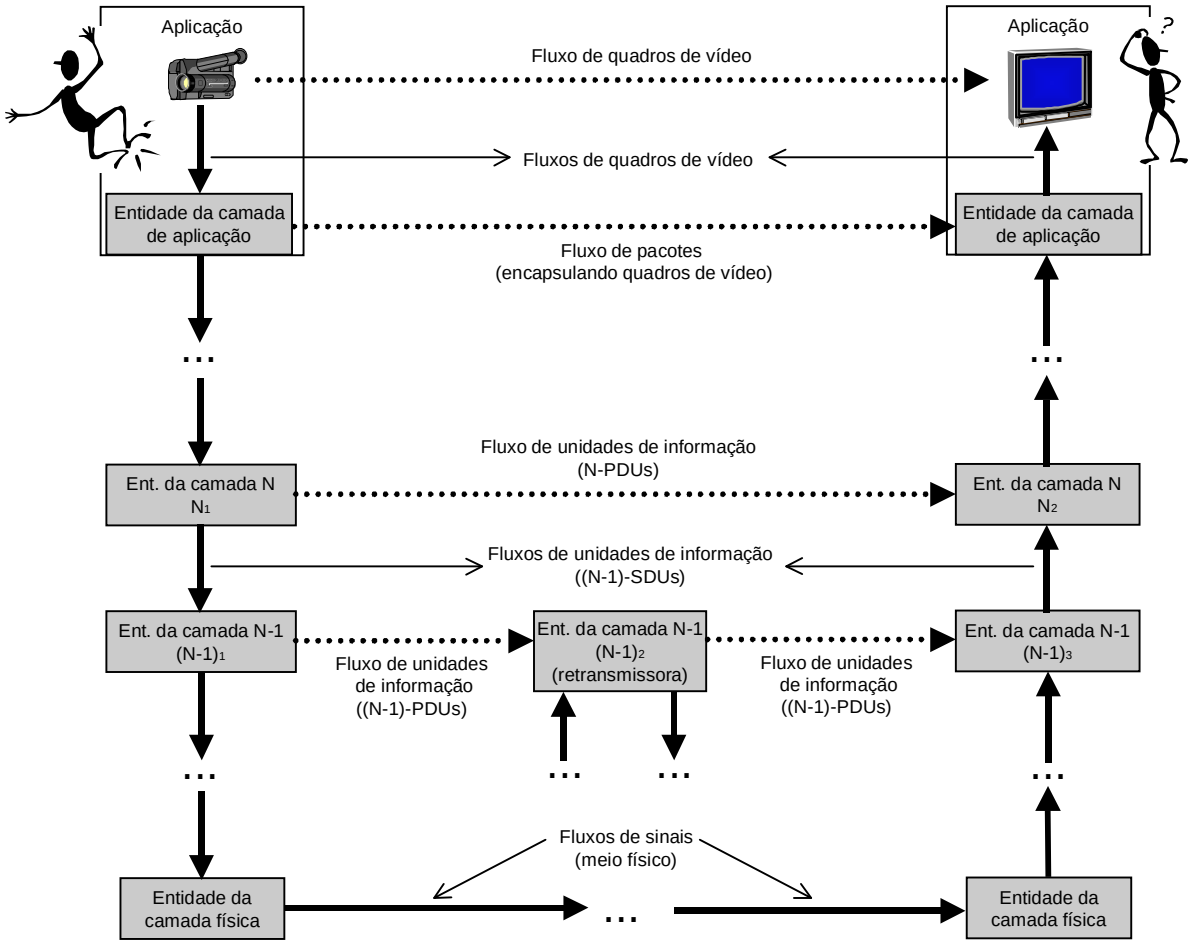


Figura 32: composição de fluxos em um sistema multimídia distribuído.

Este exemplo pode ser facilmente generalizado para as outras camadas do sistema de comunicação, se considerarmos que o fluxo de unidades de informação entre duas entidades pares de um protocolo de uma camada N (N > 1) pode ser visto como uma composição de:

- Fluxos que representam a troca dessas unidades de informação entre entidades de protocolo das camadas adjacentes N e N-1, e de

- Fluxos que representam a troca de unidades de informação (que encapsulam as unidades de informação referentes à camada N) entre entidades pares de um protocolo da camada N-1.

O exemplo anterior mostra como a comunicação horizontal presente em uma camada qualquer (excetuando-se, obviamente, a camada física, onde o fluxo corresponde à transmissão de sinais por um meio físico) pode ser recursivamente definida em termos de comunicações horizontais presentes em camadas inferiores e comunicações verticais que interligam essas camadas. Contudo, a noção de *horizontalidade* é bem mais ampla, se atentarmos não só para a diversidade de níveis de abstração, mas também para a de escalas de distribuição envolvidas em um ambiente distribuído. Focalizando somente os ambientes locais de execução, como mostra o exemplo da Figura 33, podemos considerar a comunicação entre entidades de protocolo de camadas adjacentes como também sendo uma comunicação horizontal. Ou seja, os fluxos existentes entre essas entidades também podem ser decompostos, dependendo da escala de distribuição das mesmas.

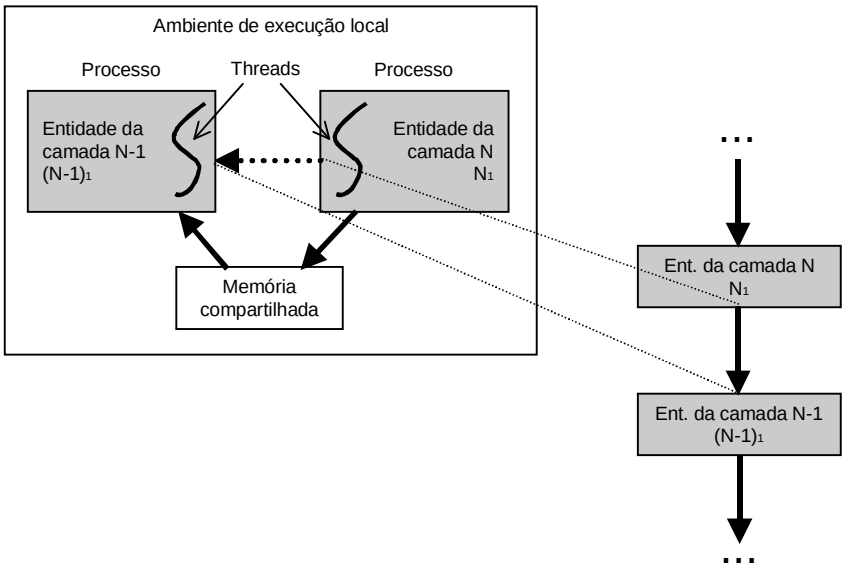


Figura 33: composição de fluxos em um ambiente de execução local.

No caso da Figura 33, as entidades estão localizadas em processos distintos, e a comunicação entre elas se dá através de memória compartilhada. O fluxo de unidades de informação entre as duas entidades (no sentido da camada N para a camada N-1) pode ser representado por duas threads: uma responsável pela escrita na memória das unidades de informação provenientes da camada N, e outra responsável

pela leitura dessas unidades de informação por parte da camada N-1. Este é um caso particular em que um fluxo de dados é "composto" por fluxos de instruções. Rigorosamente, o que ocorre é que o comportamento conjunto das threads, isto é, a forma como o processador é escalonado entre elas, define o comportamento do fluxo de dados entre as entidades[29].

4.5.2 Concatenação entre Fluxos de Dados

Quando o mecanismo de negociação da QoS recebe uma requisição de serviço, ele primeiramente identifica todos os recursos reais que podem se envolver no fornecimento do serviço requisitado, e associa a cada um deles uma parcela da responsabilidade pela provisão da QoS especificada na requisição. Este processo é efetuado, em parte, pelo mecanismo de mapeamento da QoS, responsável por traduzir a requisição em parâmetros conhecidos internamente ao fornecedor. Em face da possibilidade deste fornecedor ter diversos níveis internos de visão de QoS, sucessivas traduções podem ser necessárias até que se consigam parâmetros relacionados diretamente à capacidade de processamento ou comunicação dos recursos reais a serem envolvidos no fornecimento do serviço. Estes parâmetros de mais baixo nível permitem que o mecanismo de negociação associe a requisição do usuário a escalonadores apropriados em cada uma das árvores de recursos virtuais ligadas a esses recursos reais.

Durante a identificação dos recursos reais envolvidos, há também o acionamento do mecanismo de controle de admissão, que valida a aceitação do novo fluxo no ambiente, utilizando-se para isso de estratégias de admissão de novos recursos virtuais em cada um dos escalonadores escolhidos. Caso a validação seja positiva em todos os recursos envolvidos, os mecanismos de alocação de recursos são acionados, conforme descrito na Subseção 4.4.2. A cada um dos recursos virtuais criados corresponderá uma parcela da responsabilidade pela provisão da QoS especificada na requisição. Se, no entanto, a validação for negativa em um ou mais recursos, a requisição pode ser imediatamente negada, ou então o mecanismo de negociação pode reiniciar o processo de identificação, redistribuindo as parcelas de responsabilidade pela provisão da QoS. Em alguns casos, o mecanismo de controle de admissão pode atuar

[29] No exemplo, desprezou-se a influência dos mecanismos de paginação de memória na definição do comportamento do fluxo de dados.

também na reserva de recursos[30], como acontece, por exemplo, em protocolos de configuração de fluxos semelhantes aos apresentados na Subseção 3.2.3.

Em sistemas multimídia distribuídos, os mecanismos de negociação da QoS podem servir-se também de outros mecanismos não modelados pelo Framework para Provisão de QoS, como os de roteamento, por exemplo, para identificar os recursos reais aos quais o fluxo de dados do usuário requisitante terá acesso. Mecanismos de roteamento são modelados no Framework para Provisão de Serviço de Multicast [RODRIGUES99], e a integração entre os mecanismos de negociação e roteamento é exemplificada em um cenário de utilização do Framework para Provisão de QoS apresentado no Capítulo 5.

É importante perceber a característica inerentemente distribuída dos mecanismos de negociação, mapeamento e controle de admissão. Mais ainda, essa distribuição pode não ocorrer em uma única escala específica, devido à própria definição recursiva de fluxo. Continuando com o exemplo da Figura 32, a criação de um fluxo entre as entidades da camada N (simbolizado por $N_1 \rightarrow N_2$) envolve a criação conjunta de fluxos $N_1 \rightarrow (N-1)_1$, $(N-1)_1 \rightarrow (N-1)_2$, $(N-1)_2 \rightarrow (N-1)_3$ e $(N-1)_3 \rightarrow N_2$. Embora os fluxos $N_1 \rightarrow N_2$, $(N-1)_1 \rightarrow (N-1)_2$ e $(N-1)_2 \rightarrow (N-1)_3$ estejam em níveis distintos de abstração, os dois últimos assemelham-se ao primeiro quanto à escala de distribuição das entidades envolvidas, e por conseguinte, a criação de $(N-1)_1 \rightarrow (N-1)_2$ e $(N-1)_2 \rightarrow (N-1)_3$ pode vir a envolver a criação conjunta de outros fluxos de maneira análoga a $N_1 \rightarrow N_2$. Este procedimento de criação dos fluxos dos usuários a partir de outros fluxos internos ao fornecedor é denominado, neste trabalho, de *concatenação entre fluxos*. Por envolver, em última instância, a divisão da responsabilidade pela provisão da QoS entre os recursos aos quais o fluxo do usuário terá acesso, a concatenação entre fluxos é gerenciada pelo mecanismo de negociação da QoS. Detendo a atenção somente na comunicação entre entidades de uma mesma camada, percebemos que a concatenação entre fluxos pode continuar até um caso-base, onde, por exemplo, um fluxo entre entidades pares possa ser mapeado diretamente em uma conexão (física ou lógica).

[30] A reserva de recursos pode ser encarada como uma tarefa de criação de recursos virtuais que só podem ser ativados após confirmação por parte do mecanismo de alocação de recursos.

Por outro lado, considerando os fluxos $N_1 \rightarrow (N-1)_1$ e $(N-1)_3 \rightarrow N_2$ entre as camadas adjacentes N e N-1, notamos que estes apresentam características locais, isto é, as entidades envolvidas podem estar em processos distintos de uma mesma estação ou em um mesmo processo, e neste último caso, os métodos dos componentes que implementam essas entidades (e que estão associados diretamente ao processamento do fluxo) podem estar sendo executados por threads distintas ou por uma mesma thread. Independente da escala de distribuição das entidades N_1, $(N-1)_1$, $(N-1)_3$ e N_2, a criação dos fluxos $N_1 \rightarrow (N-1)_1$ e $(N-1)_3 \rightarrow N_2$ envolve a criação conjunta das threads responsáveis pelo processamento destes fluxos existente nessas entidades.

4.5.3 Balanceamento entre Fluxos de Dados

Quando o mecanismo de sintonização da QoS é informado da admissão de um recém-criado fluxo de usuário, ele deve responsabilizar-se por reduzir ao mínimo a probabilidade que violações no contrato estabelecido entre o usuário e o fornecedor venham a ocorrer, o que só é viável se ele tiver controle sobre todos os recursos reais envolvidos no fornecimento do serviço. Para isto, o mecanismo de sintonização dispõe do mecanismo de monitorização de fluxos, que policia os recursos virtuais aos quais o fluxo do usuário tem acesso. Este controle se dá através da geração periódica de medições relativas à real QoS sendo oferecida por esses recursos virtuais. Estas medições são obtidas a partir de parâmetros de desempenho do fornecedor relacionados (direta ou indiretamente) à carga gerada pelos recursos virtuais. Um exemplo típico de monitorização é a medição do retardo de trânsito médio dos dados de um fluxo em uma rede de distribuição. Através de *timestamps* colocados nos pacotes referentes a esse fluxo, o mecanismo de monitorização consegue estimar o retardo de um pacote isolado, e, a partir daí, gerar medições para o retardo de trânsito médio dos dados do fluxo. Esse retardo está relacionado, entre outras coisas, ao modo como os vários enlaces presentes na rede de distribuição estão sendo escalonados entre os pacotes desse fluxo.

O mecanismo de monitorização de um fluxo é também responsável por enviar alertas ao mecanismo de sintonização, que, ao recebê-los, reavalia as medições feitas pelo mecanismo de monitorização. Dependendo da política de provisão de QoS adotada, o mecanismo de sintonização pode agir somente para remediar a ocorrência de

uma violação de contrato, ou então para prevenir a mesma, quando há risco iminente do contrato ser violado. A escolha do tipo de ação a ser tomada pelo mecanismo de sintonização está sujeita também ao grau de depreciação da QoS anteriormente negociada. Dentre as ações mais comuns, estão[31]:

- Repassar o alerta, juntamente com as medições associadas, ao usuário, o que implica na tradução, através do mecanismo de monitorização, dessas medições em informações condizentes com o nível de visão de QoS do usuário.

- Reconfigurar um ou mais escalonadores envolvidos no fornecimento do serviço. Na maioria dos casos, uma violação de contrato está relacionada ao fato de que um escalonador não consegue mais oferecer a um recurso virtual envolvido no fornecimento do serviço a sua parcela de utilização do recurso real associado. Quando o fluxo do usuário tem acesso a mais de um recurso real, a reconfiguração de escalonadores permite que, dado que um recurso virtual falhe em contribuir com a sua parcela de responsabilidade pela provisão da QoS, outros recursos virtuais tenham suas parcelas de utilização de recursos redimensionadas, de modo que a QoS fim-a-fim, conforme observada pelo usuário, não seja afetada.

- Escolher recursos reais alternativos, quando a reconfiguração de escalonadores se mostra insuficiente. Neste caso, ocorre uma espécie de renegociação sem intervenção do usuário, podendo, inclusive, haver a participação de outros mecanismos não modelados pelo Framework para Provisão de QoS, como foi citado na Subseção 4.5.2.

- Acionar o mecanismo de negociação, para que este renegocie com o usuário um novo contrato de serviço factível de ser satisfeito.

Os mecanismos de monitorização e de sintonização são também recursivamente distribuídos. Ainda no exemplo da Figura 32, violações sucessivas de contrato no fluxo $N_1 \rightarrow N_2$ podem decorrer da falha de um ou mais recursos virtuais envolvidos no fluxo $(N-1)_1 \rightarrow (N-1)_2$. Se o mecanismo de sintonização não

[31] Relembrando, estas ações podem ser definidas durante a fase de requisição de serviços, como ressaltado na Subseção 2.8.1.

conseguir normalizar este fluxo através da reconfiguração[32] do mesmo, é possível que ele tenha que redistribuir a parcela de responsabilidade pela provisão da QoS entre recursos virtuais envolvidos com os outros fluxos $N_1 \rightarrow (N-1)_1$, $(N-1)_2 \rightarrow (N-1)_3$ e $(N-1)_3 \rightarrow N_2$. Este procedimento de manutenção da QoS associada aos fluxos dos usuários a partir da reconfiguração de outros fluxos internos ao fornecedor é denominado, neste trabalho, de balanceamento entre fluxos.

4.5.4 Modelos de Concatenação e Balanceamento

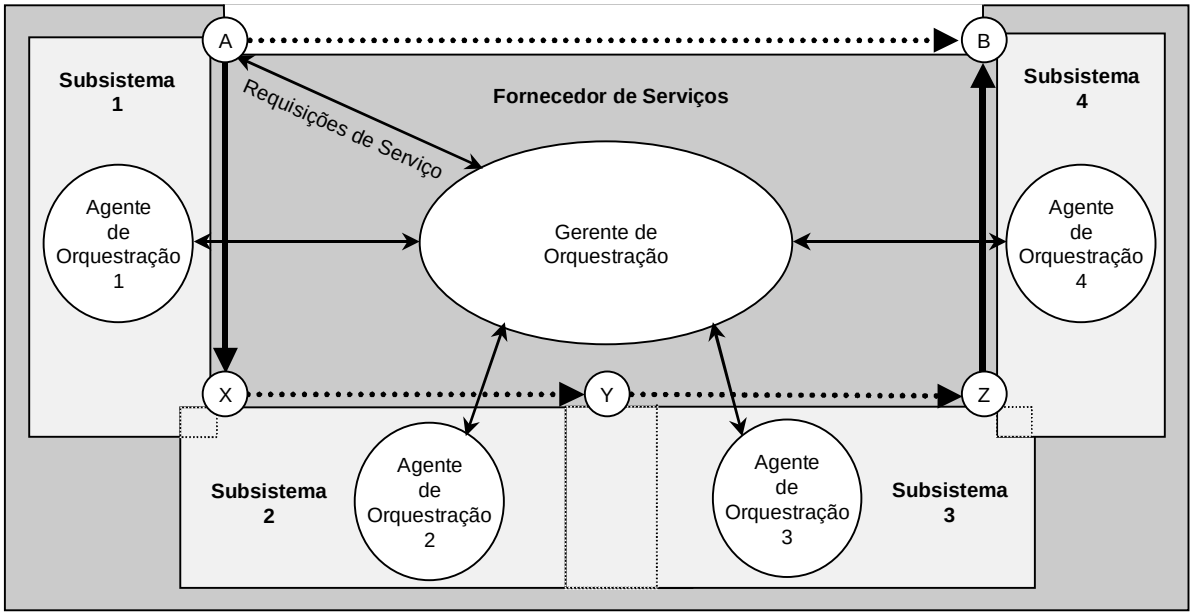
Pelo que foi apresentado nas subseções anteriores, nota-se que os mecanismos responsáveis pela orquestração de recursos devem atuar em todas as escalas de distribuição e níveis de abstração presentes em um ambiente. Dentre estes mecanismos, os de negociação e sintonização da QoS desempenham papéis centrais. São eles que efetivam a integração entre diferentes escalas de distribuição e níveis de abstração, através da concatenação e do balanceamento entre fluxos localizados nos diferentes subsistemas presentes no ambiente do fornecedor. Esta integração pode ser implementada segundo dois modelos distintos, mas que, como veremos, estão intimamente relacionados:

- O *modelo agente-gerente* e
- O *modelo peer-to-peer*.

Modelo Agente-Gerente

O modelo agente-gerente, ilustrado na Figura 34, atribui a um fornecedor de serviços um *gerente de orquestração* que responde pela coordenação de *agentes de orquestração* atuantes em cada um dos subsistemas. Cada agente é responsável pela gerência de fluxos em um subsistema específico, enquanto o gerente trata da integração entre eles, fazendo o papel de uma interface de gerência de fluxos (compostos por fluxos em cada um dos subsistemas) que realizam os serviços requisitados pelos usuários.

[32] É importante distinguir *reconfiguração de fluxos* e *reconfiguração de escalonadores*, uma vez que a reconfiguração de fluxos pode envolver tanto a reconfiguração de escalonadores quanto a escolha de recursos reais alternativos.



Fornecedor de Serviços:
- usuários: A e B.
- componentes internos: X, Y, Z, gerente e agentes de orq. 1, 2, 3 e 4.

Subsistema 1:
- usuários: A e X.
- componentes internos: agente de orquestração 1.

Subsistema 2:
- usuários: X e Y.
- componentes internos: agente de orquestração 2.

Subsistema 3:
- usuários: Y e Z.
- componentes internos: agente de orquestração 3.

Subsistema 4:
- usuários: Z e B.
- componentes internos: agente de orquestração 4.

Figura 34: modelo agente-gerente de orquestração de recursos.

Modelo Peer-to-Peer

Ao contrário do modelo agente-gerente, que define o gerente de orquestração como sendo um componente central que controla outros componentes (isto é, os agentes de orquestração) atuantes nos diversos subsistemas, no modelo peer-to-peer, ilustrado na Figura 35, o gerente não é um componente propriamente dito, mas sim uma abstração representada pela composição de agentes parceiros interdependentes. Isto significa que todos os agentes são igualmente responsáveis não só pela gerência de fluxos em seus subsistemas, mas também por tratar da integração entre fluxos presentes em todos os outros subsistemas. Assim, o papel de interface de gerência de fluxos, no modelo peer-to-peer, é feito de maneira conjunta por todos os agentes parceiros.

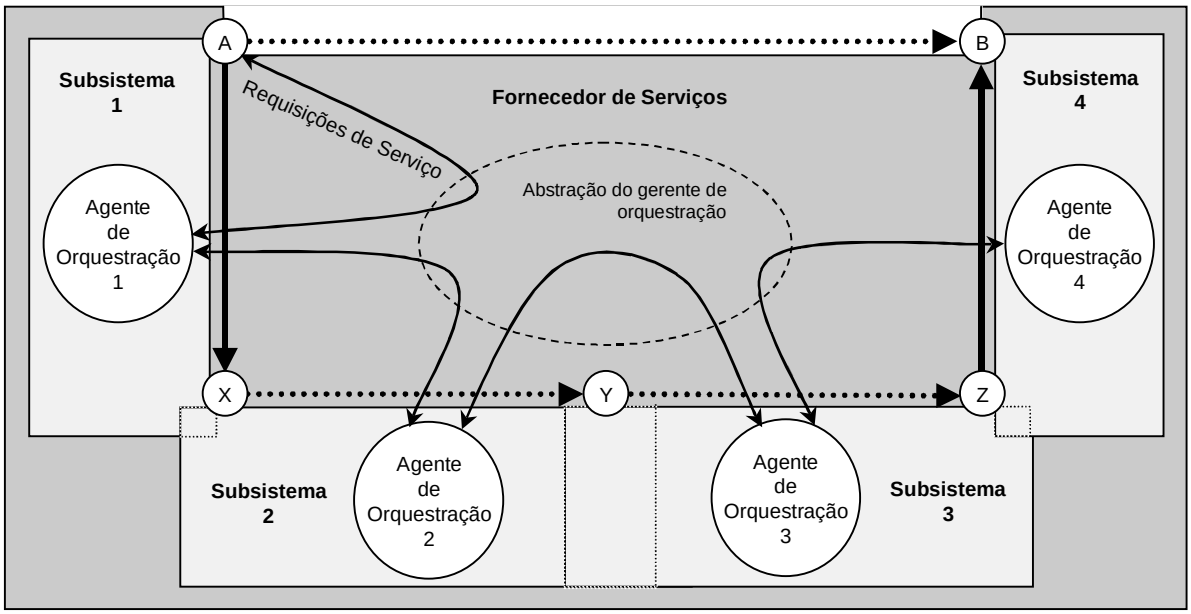


Figura 35: modelo peer-to-peer de orquestração de recursos.

Considerações a Respeito dos Modelos Agente-Gerente e Peer-to-Peer

A legenda na parte inferior da Figura 34 destaca o fato de que um subsistema desempenha, sob o ponto de vista do seu nível de abstração, o papel de um fornecedor de serviços. Nos exemplos apresentados na Figura 34 e na Figura 35, os componentes A e X são usuários do subsistema 1. Já no nível de abstração do fornecedor de serviços de mais alto nível, o componente A é um usuário, enquanto o componente X é um componente interno do fornecedor. Os agentes de orquestração localizados nos subsistemas englobados pelo fornecedor de mais alto nível são na verdade interfaces de gerência de fluxos nesses subsistemas. Cada uma dessas interfaces pode ser representada por um gerente (seja ele um componente ou uma abstração) que coordena agentes só conhecidos internamente ao subsistema. Dessa forma, a distinção entre agente e gerente só faz sentido no contexto de um único nível de abstração. Nos níveis mais baixos de abstração, agentes funcionam também como *gerentes de recursos*, cuja tarefa principal é coordenar os componentes que implementam os mecanismos responsáveis pelo compartilhamento de recursos do fornecedor.

Pelo fato da gerência de fluxos em um fornecedor ser mascarada por intermédio de uma interface, não importa aos usuários qual o modelo de integração utilizado, contanto que ele tenha acesso (direto ou indireto) a essa interface. De modo geral, quando o ambiente envolve múltiplos níveis de abstração, o fornecedor de

serviços pode utilizar, para cada nível, um modelo diferente de integração com os níveis adjacentes. Além disso, os mecanismos de negociação e de sintonização podem ser implementados segundo diferentes modelos em um mesmo nível de abstração.

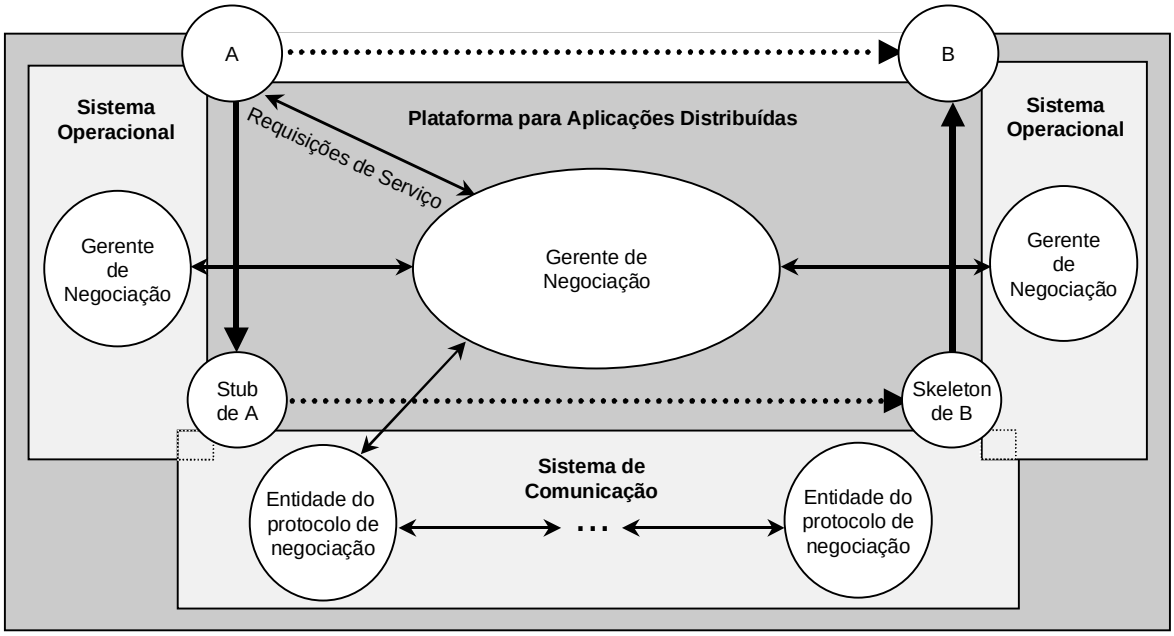


Figura 36: concatenação de fluxos em um ambiente distribuído.

O exemplo da Figura 36 mostra como ambos os modelos podem ser simultaneamente aplicados na implementação do mecanismo de negociação da QoS em um ambiente distribuído. No nível de abstração da plataforma para aplicações distribuídas, o gerente responsável pela negociação da QoS é centralizado, o mesmo acontecendo no nível de abstração dos sistemas operacionais. Já no nível de abstração do sistema de comunicação é adotado o modelo peer-to-peer, através da implementação de um protocolo de negociação. O gerente de negociação da QoS da plataforma para aplicações distribuídas não precisa ter conhecimento do modelo utilizado nos sistemas operacionais e de comunicação, desde que ele saiba endereçar os componentes das aplicações usuárias da plataforma, a fim de que possa criar corretamente tanto o fluxo vertical entre estes componentes e os componentes internos que definem os stubs e skeletons da plataforma, quanto o fluxo horizontal entre esses componentes internos. Considerações a respeito do endereçamento de componentes em ambientes genéricos de comunicação não estão no escopo deste trabalho, podendo ser encontradas em [RODRIGUES99] maiores informações sobre este assunto.

4.5.5 Componentes do Framework para Negociação da QoS

A Figura 37 mostra as classes e os respectivos relacionamentos que constituem o Framework para Negociação da QoS. A classe *QoSNegotiator* simboliza abstratamente os componentes que, em conjunto, implementam os mecanismos de negociação da QoS. O pattern estrutural Facade [GAMMA95] é utilizado para mascarar, sob uma interface única de requisição de serviços (provida pelo método abstrato *request()* em *QoSNegotiator*), toda a complexidade associada ao estabelecimento de contratos de serviço. Em todos os casos de uso possíveis do Framework de Negociação, o método *request()* deve ser dotado de argumentos que descrevam, através de parâmetros de caracterização de carga e de especificação da QoS, o comportamento do fluxo requisitado pelo usuário.

Outro pattern utilizado para representar a estrutura de concatenação de fluxos é o Chain of Responsibility [GAMMA95], simbolizado pelo relacionamento recursivo de associação `qosNAgentList` definido em *QoSNegotiator*. Este relacionamento permite que:

- O Framework de Negociação da QoS seja recursivamente aplicado em vários níveis de abstração presentes em um ambiente, e que
- Para cada nível de abstração onde o Framework de Negociação esteja sendo aplicado, qualquer um dos modelos de integração (agente-gerente ou peer-to-peer) com os níveis adjacentes possa ser utilizado, independente do modelo adotado nestes outros níveis.

A estrutura resultante da combinação entre os patterns Facade e Chain of Responsibility permite que ambos os modelos de integração possam ser usados de maneira transparente pelos usuários de um fornecedor. Qualquer que seja o nível de abstração em questão, o modelo de integração utilizado será determinado através da criação de subclasses derivadas de *QoSNegotiator* que redefinam o método *request()* apropriadamente. Assim, subclasses de *QoSNegotiator* que modelam gerentes de negociação da QoS centralizados (representados na figura por `QoSNManager`) devem redefinir `request()`, de modo que estes componentes saibam distribuir a responsabilidade pelo estabelecimento de contratos de serviço entre os seus agentes de negociação (representados por `QoSNAgent`). Já as subclasses que modelam

agentes de negociação parceiros devem redefinir `request()`, de maneira que estes componentes saibam coordenar *entre si* esta divisão de responsabilidades. Quando os agentes modelados são responsáveis também pela gerência direta de recursos, a redefinição do método `request()` nas subclasses de *`QoSNegotiator`*, que modelam estes gerentes de recursos (representados por `QoSNResourceManager`), deve permitir que estes componentes coordenem os componentes responsáveis pela admissão e criação de recursos virtuais.

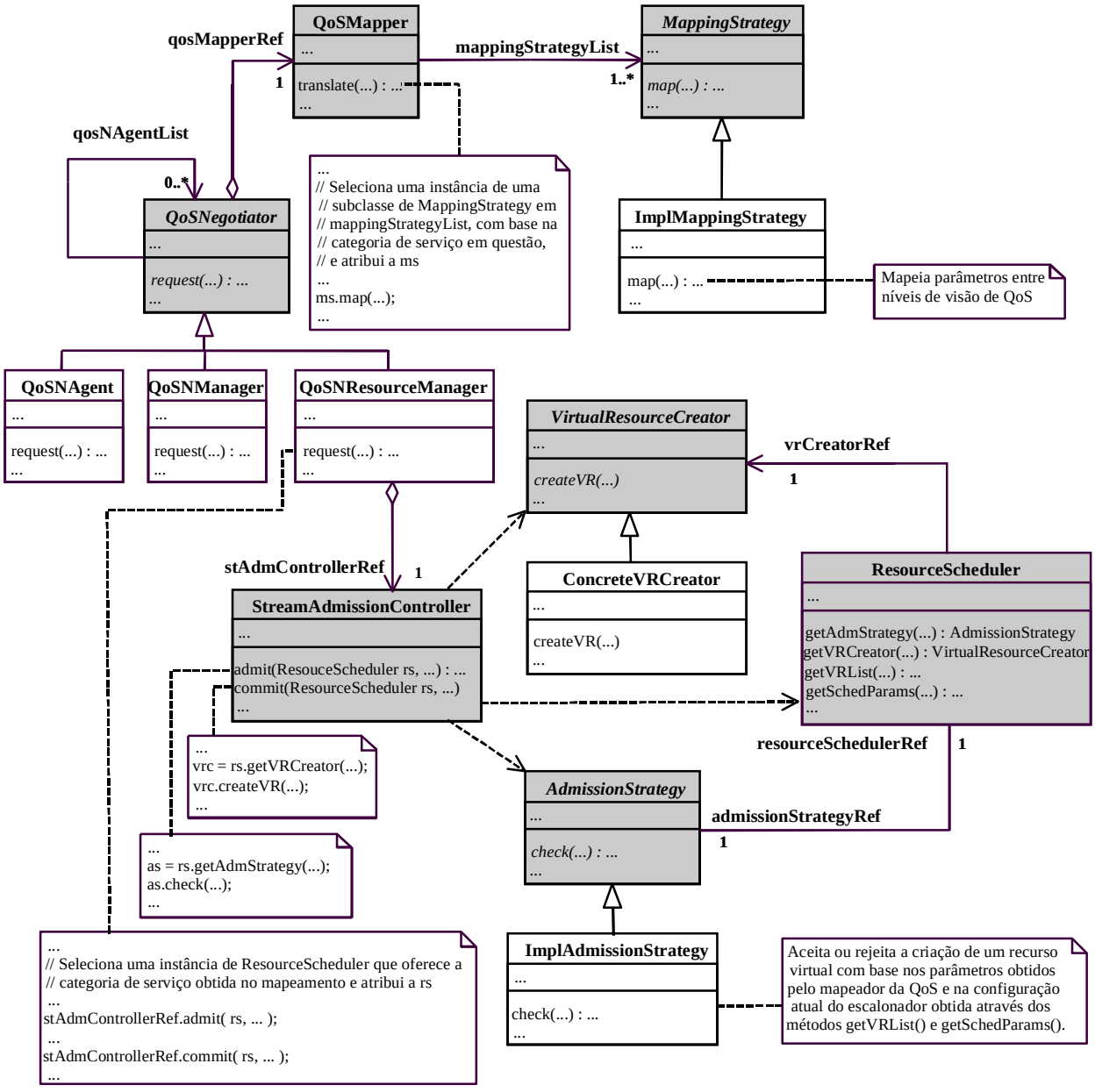


Figura 37: Framework para Negociação da QoS.

Independente do modelo adotado, parte da tarefa de integração é delegada a instâncias da classe `QoSMapper`. Esta classe modela os *mapeadores da QoS*

que, em conjunto, definem os mecanismos de mapeamento. Cabe a estes componentes traduzir as requisições de serviço feitas junto aos gerentes de negociação atuantes no nível mais alto de abstração de um fornecedor em requisições de serviço condizentes com os níveis de visão de QoS dos subsistemas envolvidos no fornecimento do serviço. Internamente a estes subsistemas, as requisições de serviço a eles direcionadas podem sofrer o mesmo processo de tradução. Para que isto seja possível, a cada instância de uma subclasse de *QoSNegotiator* está agregada, através do relacionamento `qosMapperRef`, uma instância de `QoSMapper`, independente do nível de abstração em que estejam atuando.

O processo de tradução de requisições de serviço é efetuado mediante o disparo do método `translate()` em um mapeador da QoS. A função deste método consiste basicamente em mapear parâmetros de caracterização de carga e de especificação da QoS entre níveis distintos de visão de QoS. Para possibilitar a flexibilidade dos mapeadores de QoS quanto às categorias de serviço oferecidas por um fornecedor, nos seus vários níveis de abstração, o mapeamento de parâmetros é delegado a instâncias de subclasses da classe abstrata *MappingStrategy*, através do disparo do método `map()`. A classe *MappingStrategy* representa abstratamente as estratégias de mapeamento, sendo as redefinições do método abstrato *map()* responsáveis por implementá-las concretamente em cada uma das subclasses de *MappingStrategy*. Uma variante do pattern Strategy é utilizada para definir o relacionamento de associação entre os mapeadores de QoS e os conjuntos de estratégias de mapeamento por eles usadas, simbolizado na Figura 37 por `mappingStrategyList`. A cada instância de uma subclasse de *MappingStrategy* presente em `mappingStrategyList`, corresponde uma estratégia de mapeamento de parâmetros de uma determinada categoria de serviço.

A implementação de gerentes de recursos que atuam durante a concatenação de fluxos também é simbolizada no Framework para Negociação através do relacionamento de agregação `stAdmControllerRef` entre as classes `QoSNResourceManager` e `StreamAdmissionController`. Esta última classe representa os componentes que controlam a admissão de novos recursos virtuais em subsistemas cujo nível de visão de QoS relaciona-se diretamente a recursos reais. Estes *controladores de admissão* definem, em conjunto, os mecanismos de controle de admissão de fluxos. Para cada árvore de recursos virtuais presente em um subsistema,

há um gerente de recursos e um controlador de admissão específico, responsáveis pela gerência da mesma.

O processo de validação da criação de recursos virtuais em uma árvore é representado pelo método `admit()` em `StreamAdmissionController`. O objetivo básico deste método consta da verificação de que é viável ao novo recurso virtual obter a parcela desejada de utilização de um recurso real, sem interferir em outros recursos virtuais já criados na árvore associada ao recurso. Este teste de viabilidade é delegado a uma instância de uma subclasse qualquer da classe abstrata *`AdmissionStrategy`*, através do disparo do método `check()`. A classe *`AdmissionStrategy`* representa abstratamente as estratégias de admissão, sendo as redefinições de *`check()`* responsáveis por implementá-las em cada uma de suas subclasses.

Uma outra variante do pattern Strategy é aplicada para definir o relacionamento entre os controladores de admissão e as possíveis estratégias de admissão a serem usadas. Conforme foi citado na Subseção 4.4.2, em cada uma das árvores envolvidas no fornecimento de um serviço requisitado pelo usuário, é escolhido um escalonador (com base na categoria de serviço associada à requisição) que será responsável por ceder ao novo recurso virtual parcelas de utilização do recurso real em questão. Portanto, é sobre a parcela de utilização do recurso real destinada ao escalonador que será feito o teste de viabilidade.

A interligação entre escalonadores e estratégias de admissão é representada no Framework de Negociação da QoS através dos relacionamentos de associação `admissionStrategyRef` e `resourceSchedulerRef` entre `ResourceScheduler` e *`AdmissionStrategy`*. Esta interligação é reforçada pelo fato de que ambas as estratégias de admissão e escalonamento objetivam também maximizar a utilização de recursos, logo as implementações destas estratégias com relação a uma determinada categoria de serviço estão intimamente ligadas. Para permitir a máxima flexibilidade dos controladores de admissão com relação às estratégias de admissão a serem utilizadas, o método `admit()` é dotado de um argumento que identifica a que escalonador está se tentando adicionar um novo recurso virtual. Graças a este argumento, um controlador de admissão pode ter acesso à estratégia de admissão desejada, por intermédio do método `getAdmStrategy()` presente em `ResourceScheduler`. Já os métodos `getVRList()` e `getSchedParams()`, definidos nesta mesma classe, permitem que as estratégias de admissão tenham acesso à

estrutura atual de escalonamento dos escalonadores, possibilitando a execução dos testes de viabilidade de maneira correta. A seqüência de invocações de métodos ilustrada na Figura 38 exemplifica todo o processo de admissão e criação de recursos virtuais coordenado por um gerente de recursos.

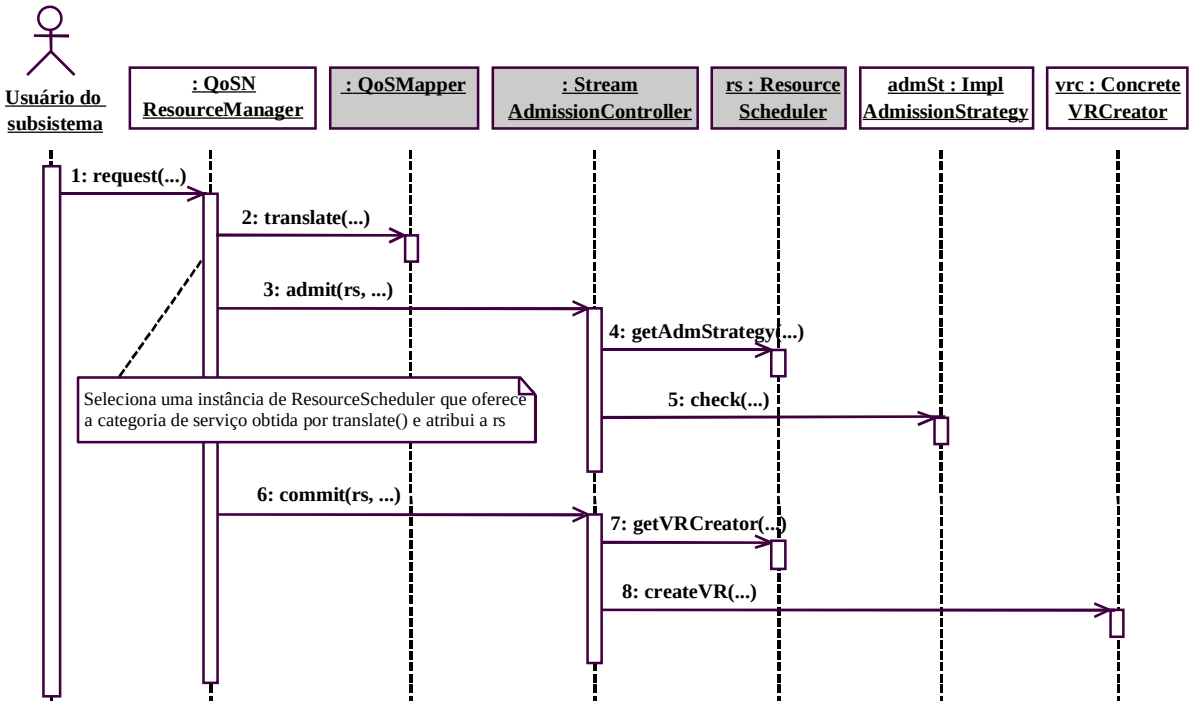


Figura 38: seqüência de admissão e criação de recursos virtuais.

Como mostra a Figura 38, o método `commit()` em `StreamAdmissionController` complementa o método `admit()`. Ele é o responsável por iniciar o processo de criação de recursos virtuais, através do disparo do método `createVR()` no componente de criação ligado ao escalonador escolhido durante a admissão. Esta ligação é representada na Figura 37 pelo relacionamento de associação `vrCreatorRef` entre `ResourceScheduler` e *`VirtualResourceCreator`*. O método `commit()` também é dotado de um argumento que identifica o escalonador em questão, para que os controladores de admissão possam ter acesso aos componentes de criação, por intermédio do método `getVRCreator()` presente em `ResourceScheduler`.

4.5.6 Exemplo de Aplicação do Framework para Negociação

A Figura 39 apresenta um caso de uso de uma variação do Framework para Negociação da QoS, onde é modelada uma arquitetura para protocolos de configuração de fluxos fim-a-fim, cujas requisições de criação são feitas pelas

aplicações de origem dos mesmos (analogamente ao ST-II – Subseção 3.2.3). Na figura (necessariamente "poluída"), é dado um exemplo de como esta arquitetura pode ser personalizada para a configuração específica de fluxos de quadros de vídeo. Vários detalhes da arquitetura foram omitidos, para não sobrecarregar ainda mais a figura.

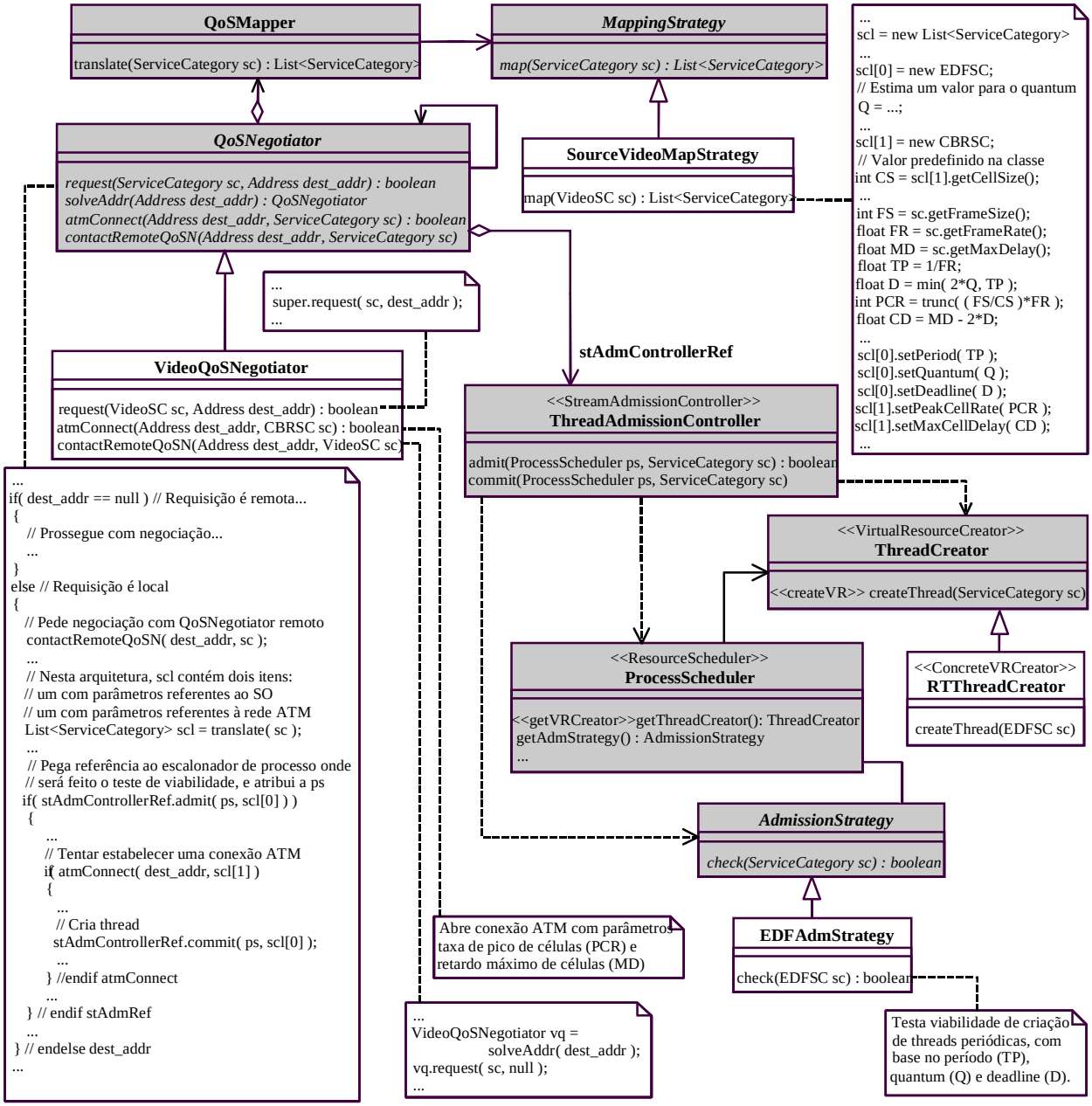


Figura 39: exemplo de aplicação do Framework para Negociação da QoS.

A Figura 40 apresenta uma hierarquia simplificada de categorias de serviço (baseada no Framework para Parametrização de Serviços) própria para o exemplo do protocolo de configuração de fluxos de vídeo.

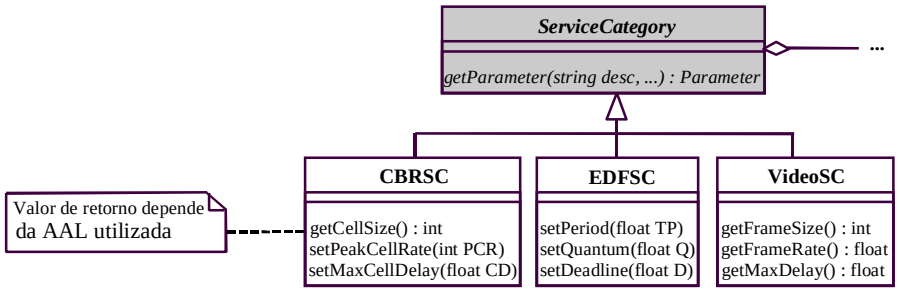


Figura 40: categorias de serviço usadas na configuração de fluxos de vídeo.

O ambiente hipotético (vide Figura 41) sobre o qual a arquitetura é definida constitui-se em:

- Uma rede ATM dando suporte à transmissão e recepção de dados com garantias de QoS;

- Uma plataforma para a comunicação entre objetos remotos (baseada, por exemplo, na arquitetura CORBA); e

- Estações finais controladas por sistemas operacionais cujo mecanismo de escalonamento de threads baseia-se na arquitetura apresentada na Subseção 4.4.4. Cada uma dessas estações é também dotada de AALs que permitem o estabelecimento de conexões ATM com outras estações, e a transmissão e recepção de dados por intermédio dessas conexões[33]. Pressupõe-se que a comunicação entre threads e a AAL de transmissão se dá assincronamente, através de buffers implementados sobre memória compartilhada, permitindo que as threads não sejam bloqueadas por causa dessa comunicação.

Como mostra a Figura 41, cada uma das entidades de um protocolo de configuração de fluxos (que tenha sido modelado segundo a arquitetura proposta nesta subseção) deve ser dotada de dois componentes básicos: um *mapeador da QoS* (representado por `QoSMapper`) e um *negociador da QoS* (implementado por uma subclasse de *`QoSNegotiator`*). Na arquitetura proposta nesta subseção, o esquema de configuração de fluxos adotado baseou-se na abordagem de intermediação apresentada na Subseção 3.4.2. Assim, os negociadores da QoS são equivalentes aos QoS Brokers, no sentido que eles são responsáveis também pela gerência de recursos das estações

finais, bem como pela comunicação com a interface de gerência de recursos provida pela AAL de sinalização. Tanto a Figura 39 quanto a Figura 41 concentram-se na parte da configuração de fluxos referente às estações onde se originam as requisições.

Quando uma aplicação requisita a criação de um fluxo com uma outra aplicação remota[34], há, em primeiro lugar, uma negociação de alto nível entre as entidades pares do protocolo, que verifica, basicamente, se o usuário da aplicação remota concorda com a especificação do fluxo definida pelo usuário da aplicação origem[35]. Para que o negociador presente na estação onde foi originada a requisição entre em contato com o negociador remoto correto, o método `request()` em *`QoSNegotiator`* é dotado do argumento `dest_addr`, que endereça a aplicação remota. Este argumento permite ao método *`contactRemoteQoSN`*`()` (redefinido nas subclasses de *`QoSNegotiator`*) disparar o método `request()` no negociador remoto. Para tanto, `contactRemoteQoSN()` vale-se do método `solveAddr()` para obter uma referência à instância remota da subclasse de *`QoSNegotiator`*, que implementa o negociador remoto. Para obter essa referência, `solveAddr()` pode utilizar, por exemplo, os serviços de *trading* oferecidos pela maioria das plataformas para comunicação entre objetos remotos. No exemplo de personalização da arquitetura apresentado na Figura 39, é definida a classe `VideoQoSNegotiator` (derivada de *`QoSNegotiator`*), que implementa o negociador da QoS para o protocolo de configuração de fluxos de vídeo. Esta classe redefine, além de `contactRemoteQoSN()`, o método `request()`, para permitir que somente requisições para criação de fluxos de vídeo (representadas na Figura 40 pela classe `VideoSC`) sejam aceitas por `VideoQoSNegotiator`.

---

[33] Por questões de simplificação, o exemplo apresentado não faz nenhuma consideração a respeito de que tipos de AAL de transmissão são adequadas ao fornecimento de que tipos de serviço.
[34] Assumimos, no exemplo, somente a possibilidade de criação de fluxos ponto-a-ponto.
[35] Opcionalmente, esta negociação pode verificar, de antemão, se há recursos suficientes na estação onde a aplicação remota se localiza.

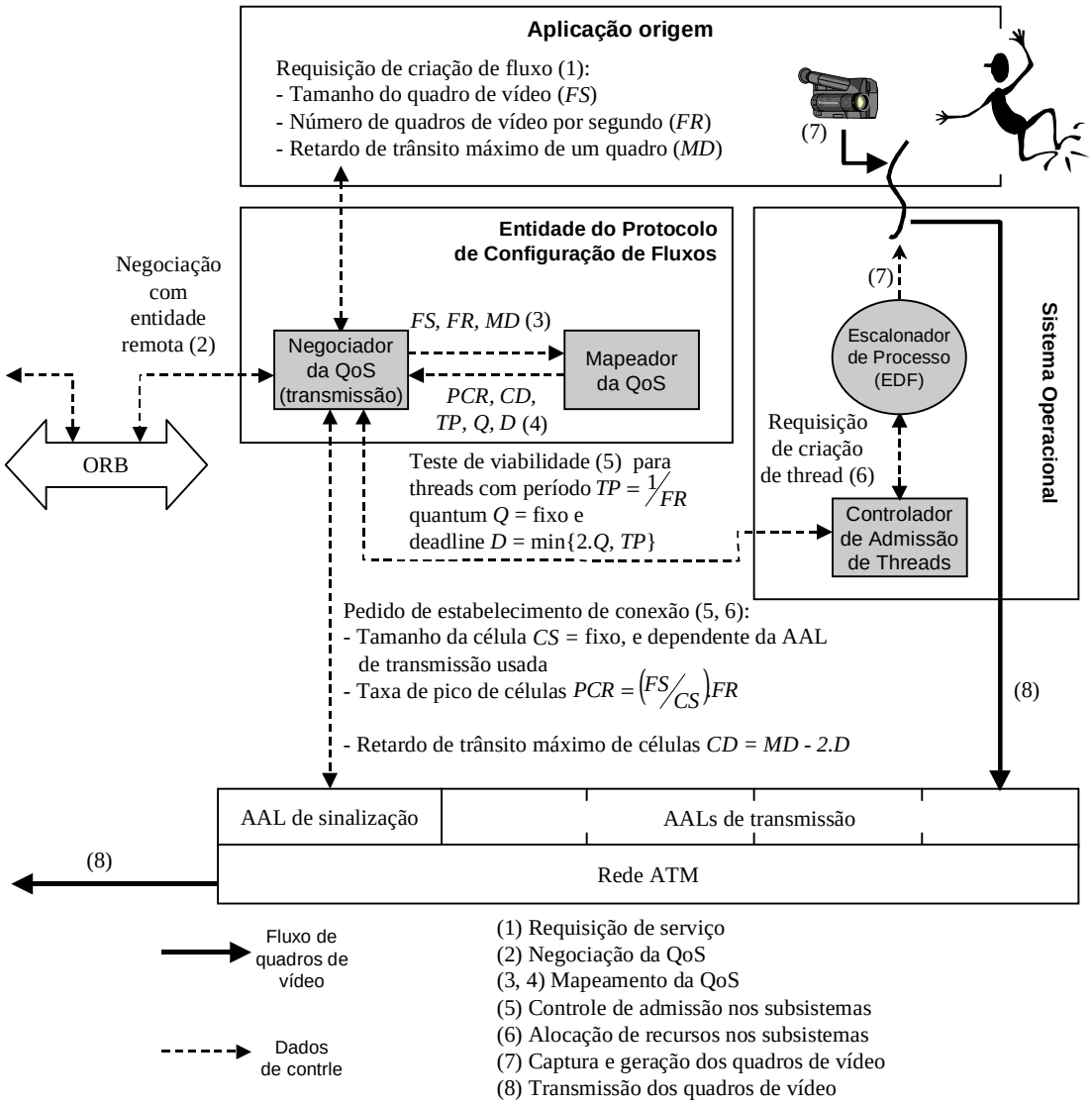


Figura 41: protocolo de configuração de fluxos de vídeo.

Voltando ao esquema de configuração de fluxos modelado pela arquitetura, a etapa de negociação é seguida do acionamento do mapeador da QoS (através do método `translate()`), que responde pela tradução dos parâmetros informados durante a requisição de criação de um fluxo em parâmetros condizentes com os níveis de visão de QoS do sistema operacional e da rede ATM. A classe `SourceVideoMapStrategy` é a responsável por implementar a estratégia de mapeamento relacionada ao protocolo de configuração de fluxos de vídeo[36]. A partir de instâncias de `VideoSC` (obtidas durante as requisições das aplicações), o método `map()`

[36] Nas estações de destino, o mapeamento pode ser diferente, não envolvendo, por exemplo, conexões ATM, uma vez que os pedidos de estabelecimento das mesmas provêm das estações de origem. Daí o nome **SourceVideoMapStrategy** dado à estratégia de mapeamento utilizada na origem.

em `SourceVideoMapStrategy` cria instâncias das classes `EDFSC`[37] e `CBRSC`[38], que representam, respectivamente, requisições de criação de threads periódicas, e pedidos de estabelecimento de conexões ATM com vazão e retardo de trânsito garantidos. Alguns detalhes sobre este mapeamento são ilustrados na Figura 39 e na Figura 41.

Após o mapeamento de uma requisição, pode ser necessária a execução, no sistema operacional, de um teste que valide a admissão de uma nova thread, antes da criação efetiva da mesma. Para que seja possível este controle de admissão, é adicionado à arquitetura de escalonamento apresentada na Subseção 4.4.4 um *controlador de admissão de threads*, representado pela classe `ThreadAdmissionController`. O método `admit()`, definido nesta classe, retorna um valor booleano, que indica se a thread pode ser admitida no escalonador de processo passado como argumento do método[39]. Na Figura 41, é definida somente a estratégia de admissão associada ao EDF (classe `EDFAdmStrategy`), visto que, para o exemplo do protocolo de configuração de fluxos de vídeo, só interessa a admissão de threads periódicas. O teste de viabilidade de uma thread periódica é feito com base na instância de `EDFSC` obtida a partir do mapeamento.

Já com relação à rede ATM, são utilizadas as suas funções nativas de controle de admissão e de alocação de recursos, acessíveis por intermédio da AAL de sinalização. O negociador da QoS se comunica com esta AAL através do método *`atmConnect()`*, redefinido nas subclasses de *`QoSNegotiator`*. Este método é o responsável por pedir à AAL de sinalização o estabelecimento de uma conexão ATM com uma estação remota, endereçada pelo argumento `dest_addr`. No exemplo do protocolo de configuração de fluxos de vídeo, `atmConnect()` é implementado em `VideoQoSNegotiator` de forma que, ao ser disparado, é feito um pedido de estabelecimento de uma conexão ATM com vazão e retardo de trânsito garantidos, com base na instância de `CBRSC` obtida a partir do mapeamento.

[37] O valor do parâmetro quantum depende da implementação do código executado pela thread periódica. No exemplo, é assumido que a estratégia de mapeamento tem como estimar esse valor.
[38] A classe `CBRSC` não corresponde exatamente a uma especificação de contrato de tráfego normalmente encontrada em redes ATM. O uso dos parâmetros definidos nesta classe visou somente simplificar o exemplo.
[39] O procedimento pelo qual o negociador da QoS obtém uma referência ao escalonador de processo apropriado não é apresentado.

4.5.7 Componentes do Framework para Sintonização da QoS

As classes e os respectivos relacionamentos que constituem o Framework para Sintonização da QoS são ilustrados na Figura 42. A classe *QoSTuner* simboliza abstratamente os componentes que, em conjunto, implementam os mecanismos de sintonização da QoS. O pattern Facade é aplicado junto a *QoSTuner*, de modo a prover uma interface única de indicação de alertas aos usuários, representada pelo método abstrato *alert()*. Funcionalmente, este método não representa, qualquer que seja o caso de uso do Framework para Sintonização, uma interface diretamente acessível ao usuário. Ao invés disso, ele permite que sejam implementados, nas subclasses derivadas de *QoSTuner*, mecanismos orientados a eventos assíncronos (como por exemplo, interrupções, *upcalls* [COULSON93], e *flags de indicação*), que permitam ao usuário receber os alertas gerados pelo mecanismo de monitorização. Em determinados casos de uso do Framework para Sintonização, o método *alert()* pode ser dotado de argumentos que descrevam, através de parâmetros de desempenho do fornecedor, o comportamento do fluxo responsável pela geração do alerta, para que este comportamento possa ser informado ao usuário.

Outro pattern utilizado para representar a estrutura de balanceamento de fluxos é o Chain of Responsibility, simbolizado pelo relacionamento recursivo de associação `qosTAgentList`, definido em *QoSTuner*. Identicamente ao que ocorre com o relacionamento `qosNAgentList`, definido em *QoSNegotiator* (Subseção 4.5.5), o relacionamento `qosTAgentList` permite a aplicação recursiva do Framework para Sintonização em vários níveis de abstração, bem como a independência do mesmo com relação aos modelos de integração utilizados nos diferentes níveis.

A estrutura resultante da combinação entre os patterns Facade e Chain of Responsibility permite que ambos os modelos de integração possam ser usados de maneira transparente para os usuários de um fornecedor. Qualquer que seja o nível de abstração em questão, o modelo de integração usado será determinado através da redefinição do método abstrato *tune()* nas subclasses derivadas de *QoSTuner*. Desta forma, subclasses de *QoSTuner* que modelam gerentes de sintonização da QoS centralizados (representados na figura por `QoSTManager`) devem redefinir `tune()`, de maneira que, à face de uma potencial violação de contrato detectada por um de seus agentes de sintonização (representados por `QoSTAgent`), estes gerentes saibam

redistribuir a responsabilidade pela manutenção de contratos de serviço entre os outros agentes. Subclasses que modelam agentes de sintonização parceiros devem redefinir `tune()`, de modo que estes componentes saibam coordenar *entre si* a redistribuição dessa responsabilidade. Quando os agentes modelados são responsáveis também pela gerência direta de recursos, o método `tune()` deve ser redefinido, de maneira a permitir a estes gerentes de recursos (representados por `QoSTResourceManager`) coordenar os escalonadores presentes nas diferentes árvores de recursos virtuais que estão sob sua responsabilidade.

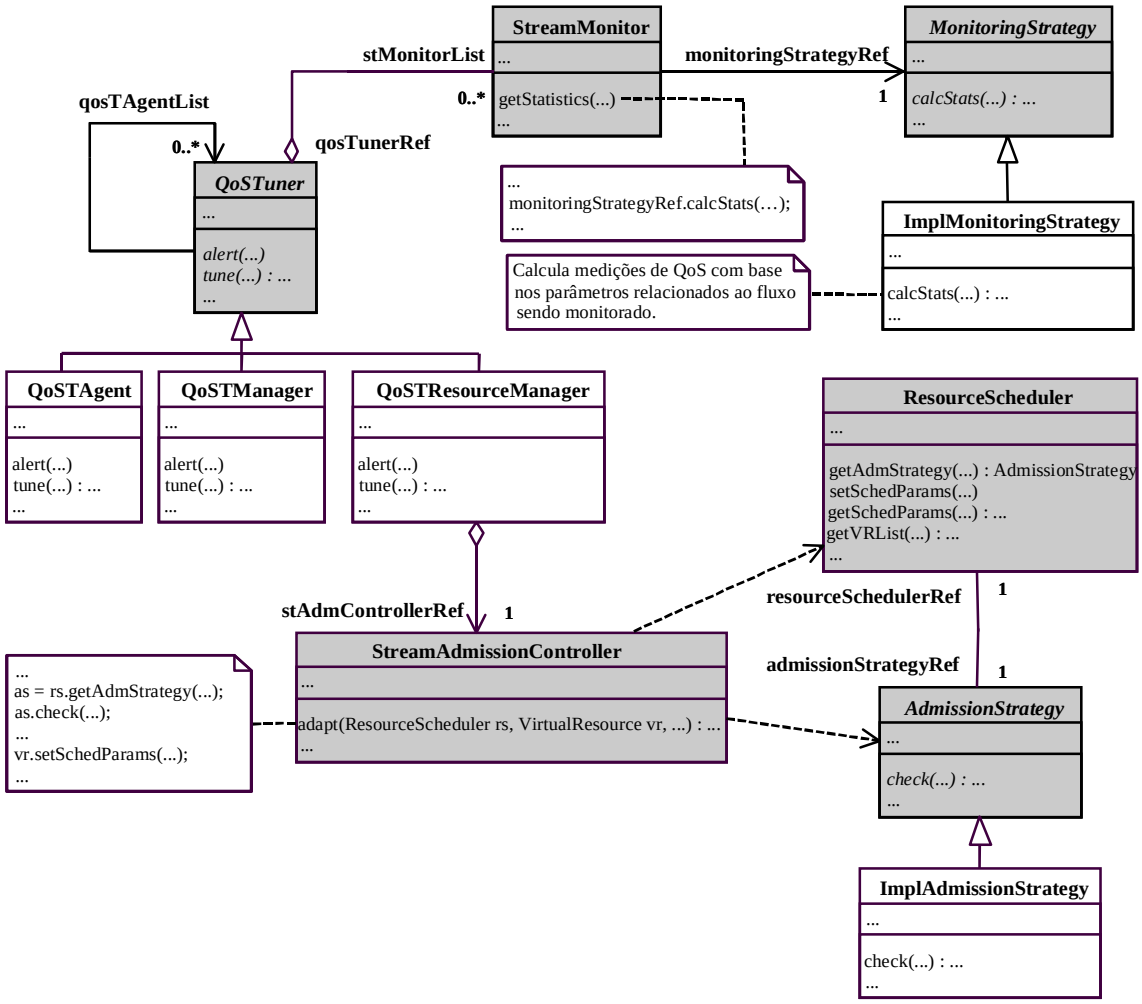


Figura 42: Framework para Sintonização da QoS.

Independente do modelo adotado, a tarefa de integração é sempre iniciada a partir da geração de um evento qualquer no ambiente que ocasione o disparo do método `getStatistics()` em alguma instância da classe `StreamMonitor`, que simboliza os *monitores de fluxos*. Estes componentes, em conjunto, definem os

mecanismos de monitorização. O método `getStatistics()` é responsável por obter as medições relativas à real QoS sendo oferecida aos usuários, e por enviar alertas ao seu gerente (ou agente) de sintonização, através do disparo do método `tune()` na instância correspondente de uma subclasse de *`QoSTuner`*[40]. A interação entre os mecanismos de sintonização e de monitorização é simbolizada no Framework para Sintonização através dos relacionamentos `stMonitorList` e `qosTunerRef` entre as classes *`QoSTuner`* e `StreamMonitor`. Para que seja possível a monitorização individual de um fluxo em cada um dos subsistemas envolvidos no fornecimento do serviço, a cada instância de uma subclasse de *`QoSTuner`* está associada, através de `stMonitorList`, uma instância de `StreamMonitor`, para cada fluxo conhecido nesse subsistema.

O cálculo, em si, das medições de QoS de um fluxo, é delegado a uma instância de uma subclasse qualquer da classe abstrata *`MonitoringStrategy`*, através do disparo do método `calcStats()`. A classe *`MonitoringStrategy`* representa abstratamente as estratégias de monitorização, sendo as redefinições do método abstrato *`calcStats()`* responsáveis por implementá-las concretamente em cada uma das subclasses de *`MonitoringStrategy`*. O pattern Strategy é utilizado para definir o relacionamento de associação (representado na Figura 42 por `monitoringStrategyRef`) entre os monitores de fluxo e as estratégias de monitorização correntemente adotadas por eles. Isto permite uma maior flexibilidade dos mecanismos de monitorização, no que diz respeito à forma como os parâmetros de caracterização de serviços utilizados durante as medições estão estruturados e distribuídos pelo ambiente.

A implementação de gerentes de recursos que atuam durante o balanceamento de fluxos também é viabilizada no Framework para Sintonização, por intermédio do relacionamento de agregação `stAdmControllerRef` entre as classes `QoSTResourceManager` e `StreamAdmissionController`. Assim como no Framework para Negociação, para cada árvore de recursos virtuais presente em um subsistema, há um gerente de recursos e um controlador de admissão específico, responsáveis pela gerência da mesma.

---

[40] Ao invés do disparo de `tune()`, poderia ser utilizado um esquema assíncrono, como por exemplo, através de flags de indicação periodicamente lidos pelas instâncias das subclasses de *`QoSTuner`*. Embora o primeiro esquema seja mais simples, na medida em que só é necessária a captura de eventos no

Através do método `adapt()` presente em `StreamAdmissionController`, o mecanismo de sintonização pode tentar manter a QoS associada a um fluxo cujo contrato de serviço tenha sido (ou esteja na iminência de ser) violado. O método `adapt()` permite o redimensionamento da parcela de utilização de um recurso delegada a um recurso virtual. Para isto, é necessário que seja feito, previamente, um teste de viabilidade. O processo de redimensionamento pode utilizar, basicamente, as mesmas estratégias de admissão (classe *`AdmissionStrategy`*) adotadas durante a concatenação de fluxos, com a diferença de que o recurso virtual já existe: o que se deseja é alterar os seus parâmetros de escalonamento. Para manter a flexibilidade dos controladores de admissão com relação às estratégias de admissão a serem utilizadas, o método `adapt()` é dotado de argumentos que identificam o recurso virtual em questão, bem como o escalonador ao qual este recurso virtual está ligado. Esta estrutura corresponde à mesma variante do pattern Strategy utilizada na Subseção 4.5.5.

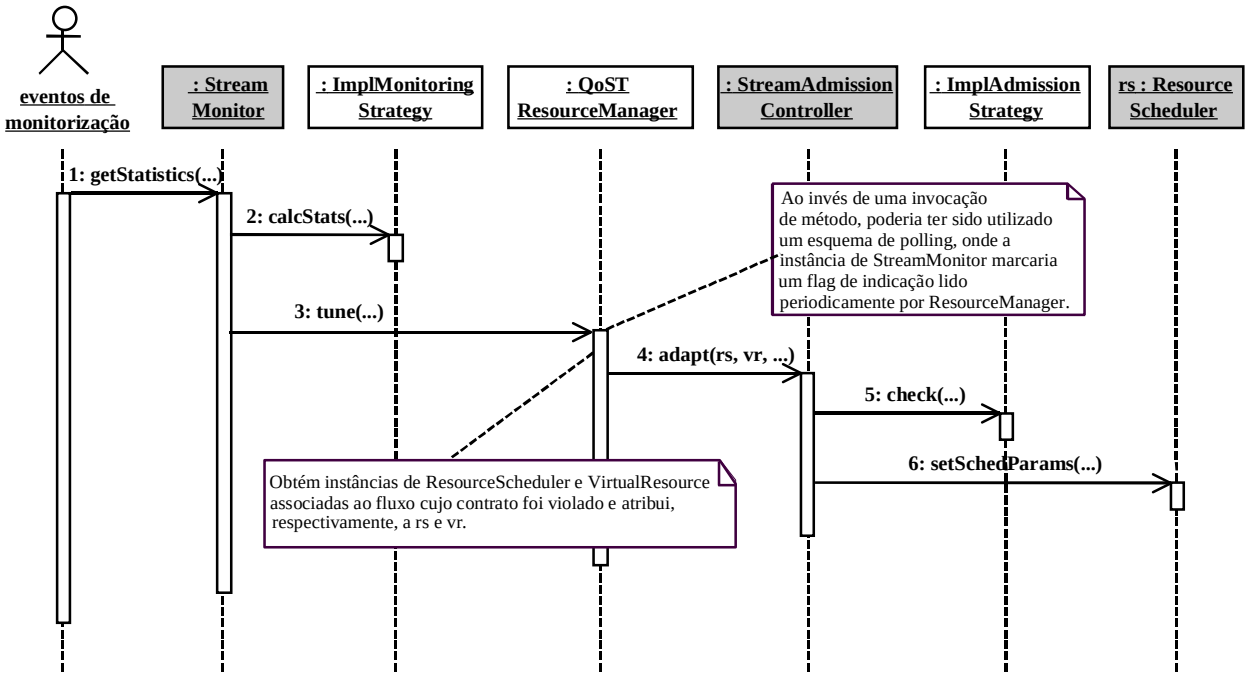


Figura 43: seqüência de redimensionamento de recursos virtuais.

A alteração dos parâmetros de escalonamento de um recurso virtual, que se quer redimensionar, é feita através do método `setSchedParams()` presente em `ResourceScheduler`. A seqüência de invocações de métodos da Figura 43

---

ambiente por parte das instâncias de `StreamMonitor`, o Pattern para Sintonização não faz nenhuma restrição quanto ao uso de esquemas assíncronos.

exemplifica todo o processo de redimensionamento coordenado por um gerente de recursos, partindo de uma violação de contrato detectada por um de seus monitores de fluxo.

4.5.8 Exemplo de Aplicação do Framework para Sintonização

A Figura 44 ilustra um caso de uso do Framework para Sintonização da QoS, onde é modelada uma arquitetura para protocolos de manutenção da QoS fim-a-fim de fluxos, criados segundo a arquitetura apresentada na Subseção 4.5.6. Da mesma forma, a figura dá um exemplo de como a arquitetura para protocolos de manutenção da QoS pode ser personalizada especificamente para fluxos de quadros de vídeo.

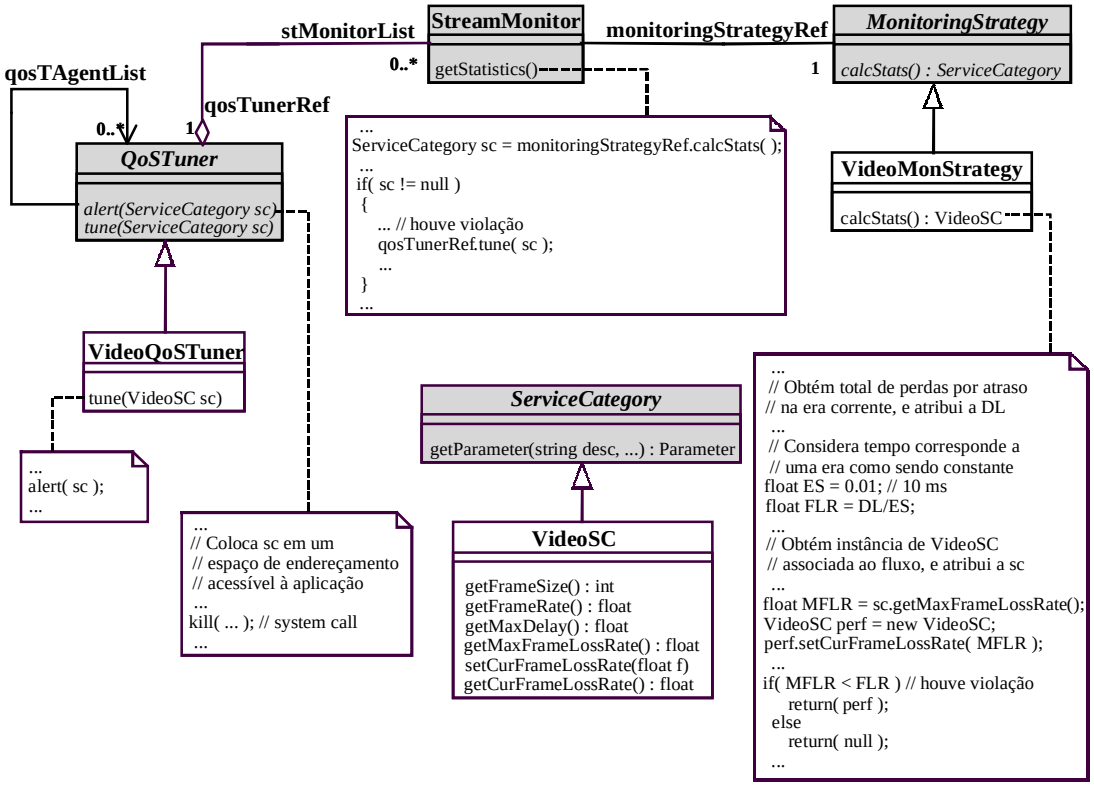


Figura 44: exemplo de aplicação do Pattern Para Sintonização da QoS.

O ambiente hipotético (vide Figura 45) sobre o qual a arquitetura é definida é basicamente o mesmo que o apresentado na Subseção 4.5.6, com algumas diferenças. Uma delas refere-se às garantias de QoS providas pela rede ATM. No exemplo da configuração de fluxos de vídeo, considerou-se que a rede ATM seria capaz de prover um serviço de comunicação com garantias no retardo de trânsito entre

estações, o que permitiria também garantias no retardo de trânsito fim-a-fim dos quadros de vídeo entre os usuários (incluindo o retardo decorrente da captura, geração e apresentação dos quadros de vídeo pelas threads das aplicações).

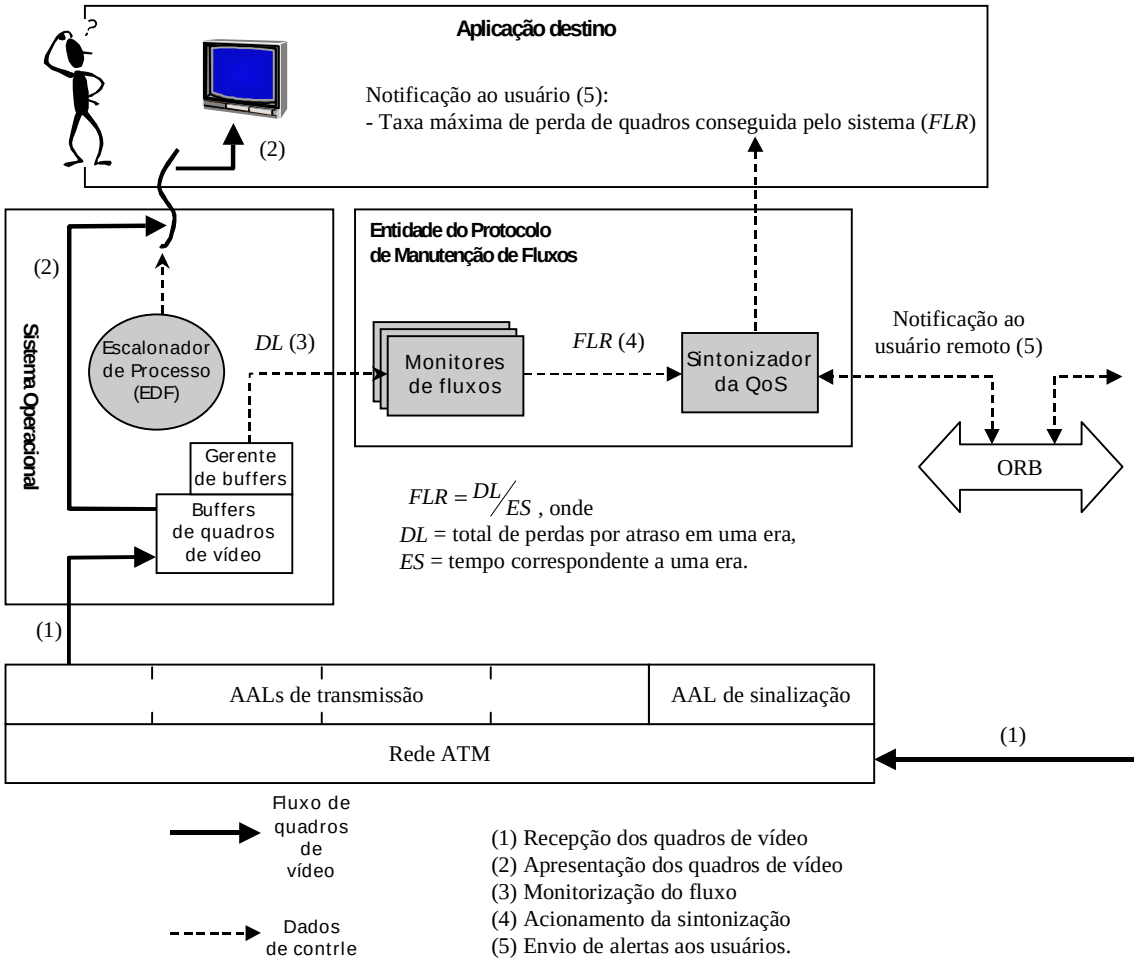


Figura 45: protocolo de manutenção de fluxos de vídeo.

Em tese, a afirmação feita no parágrafo anterior é verdadeira, visto que, segundo o ITU-T [ITU.T.Q.2931], o parâmetro retardo de trânsito deve estar disponível aos usuários da rede ATM através da AAL de sinalização. Na prática, porém, a maioria dos equipamentos existentes para redes ATM (interfaces, comutadores, etc.) seguem a especificação Q.39B [ATM93], que não oferece este parâmetro. Para os propósitos desta subseção, consideraremos que a AAL de sinalização oferece este parâmetro, mas o nível de serviço associado a ele corresponde ao de melhor esforço, ou seja, não é dada nenhuma garantia de que o retardo máximo especificado será respeitado pela rede ATM. Graças a essa ausência de garantia, torna-se difícil para um protocolo de manutenção de fluxos de vídeo restringir as variações no retardo de trânsito dos quadros de vídeo entre os usuários. E essas variações são especialmente críticas em serviços

conversacionais. Nesse tipo de serviço, quadros atrasados são, em geral, pouco úteis para reconstituir corretamente na estação de destino o sinal de vídeo capturado na estação de origem. Na maioria dos casos, estes quadros atrasados são descartados. Sob o ponto de vista do usuário, é interessante que a taxa máxima tolerável de perda de quadros por atraso possa ser especificada[41]. No exemplo desta subseção, este parâmetro é incluído na categoria de serviço representada pela classe `VideoSC` (apresentada na Subseção 4.5.6), como mostra a Figura 44 .

A arquitetura apresentada nesta subseção modela protocolos de manutenção que atuam, basicamente, nas estações finais. No exemplo do protocolo de manutenção de fluxos de vídeo, o protocolo deve se responsabilizar por tentar manter o retardo de trânsito de quadros de vídeo abaixo do valor máximo estipulado pelo usuário, possivelmente descartando quadros que cheguem atrasados. Como mostra a Figura 45, cada uma das entidades de um protocolo de manutenção de fluxos (que tenha sido modelado pela arquitetura proposta nesta subseção) deve ser dotada dos seguintes componentes: *monitores de fluxos* (representados por `StreamMonitor`) e um *sintonizador da QoS* (implementado por uma subclasse de *`QoSTuner`*). Cada fluxo mantido pelo protocolo é controlado por um monitor individual. O esquema de manutenção adotado na arquitetura age como se segue. Os monitores presentes nas estações de destino dos fluxos são periodicamente acionados por algum evento no ambiente, através do disparo do método `getStatistics()`, fazendo com que esses monitores recalculem os parâmetros de desempenho associados aos seus fluxos e, em caso de uma possível violação de contrato, acionem o sintonizador da QoS, através do método `tune()`[42].

A classe `VideoMonStrategy` é a responsável por implementar, através do método `calcStats()`, a estratégia de monitorização relacionada ao protocolo de manutenção de fluxos de vídeo. Toda vez que uma thread periódica, na estação de destino, tenta ler um quadro de vídeo (para apresentá-lo ao usuário), é verificado se houve algum atraso, seja pela ausência de quadros no buffer (relembrando que a leitura

[41] No exemplo, desprezou-se a possibilidade de ocorrência de erros durante a transmissão pela rede ATM.

é feita assincronamente, conforme citado na Subseção 4.5.6), ou pela presença, neste buffer, de quadros pertencentes a períodos anteriores da thread[43]. Em caso afirmativo, é incrementado o valor de um parâmetro de desempenho que indica o número de atrasos ocorridos no período atual de monitorização do fluxo. Para que não sejam confundidos os termos *período da thread* e *período de monitorização do fluxo*, chamaremos este último de *era* (conforme descrito em [NAHRSTEDT95]). Ao final de uma era, o monitor do fluxo obtém o valor deste parâmetro de desempenho (atribuindo-lhe, em seguida, o valor zero), divide-o pelo tempo correspondente a uma era (que pode ser variável de fluxo para fluxo), e compara o valor resultante com a taxa máxima tolerável de perda de quadros especificada pelo usuário. Caso esta taxa tenha sido ultrapassada, é detectada uma violação de contrato no fluxo de vídeo, e os sintonizadores da QoS (que, neste caso, são representados pela classe `VideoQoSTuner`) das estações de origem e de destino do fluxo são acionados.

Uma vez que não se tem garantias de retardo na rede ATM, e levando em conta o tipo de especificação de QoS provida aos usuários, só há uma ação cabível aos sintonizadores da QoS para fluxos de vídeo, que é informar aos usuários a ocorrência de violações do contrato, através do método `alert()`. Supondo que o sistema operacional utilizado seja da família UNIX, este método pode utilizar a system call `kill()`, tradicionalmente definida nestes sistemas, para iniciar um tratamento de exceção, que permita a uma aplicação informar ao usuário que a QoS negociada não está sendo plenamente cumprida.

4.5.9 Considerações a Respeito dos Frameworks para Orquestração de Recursos

A grande vantagem de se utilizar as arquiteturas propostas na Subseção 4.5.6 e na Subseção 4.5.8 decorre da flexibilidade intrínseca aos Frameworks para Orquestração de Recursos, o que possibilita a definição de diferentes protocolos de configuração e manutenção fim-a-fim de fluxos, através da derivação apropriada de

[42] Por motivo de simplificação, são omitidos na Figura 45 os detalhes de como é viabilizada a comunicação entre sintonizadores remotos. Esta comunicação se dá, em termos gerais, de maneira semelhante à comunicação entre negociadores remotos, detalhada na Subseção 4.5.6.

[43] Por motivo de simplificação, são omitidos os detalhes de como esta verificação pode ser feita. Na Figura 45, considerou-se que esta verificação é de responsabilidade de algum componente, no sistema operacional, que responde pela gerência dos buffers de quadros de vídeo. Este gerente não é modelado pela arquitetura proposta nesta subseção.

subclasses de *QoSNegotiator*, *QoSTuner,* *MappingStrategy,* *MonitoringStrategy* e *AdmissionStrategy*. Essa vantagem se torna mais evidente na medida em que o Framework para Parametrização de Serviços é utilizado para estruturar os parâmetros (relativos à QoS) presentes nestas arquiteturas.

Porém, associada ao alto grau de flexibilidade oferecido pelos Frameworks para Orquestração, há a complexidade em utilizá-los. Devido ao fato da concatenação e do balanceamento entre fluxos serem responsabilidades conjuntas dos mecanismos de negociação, sintonização, mapeamento, monitorização e admissão, a implementação de subclasses de *QoSNegotiator*, *QoSTuner*, *MappingStrategy*, *MonitoringStrategy* e *AdmissionStrategy* deve também ser conduzida de maneira conjunta. Propositadamente, os Frameworks para Orquestração de Recursos não entram em detalhes relativos à definição dessas subclasses, principalmente porque são inúmeras as formas de se implementar a concatenação e o balanceamento entre fluxos. Por outro lado, a ausência desses detalhes torna a aplicação desses frameworks bastante complexa. Portanto, seria interessante que houvesse uma estrutura que organizasse todo o processo de aplicação não só dos Frameworks para Orquestração, mas também de todos os outros elementos do Framework para Provisão de QoS, de forma a facilitar a tarefa dos projetistas. O presente trabalho delega a definição desta estrutura a trabalhos futuros.

4.6 Sumário

No presente capítulo, o Framework para Provisão de QoS foi descrito através de um conjunto de frameworks menores, que representam de maneira abstrata as estruturas de parametrização e os mecanismos de provisão de QoS componentes do modelo apresentado no Capítulo 2. Foi apresentado, para cada um desses frameworks, um exemplo de aplicação em um ambiente específico, dando-se ênfase à definição de arquiteturas configuráveis com relação à QoS. Procurou-se também relacionar essas arquiteturas, a fim de que pudesse ser ilustrado o relacionamento entre os componentes dos frameworks descritos. Em cada um dos exemplos apresentados, foram ressaltadas

vantagens, desvantagens, e implicações em se utilizar esses frameworks. Com base nas idéias surgidas a partir desses exemplos, o Capítulo 5 apresentará o projeto de implementação de um sistema real, também arquitetado com base nos frameworks apresentados neste capítulo.

Capítulo 5

Cenários de Uso do Framework

Este capítulo focaliza a aplicação do Framework para Provisão de QoS durante a implementação de um protótipo de um serviço de distribuição de vídeo sobre redes ATM. A implementação deste protótipo engaja-se no projeto de construção de um sistema de distribuição de informações, com suporte à gerência de grupos de usuários, à comunicação multicast e à configuração de QoS. A construção deste sistema visa validar alguns dos trabalhos em desenvolvimento junto ao projeto RAVel no Laboratório TeleMídia da PUC-Rio. A implementação do protótipo envolve, além do Framework para Provisão de QoS, a aplicação do Framework para Provisão de Serviço de Multicast apresentado em [RODRIGUES99]. Considerações a respeito deste framework serão citadas no decorrer deste capítulo.

A aplicação do Framework para Provisão de QoS na construção do sistema de distribuição de informações envolve tanto as estações finais (produtoras ou consumidoras de vídeo) quanto a rede ATM que dá suporte à comunicação. Nas estações finais, a idéia original consiste, basicamente, em definir um núcleo para o sistema operacional Linux, que dê suporte a serviços de processamento configuráveis com relação às categorias de serviço e às estratégias de escalonamento de processadores utilizadas. Porém, o desenvolvimento de um núcleo com estas características demanda um tempo muito grande de implementação. Uma vez que o desenvolvimento deste núcleo não é o objetivo principal do presente trabalho, optou-se pela aplicação do framework na implementação de um servidor de escalonamento sobre o sistema

operacional Solaris 2.5, a partir do qual as aplicações podem requisitar serviços de processamento com garantias estatísticas de QoS.

Já na rede ATM, a idéia original consiste em utilizar comutadores ATM abertos e programáveis pelos usuários[44], possibilitando o suporte a múltiplos serviços de comunicação explicitamente nesses comutadores. Até a concepção deste trabalho, não se dispunha, no Laboratório TeleMídia, de comutadores ATM com essas características. Por este motivo, foi desenvolvida uma plataforma que simula o funcionamento de uma rede ATM programável.

5.1 Arquitetura do Sistema de Distribuição de Informações

Um exemplo hipotético de um sistema de distribuição de informações pode ser visualizado na Figura 46. No que diz respeito à provisão de QoS, o objetivo principal do fornecedor do serviço de distribuição é tornar o conceito de QoS acessível às aplicações multimídia (sejam elas servidoras ou clientes de fluxos de distribuição), através de uma interface de configuração condizente com as suas visões de QoS. Em se tratando de um sistema de distribuição, dois conjuntos principais de primitivas de interface são necessários:

- *Primitivas dos servidores*: permitem aos servidores de informações criar e publicar fluxos de distribuição (*canais*).
- *Primitivas dos clientes*: permitem aos clientes obter informações sobre canais oferecidos no sistema, bem como assinar e selecionar canais específicos.

[44] Neste sentido, usuários podem ser gerentes da rede ou mesmo programadores das aplicações, dependendo do domínio administrativo em questão (rede pública, rede privada, etc.).

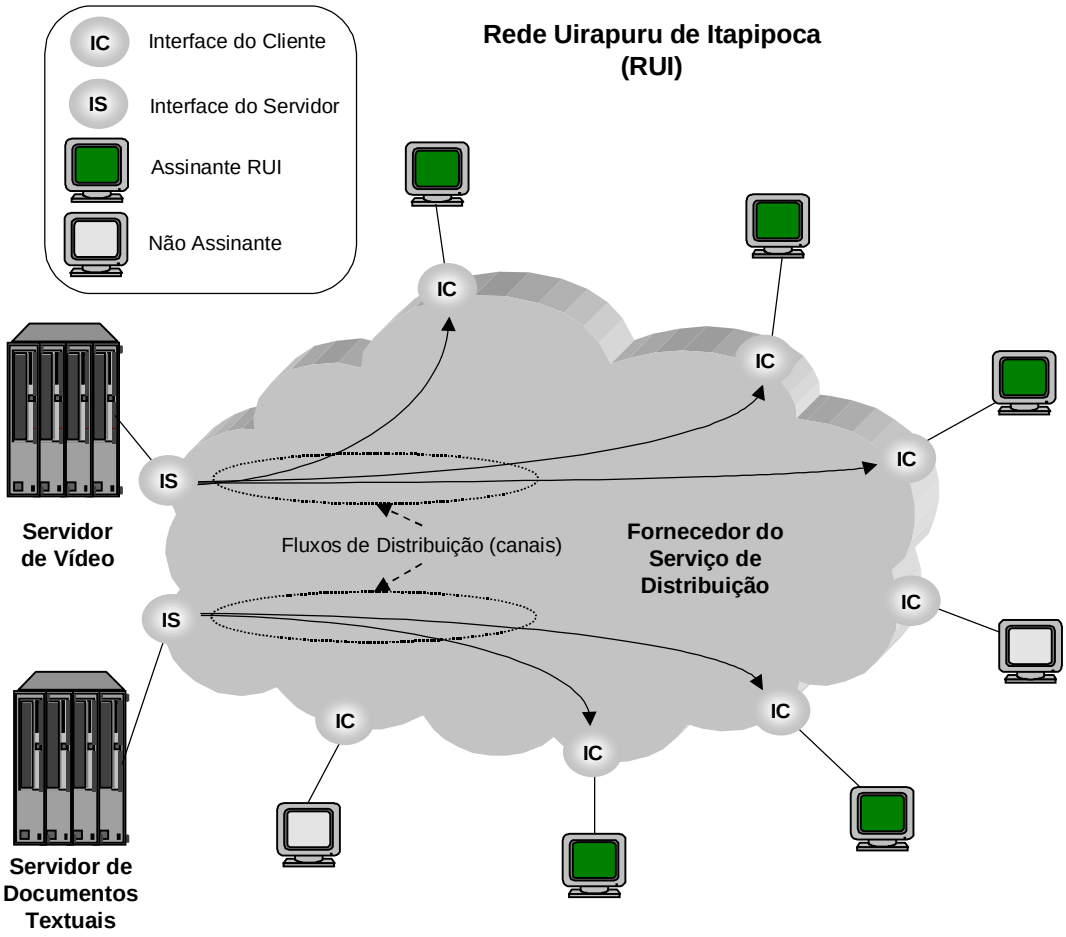


Figura 46: exemplo de um sistema de distribuição de informações.

A Figura 47 ilustra, esquematicamente, os elementos que compõem a arquitetura do sistema de distribuição de informações. O *gerente de serviços*, localizado em um nó qualquer da rede, funciona como uma base centralizada de informações de gerência, onde as interfaces de serviço dos clientes (*ICs*) e servidores (*ISs*), localizadas nas estações finais, podem publicar e obter informações sobre a configuração de grupos de usuários (através da *central de gerência de grupos e endereços - CGE*) e de canais (através da *central de gerência de canais - CGC*). Por intermédio do *gerente de negociação da QoS* (*GNQoS*), também presente junto ao gerente de serviços, é feita a negociação da QoS entre os clientes e os servidores.

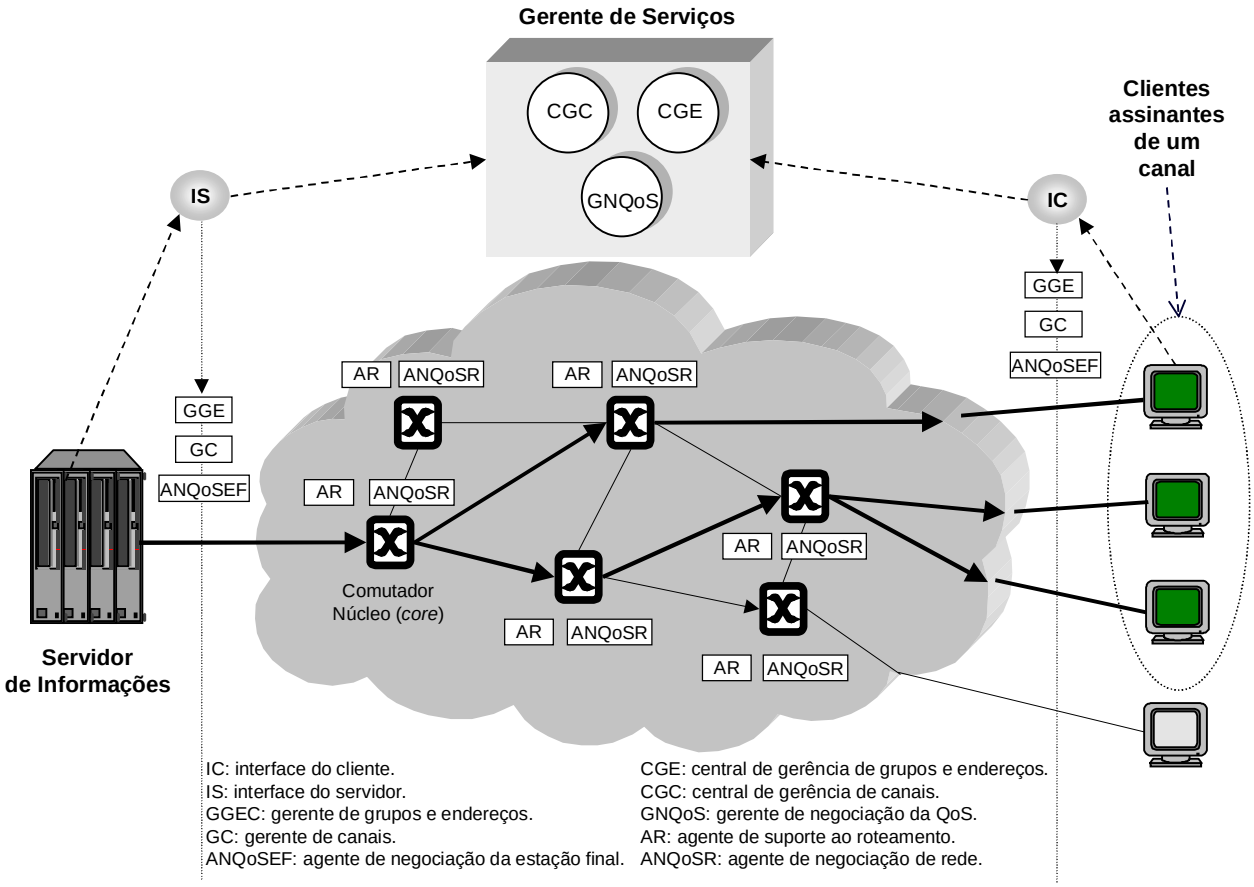


Figura 47: arquitetura do sistema de distribuição de informações.

Além de interfaces de serviço, cada estação final é dotada também de um *gerente de grupos e endereçamento* (*GGE*), um *gerente de canais* (*GC*) e *um agente de negociação da QoS da estação* (*ANQoSEF*). Os dois primeiros componentes atuam, basicamente, como caches de informações gerenciadas, respectivamente, pelo CGE e pelo CGC, enquanto o último é responsável por gerenciar os recursos da estação final, e por iniciar o processo de reserva e alocação de recursos na rede, que é executado por *agentes de negociação da QoS da rede* (*ANQoSR*), presentes em cada um dos comutadores ATM. O suporte ao roteamento nesta rede, em cada um de seus comutadores, é dado pelo *agente de roteamento* (*AR*).

Cada um dos componentes do sistema de distribuição, introduzidos até este ponto, estão simbolizados no diagrama de classes da Figura 48. As classes presentes neste diagrama, bem como os relacionamentos entre elas, são apresentadas nas subseções a seguir. As classes delimitadas pela área hachurada correspondem ao escopo de aplicação do Framework para Provisão de QoS.

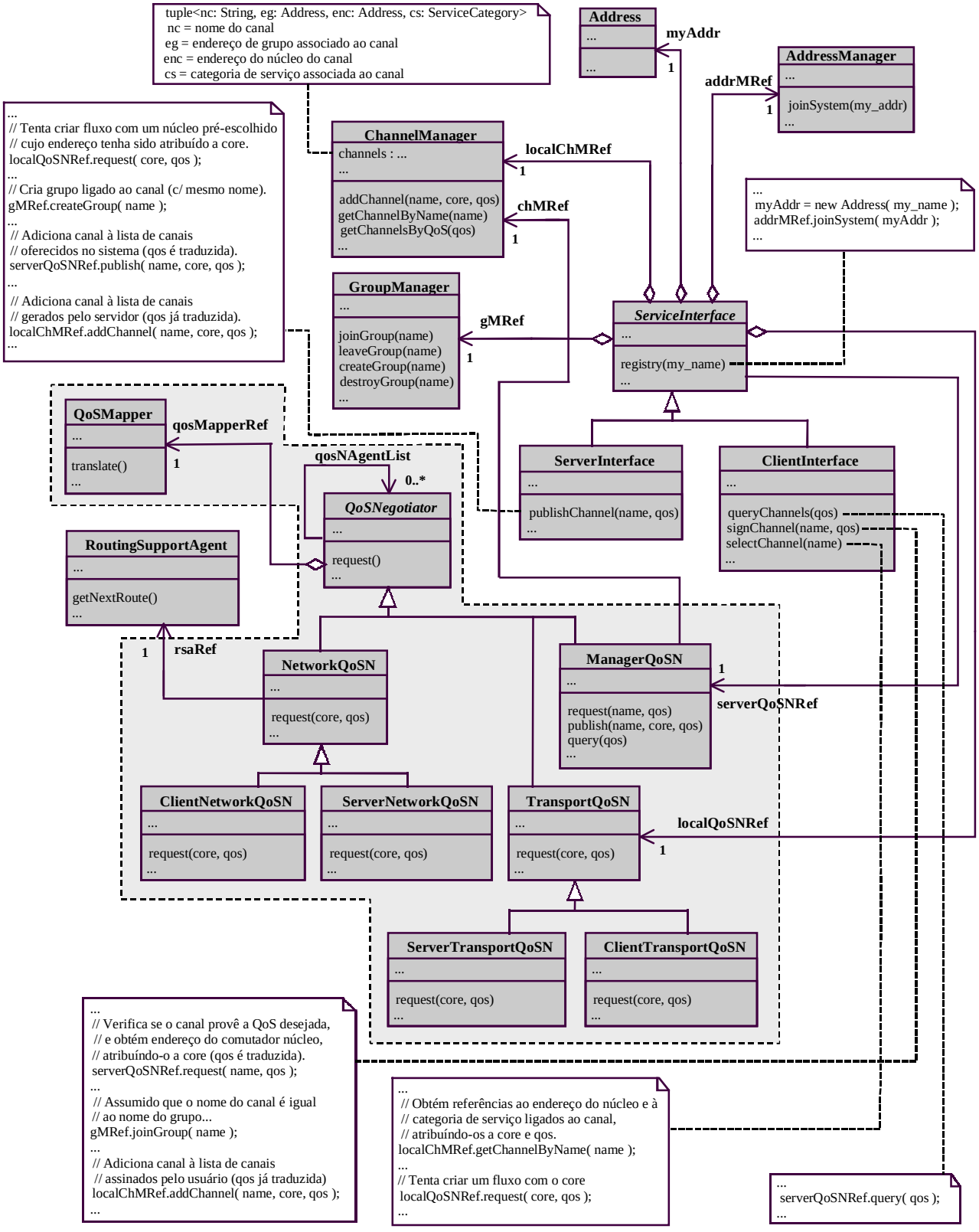


Figura 48: diagrama de classes do sistema de distribuição.

5.1.1 Primitivas de Interface

As classes `ClientInterface` e `ServerInterface` representam, respectivamente, as ICs e as ISs. A cada cliente ou servidor presente no sistema, está associada uma instância de uma dessas classes. A classe abstrata *`ServiceInterface`*, da qual as duas classes anteriores são especializadas, oferece o método `registry()`, que permite o ingresso de clientes e servidores no sistema de distribuição. Este método está relacionado à gerência de endereços do sistema, simbolizada na Figura 48 pelas classes `Address` e `AddressManager`.

A classe `ServerInterface` oferece aos servidores o método `publishChannel()`, que permite a publicação de novos canais no CGC presente no gerente de serviços, e a criação de grupos ligados a estes canais. Este método é responsável também pela configuração de fluxos entre os servidores e os *comutadores núcleos* (*core switches*) destes canais. A cada canal está associado um núcleo, responsável pelo repasse do fluxo de unidades de informação referente ao canal até os clientes. Detalhes de como fluxos são configurados entre servidores e núcleos, bem como vantagens e implicações desta abordagem em relação à criação de fluxos diretamente entre servidores e clientes, serão apresentados mais adiante.

A classe `ClientInterface` oferece aos clientes um conjunto de métodos. O método `queryChannels()` permite a um cliente consultar os canais, oferecidos pelo sistema de distribuição, que atendam a um determinado conjunto de restrições. No contexto do presente trabalho, a única restrição que pode ser definida é a da categoria de serviço[45] associada aos canais. O método `signChannel()` possibilita a assinatura de um canal, o que envolve a negociação entre a QoS desejada pelo cliente e a QoS oferecida pelo servidor do canal, e a posterior entrada do cliente assinante no grupo de usuários associado ao canal. E o método `selectChannel()` possibilita a um cliente requisitar a criação de um fluxo que leve até ele as informações transmitidas pelo servidor de um determinado canal assinado por ele (cliente). Este método envolve a configuração de um fluxo entre o núcleo do canal em questão e o cliente. Detalhes desta configuração também serão apresentados posteriormente neste capítulo.

[45] Por motivo de simplificação, foram omitidas as classes que modelam a estrutura de parametrização de serviços. Elas correspondem a uma aplicação do Framework para Parametrização de Serviços, apresentado na Seção 4.3.

5.1.2 Gerência de Canais, Grupos e Endereços

O CGC e os GCs são modelados, na Figura 48, pela classe `ChannelManager`. Nos servidores, instâncias desta classe atuam como GCs que armazenam informações sobre os canais oferecidos por estes servidores. Nos clientes, elas atuam como GCs que armazenam informações sobre os canais assinados por estes clientes. E no gerente de serviços, há uma única instância que atua como CGC, sendo responsável por armazenar informações sobre todos os canais oferecidos no sistema. As informações armazenadas por essas instâncias de `ChannelManager` correspondem a tuplas do tipo {`nc`, `eg`, `enc`, `cs`}, onde `nc` é o nome do canal, `eg` é o endereço de grupo associado ao canal, `enc` é o endereço do núcleo do canal, e `cs` é a categoria de serviço associada ao canal. O método `addChannel()` permite a inclusão de tuplas no conjunto gerenciado pela instância de `ChannelManager`. O método `getChannelByName()` permite a obtenção de uma tupla com base no nome do canal. E o método `getChannelsByQoS()` permite a obtenção de subconjuntos de tuplas com base na informação sobre a categoria de serviço associada aos canais.

Na Figura 48, o relacionamento de agregação `localChMRef`, entre as classes *`ServiceInterface`* e `ChannelManager`, representa a comunicação entre as interfaces e os GCs. Já os relacionamentos de associação `serverQoSNRef`, entre as classes *`ServiceInterface`* e `ManagerQoSN` (vista na próxima subseção), e `chMRef`, entre as classes `ManagerQoSN` e `ChannelManager`, representam a comunicação entre as interfaces e o CGC do gerente de serviços. A Figura 48 apresenta também a classe `GroupManager`, que representa, juntamente com a classe `AddressManager` citada na Subseção 5.1.1, os GGEs e o CGE na arquitetura do sistema de distribuição. Por estarem associadas mais especificamente à gerência de grupos e endereços, estas classes (bem como seus relacionamentos com os outros componentes do sistema de distribuição) não serão detalhadas. Estas classes correspondem a uma aplicação do Framework para Provisão de Serviço de Multicast, podendo, em [RODRIGUES99], serem encontradas maiores informações a respeito.

5.1.3 Negociação da QoS entre Clientes e Servidores

O GNQoS é modelado, na Figura 48, pela classe `ManagerQoSN`. A negociação da QoS entre clientes e servidores, provida pela instância de `ManagerQoSN`

presente no gerente de serviços, é feita assincronamente. Quando um servidor publica um canal no gerente de serviços, ele informa a categoria de serviço associada ao canal, através do método `publish()` em `ManagerQoSN`. Esta categoria de serviço é traduzida por um *mapeador de QoS* (classe `QoSMapper`) e, em seguida, armazenada pelo CGC. A comunicação entre o GNQoS e o mapeador de QoS é simbolizado, na Figura 48, pelo relacionamento de agregação `qosMapperRef` entre *`QoSNegotiator`* e `QoSMapper`.

Posteriormente, quando um cliente tenta assinar um canal, o GNQoS é contatado, através do método `request()`, para comparar a QoS desejada pelo cliente com a QoS associada ao fluxo gerado pelo servidor. A assinatura, e a posterior entrada do cliente no grupo associado ao canal, só é confirmada caso ambas as especificações de QoS sejam compatíveis. Para que esta comparação seja possível, a requisição do cliente deve também ser previamente traduzida pelo mapeador de QoS.

O acesso das interfaces ao GNQoS é representado, na Figura 48, pelo relacionamento `serverQoSNRef` entre as classes `ServiceInterface` e `ManagerQoSN`. Já o acesso do GNQoS às informações contidas no CGC é representado pelo relacionamento `chMRef` entre as classes `ManagerQoSN` e `ChannelManager`. A Figura 49 e a Figura 50 ilustram as seqüências de invocações de métodos ocorridas, respectivamente, durante a publicação e a assinatura de um canal.

5.1.4 Configuração de Canais nas Estações Finais e na Rede ATM

Conforme ilustrado na Figura 48, os ANQoSEFs e ANQoSRs são modelados, respectivamente, por subclasses de `TransportQoSN` e `NetworkQoSN`. Tanto nos clientes quanto nos servidores, instâncias dessas subclasses são responsáveis pela configuração de fluxos com os núcleos dos canais. Os ANQoSEFs são responsáveis pela gerência de criação de threads com garantias de QoS nas estações finais, bem como pela requisição de alocação de banda passante aos ANQoSRs distribuídos pelos comutadores da rede ATM. Cada um dos ANQoSEFs e ANQoSRs existentes no sistema estão também associados a mapeadores de QoS que permitem a tradução das requisições, no caso dos ANQoSEFs, provenientes dos clientes e servidores[46], e no caso dos ANQoSRs, provenientes dos ANQoSEFs.

[46] Relembrando que, neste ponto, a QoS pode já ter sofrido um processo prévio de tradução, efetuado pela instância de `QoSMapper` ligada ao GNQoS.

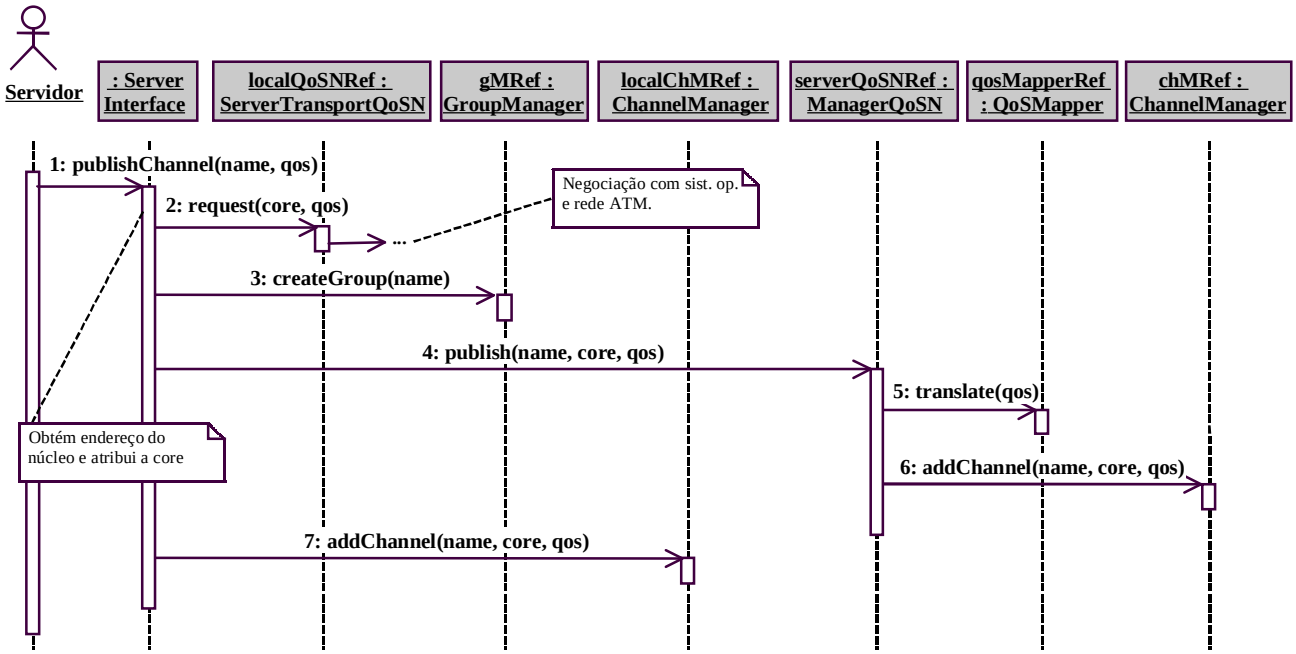


Figura 49: seqüência de publicação de um canal.

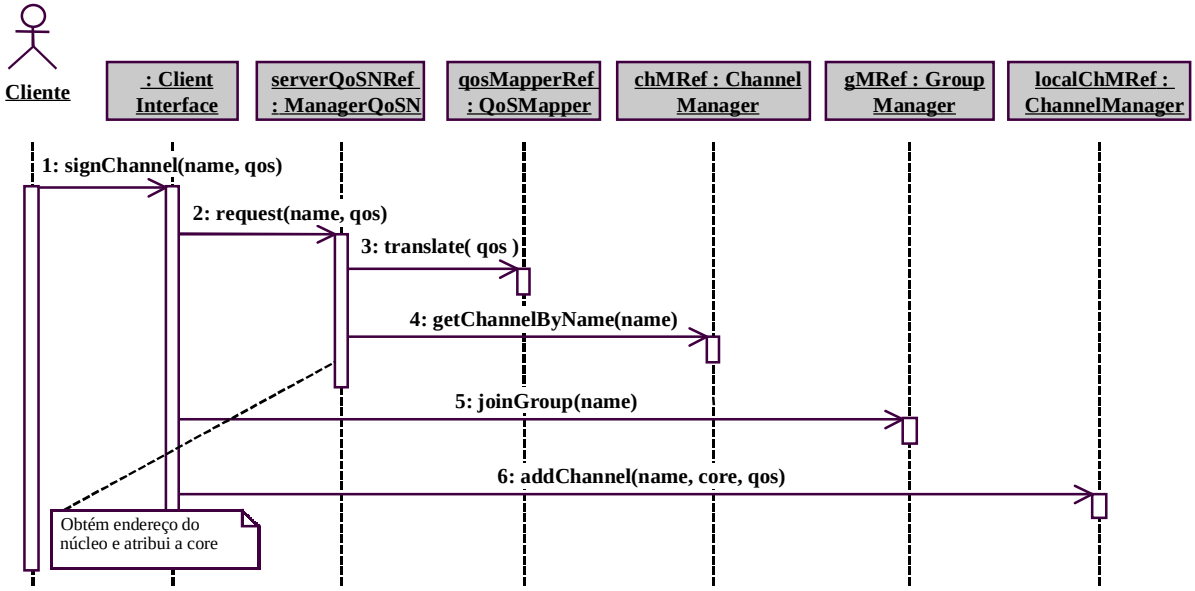


Figura 50: seqüência de assinatura de um canal.

A Figura 51 ilustra alguns componentes adicionais do sistema de distribuição, omitidos na Figura 48. Os ANQoSEFs estão associados a *controladores de admissão de threads* (classe `ThreadAdmissionController`), que validam a admissão de novas threads nas estações finais. Os ANQoSRs, por sua vez, estão associados a *controladores de admissão de enlaces virtuais* (classe `VirtualLinkAdmissionController`), que validam a admissão de novos enlaces virtuais entre estações finais e comutadores ATM, e entre comutadores ATM

adjacentes. Como se pode notar, a Figura 51 corresponde a uma aplicação do Framework para Negociação da QoS.

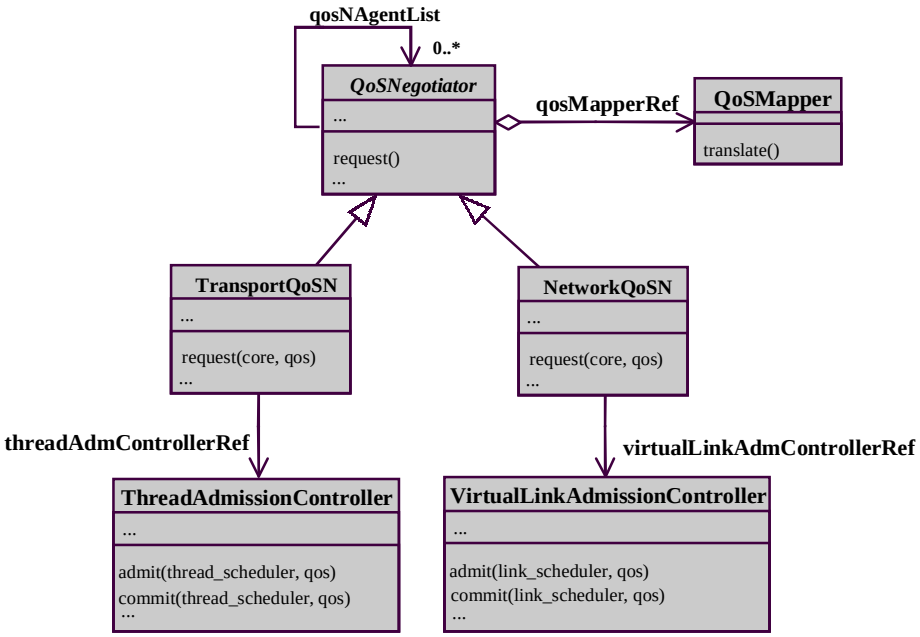


Figura 51: componentes de admissão de canais no sistema de distribuição.

Canais são definidos a partir da concatenação entre threads e enlaces virtuais. Esta concatenação é feita sempre em dois passos: primeiramente, é criado um fluxo entre o servidor e o núcleo de um canal em processo de publicação; posteriormente, à medida que clientes vão selecionando este canal, fluxos vão sendo criados entre o núcleo do canal e estes clientes.

Durante a publicação de um canal, o ANQoSEF presente na estação do servidor reserva uma parcela de utilização do processador[47] desta estação (através do método `admit()`, definido em `ThreadAdmissionController`), e repassa o pedido de publicação ao ANQoSR presente na estação. Este ANQoSR reserva uma parcela de utilização do enlace que liga a estação a um comutador da rede (através do método `admit()`, definido em `VirtualLinkAdmissionController`), e sinaliza o pedido de publicação ao ANQoSR presente neste comutador. Este ANQoSR, por sua vez, reserva uma parcela de utilização do enlace que liga esse comutador ao comutador seguinte na rota até o núcleo, e sinaliza o pedido de publicação ao ANQoSR presente no comutador seguinte. Esta sinalização se repete até que o núcleo seja alcançado. A confirmação de alocação do processador na estação do servidor (ou seja, a criação de uma thread

[47] Supondo a existência de um único processador por estação.

associada ao canal – método `commit()` em `ThreadAdmissionController`) só ocorre após uma resposta positiva de todos os ANQoSRs envolvidos na configuração do canal. Antes de responder a uma sinalização, um ANQoSR aguarda, em primeiro lugar, a resposta do ANQoSR presente no comutador seguinte na rota até o núcleo, e, em segundo lugar, a confirmação da alocação do enlace que liga o seu comutador ao comutador seguinte (ou seja, a criação de um enlace virtual associado ao canal – método `commit()` em `VirtualLinkAdmissionController`). A descoberta de quem será o núcleo de um determinado canal, bem como da rota até este núcleo, é responsabilidade dos ARs, representados na Figura 48 por `RountingSupportAgent`. Os detalhes desta classe são omitidos, podendo ser encontradas em [RODRIGUES99] maiores informações a respeito das estratégias de roteamento que podem ser utilizadas. A Figura 52 ilustra a continuação da seqüência de invocação de métodos associada à publicação de canais apresentada na Figura 49.

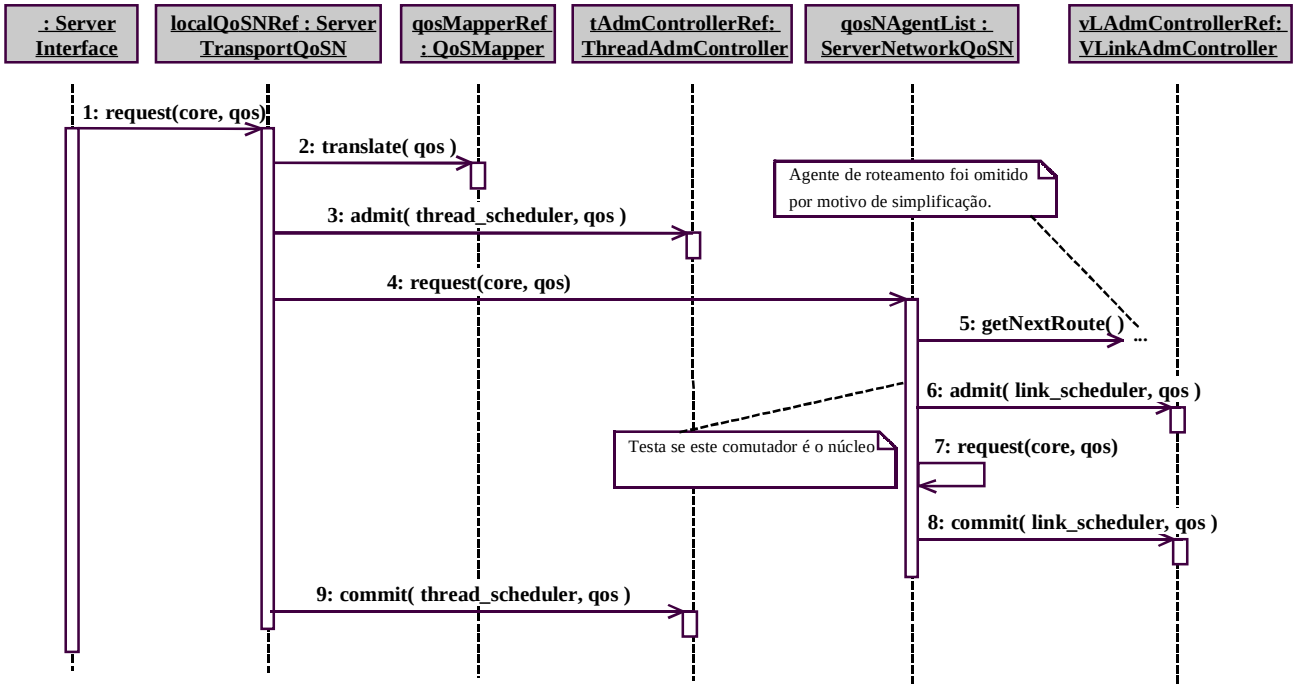


Figura 52: continuação da seqüência de publicação de um canal.

O processo de seleção de um canal, no que concerne à criação de fluxos, é bastante parecido com o processo de publicação. A diferença básica reside no fato de que a busca pelo núcleo pode ser encerrada antes mesmo dele ter sido alcançado. Para isto, basta que, durante o roteamento, seja encontrado um comutador por onde já passe o

fluxo associado ao canal que se quer selecionar. A informação de que um comutador repassa os dados de um determinado fluxo é armazenada pelo AR deste comutador. A Figura 53 ilustra a seqüência de invocação de métodos associada à seleção de canais.

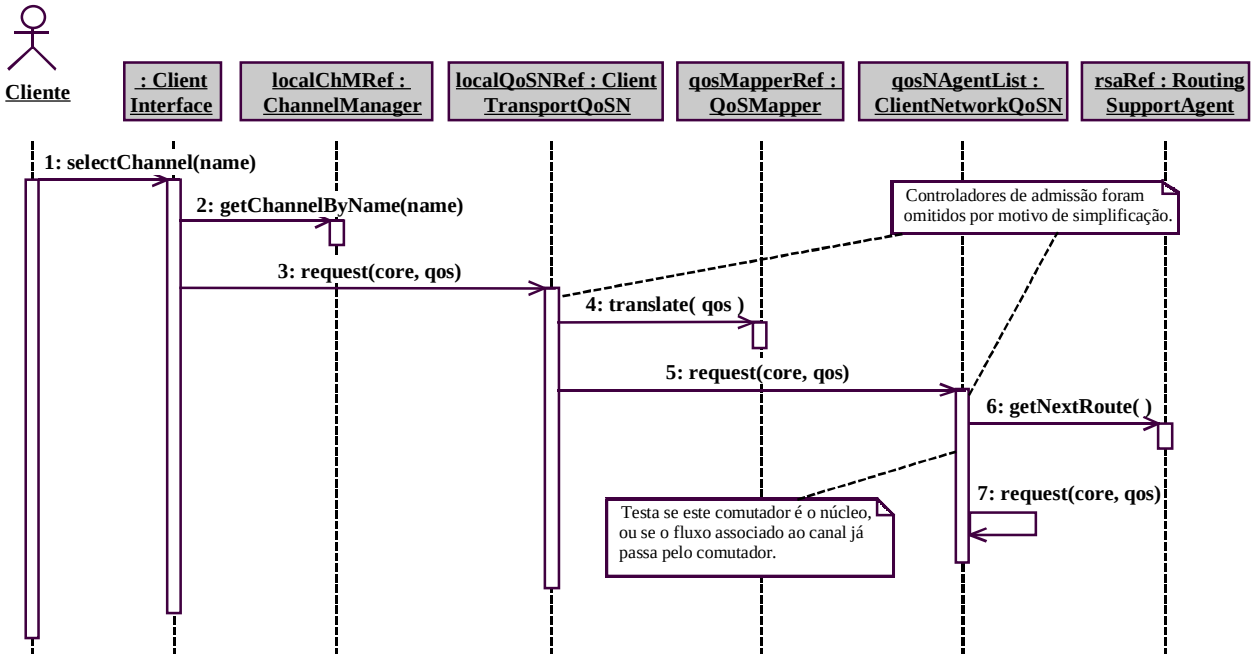


Figura 53: seqüência de seleção de um canal.

Dentre as principais características do protocolo de configuração de canais apresentado nesta subseção, podemos destacar o fato de que o servidor não precisa se responsabilizar pela gerência de grupos e canais, o que é bastante vantajoso, se considerarmos que a inclusão de novos clientes em um canal é custosa computacionalmente. Porém, para que esta abordagem seja viável, os comutadores devem ser capazes de encerrar conexões ATM de transmissão, quando, em geral, comutadores ATM convencionais só permitem o estabelecimento de conexões de transmissão entre estações finais. Este foi um dos motivos principais de ter sido definida uma plataforma de simulação, sobre a qual pôde-se implementar um protótipo do protocolo de configuração descrito nesta subseção.

Outra característica está relacionada ao fato de que não são especificados os mecanismos que possibilitam a alocação e o escalonamento de recursos no sistema de distribuição (isto é, nas estações finais e nos comutadores ATM). Isto permite que o sistema seja portável entre vários ambientes (levando-se em conta, obviamente, a

limitação inerente à “programabilidade” das redes ATM), bastando, para isso, que as subclasses de `TransportQoSN` e `NetworkQoSN` (além das classes `ThreadAdmissionController` e `VirtualLinkAdmissionController`) sejam corretamente implementadas.

5.2 Ambiente de Implementação

O ambiente de implementação utilizado consta de:

- Duas estações SPARC Ultra 1 e uma estação SPARC Station 5, todas com sistema operacional Solaris 2.5 e placas ATM Efficient 155 Mbps;
- Quatro estações Pentium II 333 MHz com sistema operacional Windows 95, sendo duas dotadas de placas ATM Efficient 155 Mbps, e duas dotadas de placas ATM IBM 25 Mbps; e
- Um comutador ATM IBM 8260 interligando todas estas estações.

As restrições impostas pelo ambiente acima induziram, para efeito de aplicação do Framework para Provisão de QoS, a implementação de um protótipo oferecendo somente um subconjunto dos serviços definidos na arquitetura apresentada na Subseção 5.1. Este protótipo, implementado em Java e C++, simula um serviço de distribuição de vídeo oferecido a aplicações Java. O protótipo é constituído pelos seguintes elementos: uma biblioteca de escalonamento para threads em máquinas virtuais Java; um servidor de escalonamento para sistemas operacionais Solaris 2.5; e uma plataforma de simulação de uma rede ATM programável.

5.2.1 Biblioteca de Escalonamento para Máquinas Virtuais Java

A biblioteca de escalonamento para threads permite a uma aplicação Java especificar a QoS de processamento de suas threads, e oferece mecanismos de escalonamento que permitem a provisão da QoS desejada, no escopo dessa aplicação. A biblioteca permite também que o desenvolvedor da aplicação se valha de várias estratégias concorrentes de escalonamento para threads nessa aplicação, e que defina

conjuntos adequados de parâmetros que descrevam a QoS dessas threads, segundo as estratégias definidas.

O Framework para Parametrização de Serviços é aplicado diretamente na estruturação de parâmetros da biblioteca. Já o mecanismo de escalonamento no qual se baseia a biblioteca corresponde à utilização da estrutura de escalonamento definida no Framework para Escalonamento de Recursos. O esquema de escalonamento é centrado na definição de uma thread, rodando na prioridade mais alta oferecida pela máquina virtual Java. Esta thread age como um escalonador central, controlando o escalonamento das outras threads presentes na máquina virtual, através da atribuição dinâmica de prioridades. Esta thread permite também a delegação temporária deste controle a outras threads de menor prioridade, que agem como escalonadores filhos do escalonador central. Este esquema de escalonamento estende o mecanismo de escalonamento para threads em Java proposto em [OAKS97], permitindo o escalonamento hierárquico entre threads de maneira simples e flexível. Detalhes a respeito da biblioteca de escalonamento para threads em máquinas virtuais Java podem ser encontrados em [GOMES98].

5.2.2 Servidor de Escalonamento para o Solaris 2.5

Para uma aplicação que requer garantias maiores de processamento, a biblioteca de escalonamento para threads, proposta na Subseção 5.2.1, é insuficiente. No sistema operacional Solaris 2.5, é adotado um esquema próprio de escalonamento hierárquico, onde em um primeiro nível encontram-se os LWPs, associados às aplicações, e em um segundo nível encontram-se as threads, que são escalonadas pelos LWPs, que por sua vez são escalonados pelo sistema. Para que se possa garantir efetivamente o processamento de um conjunto de threads segundo os requisitos por ela impostos, é necessário, primeiramente, que as LWPs associadas a esse conjunto de threads também tenham garantias de escalonamento.

Já no sistema operacional Windows 95, é adotado um esquema simples de escalonamento Round-Robin, baseado em prioridades. Este esquema inviabiliza o tratamento do escalonamento para threads num escopo global, que envolva todas as threads presentes na estação. Por este motivo, foi dada uma atenção especial ao Solaris 2.5. Neste sistema operacional, adotou-se uma abordagem, introduzida em [CHU97],

baseada em servidores de escalonamento, implementados em C++ e rodando em modo *super-usuário*, que utilizam o serviço de gerência de escalonamento para processos oferecido através da system call `priocntl()`. Esta abordagem é simples, visto que não é necessária a modificação do núcleo do sistema operacional, e razoavelmente genérica, uma vez que ela pode ser facilmente adaptada a outros sistemas da família UNIX.

A gerência de processos efetuada pelos servidores de escalonamento se assemelha ao esquema de escalonamento proposto na Subseção 5.2.1, correspondendo também à utilização da estrutura de escalonamento definida no Framework para Escalonamento de Recursos. Porém, os recursos virtuais gerenciados pelos servidores de escalonamento não são threads, mas LWPs. Assim, este esquema de escalonamento, que estende o mecanismo de escalonamento para processos proposto em [GOYAL96], permite o escalonamento hierárquico entre LWPs. Em conjunto com a biblioteca de escalonamento para threads, os servidores de escalonamento oferecem às aplicações Java garantias estatísticas[48] de escalonamento para suas threads.

5.2.3 Plataforma de Simulação de uma Rede ATM Programável

A plataforma de sinalização, sobre a qual foi implementado um protótipo do protocolo de configuração de canais apresentado na Subseção 5.1.4, é baseada no trabalho apresentado em [LI97]. Os componentes básicos dessa plataforma são os *comutadores virtuais*. Um comutador virtual é representado por um conjunto de objetos, implementados em Java, que simulam o comportamento de um comutador real programável. Dentre estes objetos, os de maior interesse para o presente trabalho são os ARs (instâncias de `RoutingSupportAgent`), os controladores de admissão de enlaces virtuais (instâncias de `VirtualLinkAdmissionController`), os ANQoSRs (instâncias das subclasses de `NetworkQoSN`), e os *elementos de comutação* (ECs). Os ECs simulam os elementos de comutação encontrados nos comutadores reais, sendo implementados por threads que efetuam o repasse de fluxos de dados desde os servidores até os clientes.

[48] Tais garantias são ditas *estatísticas*, uma vez que não se tem controle sobre as threads internas à máquina virtual Java, como a coletora de lixo, por exemplo.

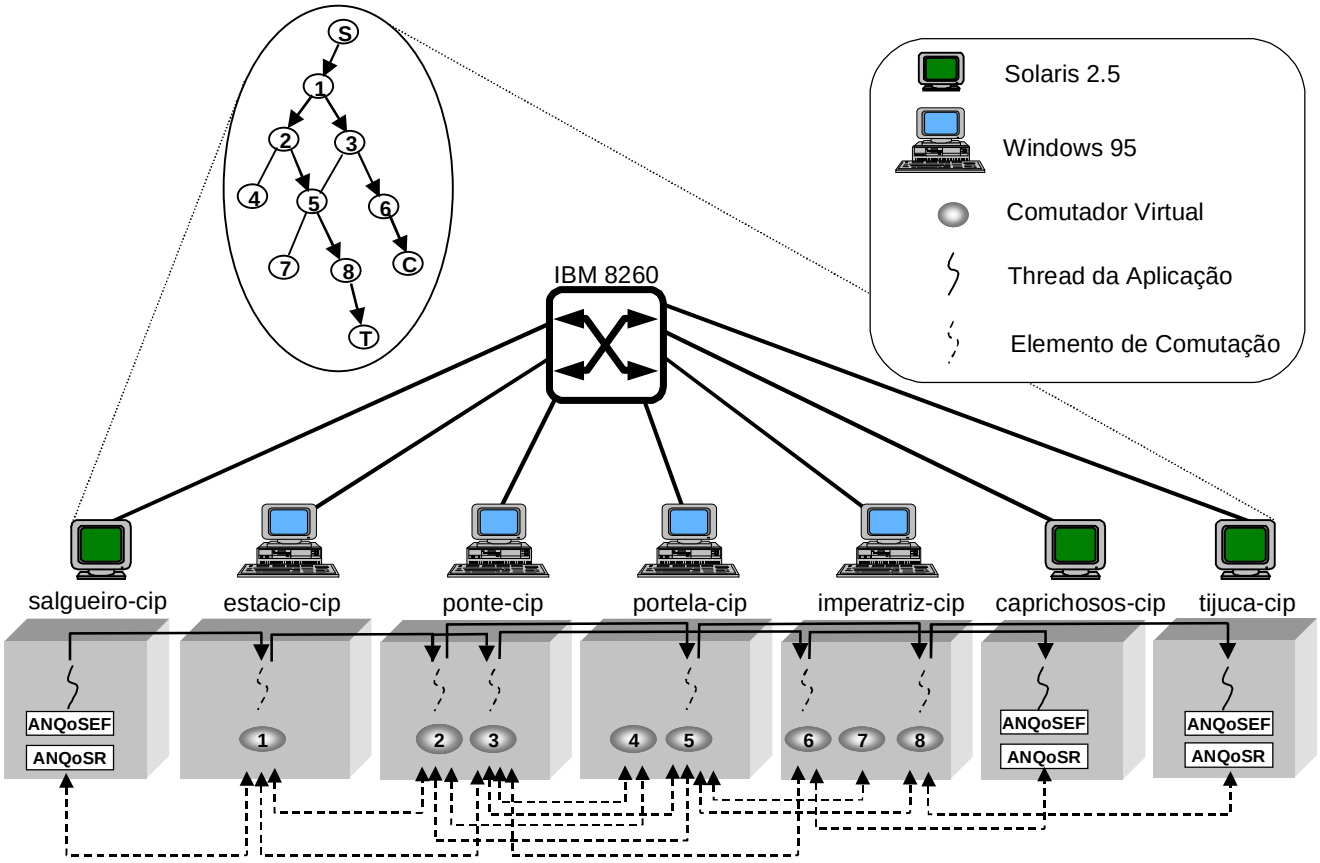


Figura 54: plataforma de simulação.

As quatro estações com sistema operacional Windows 95, presentes no ambiente de implementação, definem a infra-estrutura que permite a execução dos objetos que implementam os comutadores virtuais. Cada uma destas estações pode dar suporte a um ou mais comutadores virtuais, como mostra a Figura 54. A única regra com relação à distribuição de comutadores virtuais pelas estações é que dois comutadores virtuais adjacentes não podem estar em uma mesma estação[49]. Esta regra obriga que um enlace virtual entre dois comutadores virtuais seja sempre simulado por uma conexão ATM entre as estações onde estão localizados esses comutadores virtuais, simplificando o processo de reserva e alocação de recursos associado à admissão e criação de enlaces virtuais. Ainda assim, esta regra não restringe em nada a topologia lógica da rede, visto que quatro estações são suficientes para simular qualquer topologia. Para chegarmos a esta constatação, basta fazermos uma analogia entre o problema de se distribuir os comutadores virtuais pelas quatro estações, e o clássico *problema das quatro cores* [SZWARCFITER88], ligado à teoria de grafos.

A sinalização entre os comutadores virtuais, efetuada pelos ANQoSRs (e representada na Figura 54 pelas setas tracejadas), é implementada através do esquema de *invocações remotas de métodos* (*Remote Method Invocation – RMI*) existente em Java. Um ANQoSR presente em um comutador virtual atua, ao mesmo tempo, como cliente e como servidor RMI, para permitir tanto o recebimento de sinalizações, quanto o repasse delas a ANQoSRs presentes nos comutadores virtuais adjacentes.

A Figura 54 ilustra como um canal entre as estações `salgueiro-cip`, `caprichosos-cip` e `tijuca-cip` é constituído de enlaces virtuais (representados na figura pelas setas mais largas) entre os comutadores virtuais 1, 2, 5, 8, 3 e 6. A figura mostra também a função dos ECs de repassar fluxos de dados através de enlaces virtuais. Na plataforma implementada, não é possível ser feito um controle de escalonamento mais preciso para os ECs, logo violações de contrato podem ocorrer com freqüência. Isto indica a necessidade de se definir um mecanismo de sintonização que, contudo, não foi implementado nesta plataforma.

5.3 Sumário

Neste capítulo, foram apresentados os detalhes do projeto de um sistema de distribuição de informações, arquitetado a partir do Framework para Provisão de QoS. Contudo, as restrições impostas pelo ambiente de implementação induziram a construção de um protótipo de um serviço de distribuição de vídeo, que implementa apenas parte dos serviços definidos na arquitetura do sistema de distribuição de informações. O capítulo seguinte tece alguns comentários finais a respeito do trabalho apresentado nesta dissertação, frisando as contribuições e os pontos deixados em aberto para trabalhos futuros.

[49] A configuração da topologia lógica da rede, isto é, a distribuição dos comutadores virtuais pelas estações e a definição do grafo de conectividade entre estes comutadores, é feita previamente, durante a iniciação do serviço.

Capítulo 6

Conclusões

Os tópicos relacionados à provisão de QoS em sistemas multimídia têm sido, em sua maioria, parcialmente tratados na literatura. A maioria das abordagens apresenta pelo menos uma das seguintes características: elas são restritas a subsistemas específicos, em especial os sistemas operacionais e de comunicação; elas tratam a questão da integração da QoS sob o ponto de vista de interfaces somente; elas não facilitam a introdução de novos serviços.

Como alternativa a essas abordagens, têm-se proposto, recentemente, novos modelos que permitam o oferecimento, nos ambientes distribuídos, de um único serviço, configurável de acordo com a necessidade dos usuários. Como conseqüência da aplicação destes modelos, parte da responsabilidade pelo projeto, implementação e configuração de serviços deverá recair sobre os próprios usuários. Estas tarefas deverão se tornar tão corriqueiras quanto a implementação das aplicações dos usuários.

O Modelo de Referência Unificado, em desenvolvimento junto ao projeto RAVel no Laboratório TeleMídia da PUC-Rio, segue a abordagem de serviços configuráveis. Este modelo dá suporte à definição de ambientes de processamento e comunicação cujos componentes podem representar tanto objetos do universo das aplicações multimídia, como dispositivos multimídia, hiperdocumentos, etc., quanto entidades de protocolo em arquiteturas de comunicação. A presente dissertação insere-se no projeto RAVel como um complemento ao Modelo de Referência Unificado, ao apresentar um framework que organiza a provisão de QoS através da definição de estruturas que permitem o projeto, implementação e configuração de parâmetros de

caracterização de serviços e de mecanismos de provisão de QoS em todos os níveis de abstração e escalas de distribuição envolvidos em um ambiente de processamento e comunicação qualquer. Porém, a forma como foi definido o Framework para Provisão de QoS permite também a sua aplicação em ambientes não necessariamente modelados segundo o Modelo de Referência Unificado. A generalidade do domínio de aplicação do Framework para Provisão de QoS abrange desde a definição de mecanismos de provisão de QoS agindo sobre componentes das aplicações e dos protocolos de comunicação, até a interface com o usuário humano. Isto possibilita a construção de sistemas altamente configuráveis no que se refere à introdução de novos serviços, ao mesmo tempo que facilita a tarefa dos projetistas de sistemas no que concerne à implementação dos mecanismos de provisão de QoS.

6.1 Contribuições da Dissertação

As principais contribuições desta dissertação foram:

- Definir uma terminologia básica que formaliza o conceito de QoS; e
- Especificar o Framework para Provisão de QoS.

Definição de uma Terminologia Básica para QoS

A definição da terminologia básica apresentada no Capítulo 2 englobou a descrição dos principais elementos que compõem um ambiente de processamento e comunicação, a enumeração dos princípios que governam a implementação de mecanismos de provisão de QoS nesses ambientes, e a definição de um modelo genérico de operação para sistemas multimídia, que divide a operação desses mecanismos em diferentes fases de provisão de QoS. Esta terminologia pode servir, futuramente, como base para o projeto de outros trabalhos relacionados à questão da provisão de QoS em sistemas multimídia.

Especificação do Framework para Provisão de QoS

A especificação, no Capítulo 4, do Framework para Provisão de QoS, envolveu uma análise mais detalhada das estruturas de parametrização e dos

mecanismos de provisão de QoS que compõem o modelo genérico de operação de sistemas multimídia apresentado no Capítulo 2. Esta análise incluiu a descrição de classes de objetos e relacionamentos que representam abstratamente os componentes do Framework para Provisão de QoS. Esta descrição, feita através de uma notação textual informal auxiliada de uma notação gráfica, é dividida em três partes (um Framework para Parametrização de Serviços, Frameworks para Compartilhamento de Recursos e Frameworks para Orquestração de Recursos), refletindo os diferentes níveis de especificação abrangidos pelo framework.

6.2 Trabalhos Futuros

Há quatros importantes áreas de trabalhos futuros necessárias ao amadurecimento deste trabalho:

- Utilizar uma linguagem de especificação formal para descrever o Framework para Provisão de QoS;
- Reavaliar o Framework para Provisão de QoS com relação à questão da comunicação multicast;
- Acrescer ao Framework para Provisão de QoS um modelo que represente o processo de configuração de sistemas multimídia;
- Reforçar a validação do Framework para Provisão de QoS através da aplicação do mesmo em mais casos de uso.

Formalização do Framework para Provisão de QoS

Apesar de, neste trabalho, ter-se tido a preocupação constante em apresentar o Framework para Provisão de QoS de maneira precisa, os padrões de estrutura e comportamento delineados pelo framework não foram formalmente descritos. Além disso, a notação UML é limitada no que se refere à distinção entre descrições de famílias de arquiteturas e descrições de configurações específicas de uma arquitetura, sendo mais adequada a essas últimas. Esta característica é especialmente crítica na modelagem dos Frameworks para Compartilhamento e Orquestração de

Recursos, onde se tem em mente exatamente a descrição de famílias de arquiteturas. Embora a notação UML ofereça elementos que permitam a descrição, através de diagramas, de alternativas de projeto com relação a uma arquitetura específica modelada por esses frameworks, isto não significa que esses diagramas capturam adequadamente todas as variedades de arquiteturas modeladas por eles. A utilização de uma linguagem formal de especificação como, por exemplo, a notação *Z* [SPIVEY92] ou a linguagem de especificação de arquiteturas *Wright* [ALLEN97], em conjunto com a descrição apresentada neste trabalho, permitiria uma maior precisão na descrição dos padrões de estrutura e comportamento definidos pelo Framework para Provisão de QoS.

Provisão de Serviço de Multicast

Embora não se tenha descartado, em ponto algum desta dissertação, a possibilidade de modelagem de mecanismos que permitam a configuração de fluxos ponto-a-multiponto, há ainda muito a ser reavaliado no Framework para Provisão de QoS com relação à questão da comunicação multicast, principalmente no que se refere à heterogeneidade de consumidores de fluxos ponto-a-multiponto (ou seja, consumidores de um mesmo fluxo que possuem requisitos distintos de comunicação). Uma vez que o projeto RAVel envolve também a definição de um Framework para Provisão de Serviço de Multicast, o processo de reavaliação deverá consistir fundamentalmente na integração dos dois frameworks.

Processo de Configuração de Sistemas Multimídia

O Framework para Provisão de QoS deve ser acrescido de um modelo que represente o processo de configuração de sistemas multimídia. Embora os pontos de flexibilização do framework lhe concedam a característica altamente desejável de configurabilidade, tais pontos não são totalmente independentes entre si, o que dificulta a utilização do framework por um projetista de sistemas. Em um exemplo típico, quando insere-se em um sistema uma nova categoria de serviço, novas políticas de provisão de QoS (isto é, estratégias de mapeamento, admissão, escalonamento e monitorização) possivelmente terão que ser também introduzidas.

Validação do Framework para Provisão de QoS

Como em todo framework, o Framework para Provisão de QoS deve ser continuamente validado e refinado, conforme a sua aplicação em outros casos de uso, além dos sugeridos neste trabalho. Este é, certamente, um princípio básico de construção de frameworks:

> *"Good frameworks are usually the result of many design interactions and a lot of hard work."*
>
> Wirfs-Brock and Johnson, 1990

Apêndice A

Linguagem de Modelagem Unificada (UML)

UML (*Unified Modeling Language*) [UML97] é uma linguagem usada para especificar, visualizar, e documentar os artefatos de sistemas orientados a objetos em desenvolvimento. A linguagem UML, em sua totalidade, provê uma combinação de semântica, sintaxe, notações e um meta-modelo para o projeto desses sistemas. Este apêndice se concentra em algumas das notações oferecidas por esta linguagem, que são voltadas para a especificação dos requisitos funcionais de sistemas (ou seja, o que estes sistemas devem prover em termos de serviços para seus usuários). Maiores detalhes sobre a linguagem UML podem ser encontrados em [UML97].

Diagramas de classes

Diagramas de classes descrevem classes, suas estruturas internas e seus relacionamentos com outras classes. A Figura 55 mostra a notação UML para classes abstratas e concretas. Um *classe* é denotada por um retângulo dividido em três partes, com o nome da classe localizado na parte superior do mesmo. Os *atributos* da classe aparecem abaixo do nome da classe, e os *métodos* aparecem abaixo dos atributos. Os tipos dos atributos, bem como dos argumentos e valores de retorno dos métodos, são opcionais. Na notação UML, o nome do tipo aparece sempre após o nome do método

(no caso de valores de retorno), atributo ou argumento. Classes e métodos *abstratos* são grafados em itálico. Especificamente no presente trabalho, são utilizadas reticências para representar os atributos, métodos, argumentos e tipos considerados como não essenciais para a compreensão dos frameworks apresentados.

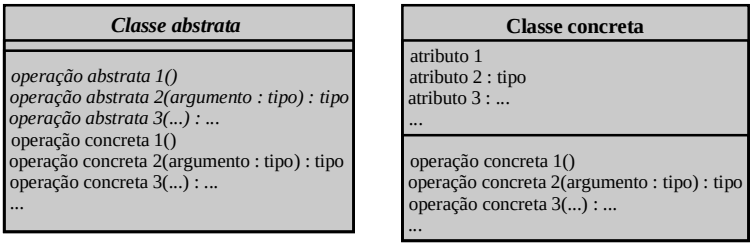


Figura 55: classes UML.

A Figura 56 mostra os vários tipos de relacionamentos entre classes. A herança entre classes (*relacionamento de especialização*) é denotada por uma seta com a ponta em triângulo, conectando a subclasse à classe pai. Uma linha simples é utilizada para representar um *relacionamento de associação* entre classes. O relacionamento de associação é usado para indicar que as instâncias das classes associadas podem possuir referências umas às outras, o que permite a troca de mensagens entre as instâncias dessas classes. Um *relacionamento de agregação* (também conhecido como "*parte-todo*") é representado por uma linha com um losango próximo à classe agregadora. O relacionamento de agregação é usado para indicar que as instâncias da classe agregadora são fisicamente construídas a partir de instâncias da classe agregada, ou que a classe agregadora contém logicamente (através de referências) instâncias da classe agregada. A distinção semântica entre os relacionamentos de agregação e de associação é, por vezes, muito tênue. Para tentar melhor diferenciar ambos os relacionamentos, o presente trabalho adota uma representação em que instâncias de uma classe agregada só podem ser referenciadas por uma única instância da classe agregadora. Já nos relacionamentos de associação, múltiplas instâncias de uma classe podem referenciar uma mesma instância da outra classe do relacionamento. Casos particulares de relacionamentos de associação e agregação são os *relacionamentos recursivos*, que envolvem instâncias de uma mesma classe. Outro relacionamento de interesse é o *relacionamento de dependência*, representado por uma linha tracejada, com uma seta simples apontando para a classe provedora, da qual a outra classe do relacionamento depende para prover certos serviços. Tais serviços incluem, por exemplo, a criação de instâncias da classe

provedora, ou o uso da classe provedora como tipo de um argumento ou valor de retorno de um método da classe dependente.

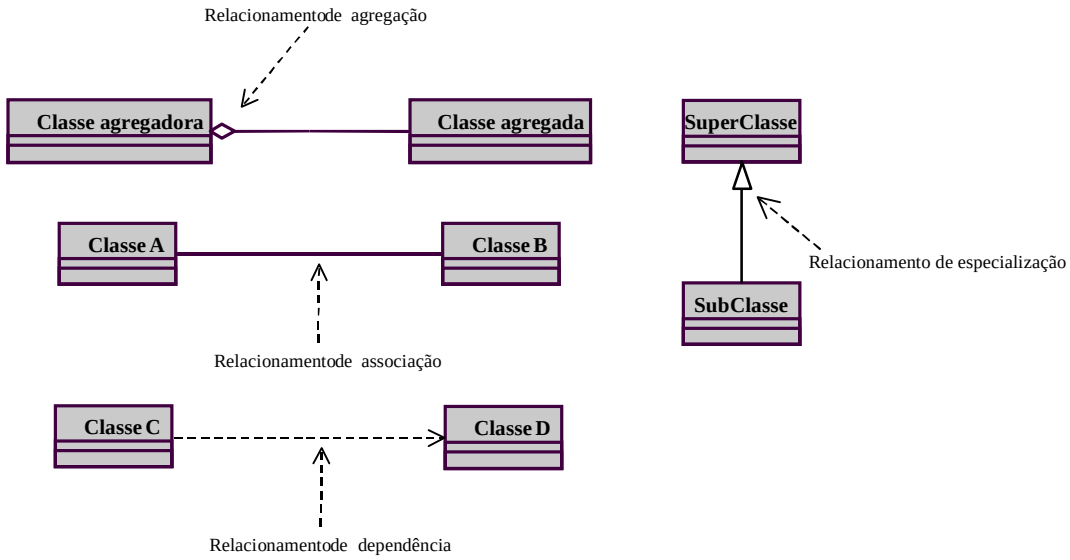


Figura 56: relacionamentos UML.

Como ilustra a Figura 57, relacionamentos de associação e agregação podem ser definidos de forma mais precisa através do uso de *adornos*. O adorno de *cardinalidade* especifica quantas instâncias de uma classe podem estar relacionadas com uma única instância de uma outra classe. Este adorno fica sempre próximo à classe da qual se quer especificar a cardinalidade. Alguns exemplos de adornos de cardinalidade são:

- 1 Exatamente um
- 0..* Zero ou mais
- 1..* Um ou mais
- 0..1 Zero ou um
- 1..* Um ou mais
- 5..8 Intervalo específico (5, 6, 7 ou 8)
- 4..7,9 Combinação (4, 5, 6, 7, ou 9)

Outro adorno bastante utilizado neste trabalho é o de *navegabilidade*. Um adorno de navegabilidade indica a direção de um relacionamento, e é representado por uma seta simples junto ao mesmo. Este adorno fica sempre próximo à classe cujas instâncias são referenciadas pelas instâncias da outra classe do relacionamento. A ausência de adornos de navegabilidade indica que o relacionamento é recíproco, ou seja, instâncias de ambas as classes podem se referenciar mutuamente.

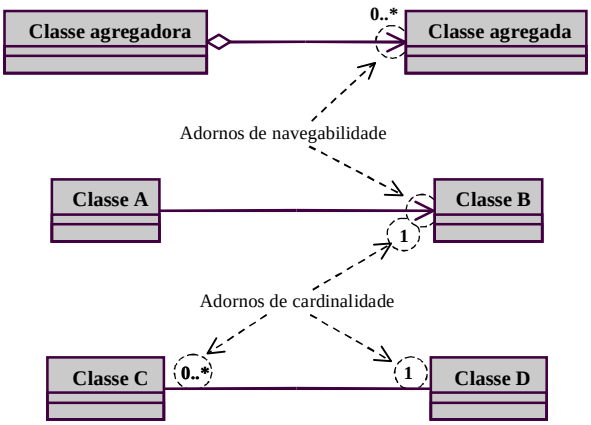


Figura 57: adornos UML.

Outros elementos secundários podem ser encontrados em um diagrama de classes que segue a notação UML. Dentre eles, *nomes de papéis* e *anotações* são os de maior interesse para o presente trabalho. As extremidades de um relacionamento (onde as classes se conectam) são denominadas, na notação UML, de *papéis* do relacionamento. *Nomes de papéis* podem ser usados para denotar o propósito pelo qual uma classe se relaciona com outra. O nome de um papel é colocado próximo à classe que exerce este papel em um relacionamento. Uma *anotação* é um campo livre que pode conter qualquer tipo de informação, desde dicas até trechos de código. Essas anotações podem ser ligadas a outros elementos de um diagrama através de *âncoras de anotação*, representadas por linhas tracejadas. Um exemplo de um diagrama de classes que utiliza todos os elementos apresentados nesta seção é apresentado na Figura 58.

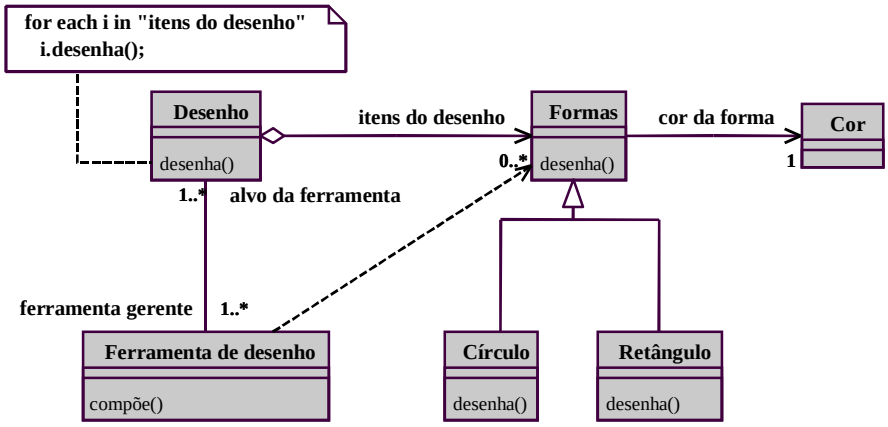


Figura 58: exemplo de diagrama de classes.

Diagramas de Seqüência

Diagramas de seqüência mostram as interações entre instâncias de classes (através do disparo de métodos) dispostas em uma seqüência temporal. Uma instância de uma classe é representada por um retângulo contendo o nome da instância, o nome da instância e da sua classe, ou somente o nome da classe. A Figura 59 mostra um diagrama de seqüência associado ao diagrama de classes da Figura 58. A linha do tempo inicia-se na parte superior e segue em direção à parte inferior de um diagrama de seqüência. O disparo de métodos entre instâncias de classes é representado por setas que apontam da instância disparadora do método para a instância onde o método será disparado.

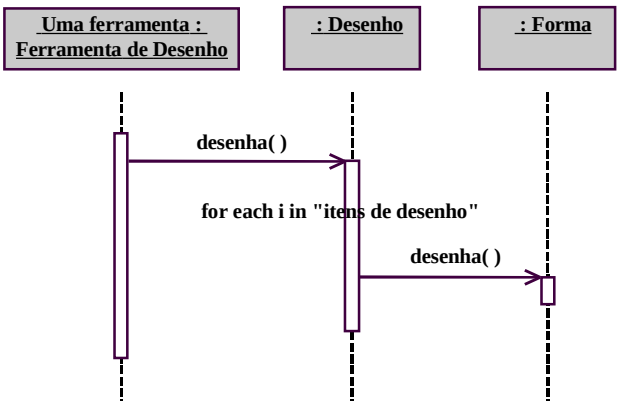


Figura 59: exemplo de diagrama de seqüência UML.

Bibliografia

[ALLEN97] ALLEN, R. “Formalizing Software Architecture”. *PhD Thesis*, Carnegie Mellon School of Computer Science, 1997.

[ANICK82] ANICK, D., MITRA, D., SONDHI, M. “Stochastic theory of a data-handling system with multiple sources”. *Bell Systems Techinical Journal*, vol. 61, pp. 1971-1894, 1982.

[ATM93] ATM FORUM. *ATM User-Network Interface Specification - Version 3.0*. 1993.

[BANERJEA91] BANERJEA, A., MAH, B. A. “The Real-Time Channel Administration Protocol”. *Proc. 2nd Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, Heidelberg, Alemanha, Novembro de 1991.

[BESSE94] BESSE, L., DAIRAINE, L., FEDAOUI, L., TAWBI, W., THAI, K. “Towards na Architecture for Distributed Multimedia Application Support”. *Proc. International Conference on Multimedia Computing and Systems*, Boston, USA, Maio de 1994.

[BUSCHMANN95] BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., STAL, M. *A System of Patterns*. Wiley, 1995.

[CAMPBELL92] CAMPBELL, A., COULSON, G., GARCIA, F., HUTCHISON, D. “A Continuous Media Transport and Orchestration Service”. *Proc. ACM SIGCOMM’92*, Baltimore, Maryland, USA, Agosto de 1992.

[CAMPBELL93] CAMPBELL, A., COULSON, G., GARCIA, F., HUTCHISON, D., LEOPOLD, H. “Integrated Quality of Service for Multimedia Communications”. *Proc. IEEE INFOCOM’93*, pg. 732-739, San Francisco, USA, Abril de 1993.

[CAMPBELL95] CAMPBELL, A., AURRECOECHEA, C., HAUW, L. “A Review of Quality of Service Architectures”. *ACM Multimedia Systems Journal*, Novembro de 1995.

[CAMPBELL96] CAMPBELL, A. “A Quality of Service Architecture”. *PhD Thesis*. Lancaster University, UK, Janeiro de 1996.

[CHU97] CHU, H., NAHRSTEDT, K. “A Soft Real Time Scheduling Server in UNIX Operating System”. *European Workshop on Interactive Multimedia Systems and Telecommunication Services (IDMS’97)*, Setembro de 1997.

[CLARK90] CLARK, D., TENNENHOUSE, D. L. “Architectural Consideration for a New Generation of Protocols”. *Proceedings of ACM SIGCOMM’90*, vol. 20, no. 4, pg. 200-208, Philadelphia, USA, Setembro de 1990.

[COLCHER98] COLCHER, S., SOARES, L. F. G. “Modelo de Referência Unificado para Arquitetura de Protocolos e Programação de Aplicações Multimídia”. *Anais do 16º Simpósio Brasileiro de Redes de Computadores*, pg. 631-650, Rio de Janeiro, Maio de 1998.

[COMER95] COMER, D. E. *Internetworking with TCP/IP – Vol 1: Principles, Protocols and Architecture*. Prentice Hall, 1995.

[COULSON93] COULSON, G., BLAIR, G. “Micro-kernel Support for Continuous Media in Distributed Systems”. *Technical Report MPG-93-04*, Department of Computing, Lancaster University, UK, 1993.

[COULSON95] COULSON, G., BLAIR, G. “Architectural Principles and Techniques for Distributed Multimedia Application Support in Operating Systems”. *ACM Operating System Review*, vol. 29, no. 4, pg. 17-24, Outubro de 1995.

[COULSON95a] COULSON, G., CAMPBELL, A., ROBIN, P., BLAIR, G. S., PAPATHOMAS, M., SHEPHERD, D. “The Design of a QoS Controlled ATM Based Communications Systems in Chorus”, *IEEE Journal on Selected Areas in Communications*, Special Issue on ATM Local Area Networks, 1995.

[COULSON97] COULSON, G., WADDINGTON, D. G. “A CORBA Compliant Real-Time Multimedia Platform for Broadband Networks”. *Technical Report*, Distributed Multimedia Research Group, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, 1997.

[DACHS88] DACHS, N. *Estatística Computacional*. Livros Técnicos e Científicos Editora Ltda., 1988.

[DANTHINE94] DANTHINE, A. "The OSI 95 Project – The OSI Transport Service with Multimedia Support". *Research Reports ESPRIT – Project 5341 – OSI95*, vol. 1, pp. 13-18, University of Liège, Belgium, 1994.

[DOERINGER90] DOERINGER, W., et al. "A Survey of Light-weight Transport Protocols for High-speed Networks". *IEEE Transactions on Communications*, Novembro de 1990.

[FERRARI95] FERRARI, D. "The Tenet Experience and the Design of Protocols for Integrated Services Internetworks". *Multimedia Systems Journal*, Novembro de 1995.

[FORD96] FORD, B., SUSARLA, S. "CPU Inheritance Scheduling". *Operating Systems Design and Implementation (OSDI'96),* Outubro de 1996.

[FIROIU95] FIROIU, V., TOWSLEY, D. "Call Admission and Resource Reservation for Multicast Sessions". *Technical Report*, Lederle Graduate Research Center, University of Massachusetts, Amherst, USA, Setembro de 1995.

[FRENCH94] FRENCH, L. J., WILLSON, I. D., GILMURRY, D. P. "ATMos II User Manual". *Technical Report*, Olivetti Research Limited, Cambridge, UK, 1994.

[GALLMEISTER95] GALLMEISTER, B. *POSIX.4: Programming for the Real World*. O'Reilly & Associates, 1995.

[GAMMA95] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[GARCIA93] GARCIA, F. "A Continuous Media Transport and Orchestration Service". *PhD Thesis*, Lancaster University, UK, Junho de 1993.

[GARRETT94] GARRETT, M., WILLINGER, W. "Analysis, modeling and generation of self-similar VBR video traffic". *Proc. ACM SIGCOMM'94*, London, U.K., Agosto de 1994.

[GOMES98] GOMES, A. T. A. "Biblioteca para Escalonamento de Threads com Diferentes Qualidades de Serviço". *Projeto Final de Programação do Curso de Mestrado em Informática da PUC-Rio.* http://www.telemidia.puc-rio.br/~atagomes/public/pfp.ps.gz, 1998.

[GOVINDAN91] GOVINDAN, R., ANDERSON, D. P. “Scheduling and IPC Mechanisms for Continuous Media”. *Thirteenth ACM Symposium on Operating Systems Principles*, Pacific Grove, California, USA, vol. 25, pg. 68-80, 1991.

[GOYAL96] GOYAL, P., GUO, X., VIN, H. M. “A Hierarchical CPU Scheduler for Multimedia Operating Systems”. *Operating Systems Design and Implementation (OSDI'96).* Outubro de 1996.

[GOYAL96a] GOYAL, P., VIN, H. M., CHENG, H. “Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks”. *Proceedings of ACM SIGCOMM’96*, 1996.

[GUEDES97] GUEDES, L. A., OLIVEIRA, P. C., FAINA, L. F., CARDOZO, E. “QoS Agency: An Agent-Based Architecture for Supporting Quality of Service in Distributed Multimedia Systems”. *IEEE Conference on Protocols for Multimedia Systems – Multimedia Networking*, Santiago, Chile, Novembro de 1997.

[HAFID95] HAFID, A. “A Hierarchical Negotiation for Distributed Multimedia Applications in a Multi-Domain Environment”. *Proceedings of the Second International Workshop on Protocols for Multimedia Systems*, Salzburg, Áustria, pg. 325-337, 1995.

[HERRTWICH92] HERRTWICH, R. G. “The HeiProjects: Support for Distributed Multimedia Applications”. *IBM Technical Report*, no. 43.9206, Março de 1992.

[HUNI95] HUNI, H., JOHNSON, R., ENGEL, R. “A Framework for Network Protocol Software”. *OOPSLA’95*, ACM SIGPLAN Notices, 1995.

[HUTCHISON91] HUTCHISON, N. C., PETERSON, L. L. “The x-Kernel: An Architecture for Implementing Network Protocols”. *IEEE Transactions on Software Engineering*, vol. 17, pg. 64-76, Janeiro de 1991.

[HUTCHISON94] HUTCHISON, D., COULSON, G., CAMPBELL, A., BLAIR, G. S. “Quality of Service Management in Distributed Systems”. *Technical Report*, Distributed Multimedia Research Group, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, 1994.

[ISO84] International Organization for Standardization / International Eletrotechnical Commitee. “Information Processing Systems – Open Systems Interconnection – Basic Reference Model – Part 1: Basic Reference Model”. *International Standard 7498-1*, 1984.

[ISO86] International Organization for Standardization / International Eletrotechnical Commitee. “Information Processing Systems – Open Systems Interconnection – Transport Protocol Specification”. *International Standard 8073*, 1986.

[ISO90] International Organization for Standardization / International Eletrotechnical Commitee. “Information Technology – Open Systems Interconnection – Management Information Services – Structure of Management Information – Part 4: Guidelines for the Definition of Managed Objects”. *International Standard 10165-4*, 1990.

[ISO95] International Organization for Standardization / International Eletrotechnical Commitee. “QoS: Basic Framework”. *Draft International Standard (DIS) 9309*, 1995.

[ISO95a] International Organization for Standardization / International Eletrotechnical Commitee. “Reference Model of Open Distributed Processing, Part I: Overview”. *Draft International Standard (DIS) 10746*, 1995.

[ISO96] International Organization for Standardization / International Eletrotechnical Commitee. “Information Processing Systems – Computer Graphics and Image Processing – Presentation Environments for Multimedia Objects (PREMO), Part 3: Multimedia Systems Services Component”. *Draft International Standard (DIS) 14478-3*, 1996.

[ITUT.I.121] International Telecommunications Union. “Broadband Aspects of ISDN”. *Recommendation I.121*, Revision 1, Julho de 1991.

[ITUT.I.211] International Telecommunications Union. “B-ISDN Service Aspects”. *Recommendation I.211*, Março de 1993.

[ITUT.I.321] International Telecommunications Union. “B-ISDN Protocol Reference Model and Its Application”. *Recommendation I.321*, Julho de 1991.

[ITUT.I.350] International Telecommunications Union. “General Aspects of Quality of Service and Network Performance in Digital Networks, Including ISDNs”. *Recommendation I.350*, Revision 1, Novembro de 1993.

[ITUT.I.362] International Telecommunications Union. “B-ISDN ATM Adaptation Layer (AAL) Functional Description”. *Recommendation I.362*, Revision 1, Novembro de 1993.

[ITUT.I.363] International Telecommunications Union. “B-ISDN ATM Adaptation Layer (AAL) Specification”. *Recommendation I.363*, Revision 1, Novembro de 1993.

[ITU.T.Q.2931] International Telecommunications Union. “B-ISDN Digital Subscriber Signalling System No. 2 (DSS 2) – User-Network Interface Specification for Basic Call/Connection Control”. *Recommendation Q.2931*, Junho de 1994.

[KNIGHTLY97] KNIGHTLY, E. W., ZHANG, H. “D-BIND: An Accurate Traffic Model for Providing QoS Guarantees to VBR Traffic”. *IEEE/ACM Transactions on Networking*, vol. 5, no. 2, pg. 219-231, Abril de 1997.

[KUROSE93] KUROSE, J. F. “Open Issues and Challenges in Providing Quality of Service Guarantees in High-Speed Networks”. *ACM Computer Communication Review*, vol. 23, no. 1, pg. 6-15, Janeiro de 1993.

[LAZAR94] LAZAR, A. A. “Challenges in Multimedia Networking”. *Proceedings of the International Hi-Tech Forum*, Osaka, Japão, Fevereiro de 1994.

[LAZAR95] LAZAR, A. A., LIM, K. S., MARCONCINI F. “Binding Model: Motivation and Description”. *http://www.ctr.columbia.edu/comet/xbind/xbind.html*, Novembro de 1995.

[LI97] LI, H., PUNG, H. K., NGOH, L. H. “Active Multicast Service Architecture for User Customized Multimedia Data Transmission over ATM Networks”. *Technical Report*, Multimedia Systems Research Lab., National University of Singapore, 1997.

[LIU73] LIU, C. L., LAYLAND, J. W. “Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment”. *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pg. 46-61, Fevereiro de 1973.

[LU96] LU, G. *Communication and Computing for Distributed Multimedia Systems*. Norwood: Artech House, 1996.

[MAUTHE96] MAUTHE, A., COULSON, G. “Scheduling and Admission Testing for Jitter Constrained Periodic Threads: Discussion and Proof”. *Technical Report MPG-96-12*, Distributed Multimedia Research Group, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, 1996.

[MEYER88] MEYER, B. *Object-Oriented Software Construction*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[NAHRSTEDT95] NAHRSTEDT, K., STEINMETZ, R. “Resource Management in Networked Multimedia Systems”. *Computer Communication Review*, pg. 52-63, Maio de 1995.

[NAHRSTEDT95a] NAHRSTEDT, K., STEINMETZ, R. “End-to-End QoS Guarantees: Lessons Learned from OMEGA”. *Technical Report*, Distributed Systems Laboratory, University of Pennsylvania, USA, 1995.

[OAKS97] OAKS, S., WONG, H. *Java Threads*. O’Reilly & Associates, 1997.

[OMG96] Object Management Group. *The Common Object Request Broker: Architecture and Specification.* Revision 2.0, 1996.

[PAN95] PAN, H., NGOH, L. H., LAZAR, A. A. "A Time Scale Dependent Disk Scheduling Scheme for Multimedia-on-Demand Servers". *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS’95)*, pg. 572-579, Hiroshima, Japão, Junho de 1995.

[POSTEL81] POSTEL, J. B. “Transmission Control Protocol – Darpa Internet Program Protocol Specification”. *Request for Comments 793*, University of Southern California / Information Sciences Institute, Setembro, 1981.

[PREE95] PREE, W. *Design Patterns for Object-Oriented Software Development*. Addison Wesley, 1995.

[RODRIGUES99] RODRIGUES, M. A. A., COLCHER, S., SOARES, L. F. G. “Um Framework para a Provisão de Serviço de Multicast em Ambientes Genéricos de Comunicação de Dados”. *A ser publicado nos Anais do 17º Simpósio Brasileiro de Redes de Computadores*, Salvador, Maio de 1999

[SALTZER84] SALTZER, J., REED, D., CLARK, D. “End-to-End Arguments in Systems Design”. *ACM Transactions on Computer Systems*, vol. 2, no. 4, 1984.

[SCHMIDT97] SCHMIDT, D. C. “The ADAPTIVE Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Software”. *Proceedings of the 12th Annual Sun Users Group Conference*, pg. 214-225, San Jose, California, USA, Dezembro de 1993.

[SCHULZRINNE95] SCHULZRINNE, H., CASNER, S. “RTP: A Transport Protocol for Real-Time Applications; RFC-1889”. *Internet Request for Comments*, no. 1889, Network Information Center, 1995.

[SILBERSCHATZ95] SILBERSCHATZ, A., GALVIN, P. B. *Operating System Concepts*. Addison Wesley, 1995.

[SOARES95] SOARES, L. F. G., LEMOS, G., COLCHER S. *Redes de Computadores: das LANs, MANs e WANs às redes ATM*. 2ª edição. Rio de Janeiro: Campus, 1995.

[SOLARIS95] SunSoft Inc. *Solaris 2.5 Multithreaded Programming Guide*. SunSoft Press, 1995.

[SPIVEY92] SPIVEY, J. M. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.

[SZWARCFITER88] SZWARCFITER, J. L. *Grafos e Algoritmos Computacionais*. 2ª edição, Editora Campus, 1988.

[TANENBAUM92] TANENBAUM, A. S. *Modern Operation Systems*. Prentice Hall, 1992.

[TENNENHOUSE89] TENNENHOUSE, D. L. “Layered Multiplexing Considered Harmful”. *Protocols for High-Speed Networks*, Rudin and Williamson (Editors), North Holland, Amsterdam, 1989.

[TOPOLCIC90] TOPOLCIC, C. “Experimental Internet Stream Protocol, Version 2 (ST-II); RFC-1190”. *Internet Request for Comments*, no. 1190, Network Information Center, Outubro de 1990.

[TURNER86] TURNER, J. S. “New Directions in Communications (or Which Way to the Information Age)”. *IEEE Communications*, vol. 24, no. 10, pg 8-15, Outubro de 1986.

[UML97] Rational Software Corporation. *Unified Modeling Language: Notation Guide*. 1997.

[VOGEL95] VOGEL, A., et al. “Distributed Multimedia and QoS: a Survey”. *IEEE Multimedia*, vol. 2, no. 2, pg. 10-18, 1995.

[WADDINGTON97] WADDINGTON, D. G., COULSON, G., HUTCHISON D. “Specifying QoS for Multimedia Communications within Distributed Programming Environments”. *Technical Report MPG-97-34*, Distributed Multimedia Research Group, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, 1997.

[WINDOWS95] Microsoft Corporation. *Microsoft Windows 95 Resource Kit*. Microsoft Press, 1995.

[WOLFINGER91] WOLFINGER, B., MORAN, M. “A Continuous Media Data Transport Service and Protocol for Real-time Communication in High Speed Networks”. *Proc. 2nd Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, Heidelberg, Alemanha, Novembro de 1991.

[YU93] YU, P. S., CHEN, M. S., KANDLUR, D. D. “Grouped Sweeping Scheduling for DASD-Based Multimedia Storage Management”. *Multimedia Systems*, vol. 1, pg. 99-109, 1993.

[ZHANG93] ZHANG, L., DEERING, S. E., ESTRIN, D., SHENKER, S., ZAPPALA, D. “RSVP: A New Resource ReSerVation Protocol”. *IEEE Network Magazine*, vol. 9, no. 5, Setembro de 1993.

[ZITTERBART92] ZITTERBART, M., STILLER, B., TANTAWY, A. “A Model for Flexible High-Performance Communication Subsystems”. *IEEE JSAC*, Maio de 1992.