4
Exemplo de Aplicação da Arquitetura

Com o objetivo de demonstrar como a arquitetura QoSOS pode ser aplicada na modelagem de um cenário real de provisão de QoS, este Capítulo descreve a implementação do suporte aos serviços integrados (Braden, 1994) sobre o subsistema de enfileiramento de pacotes de rede de um GPOS específico. Além disso, é mostrado como esse mesmo sistema deve ser modificado para aceitar a instanciação do framework de adaptação de serviços e, assim, possibilitar que as estratégias de admissão das categorias de serviço Intserv sejam substituídas em tempo de operação.

O modelo de serviços integrados foi escolhido por já ter sido alvo de outro trabalho produzido no Laboratório TeleMídia da PUC-Rio, que promoveu a definição de uma arquitetura para a provisão de QoS na Internet (Mota, 2001), baseada nos mesmos frameworks genéricos aqui utilizados. Além da própria definição da arquitetura, uma grande contribuição desse trabalho consistiu na construção de uma API de solicitação de serviços independente do modelo ou tecnologia empregada internamente no provedor de serviços. Naquela ocasião, foi proposto um cenário de uso da arquitetura constituído por sub-redes ATM e Ethernet, onde foram modelados os agentes de negociação de QoS do nível de rede associados aos roteadores e estações finais. A reserva de recursos, contudo, era localizada apenas na sub-rede ATM, pois as estações finais e roteadores dependiam de uma implementação de QoS para seus sistemas operacionais. Por isso, o cenário aqui proposto pode ser considerado um complemento daquele, sendo prudente o uso do mesmo modelo de QoS para a solicitação dos serviços a fim de integrar os dois projetos.

Nota-se que a aplicação do framework apenas sobre o enfileiramento de pacotes de rede não implica em uma implementação eficiente de provisão de QoS no domínio de sistemas operacionais, pois, como já discutido, a orquestração deve abranger outros recursos importantes, como a CPU. No entanto, considerar apenas

um dos subsistemas relevantes é suficiente para o propósito de demonstração de como os mecanismos descritos pela arquitetura podem ser aplicados.

Já o desenvolvimento do suporte à adaptação em um GPOS fica limitado, porque não é possível, por exemplo, criar um novo serviço submetendo todas as políticas de QoS sob a forma de componentes adaptáveis. Para permitir essa funcionalidade, seria necessário tornar flexível toda a estrutura de escalonamento dos recursos em questão definida no *kernel* desse GPOS. Assim, optou-se por adicionar ao *kernel* uma função antes ausente e demonstrar que ela pode ser substituída durante a operação do serviço. A extensão de um GPOS para adotar a arquitetura de adaptação de serviços é um trabalho ainda pouco explorado, mas muito interessante e deve incluir a reprogramação de grande parte dos subsistemas mais importantes.

O restante deste Capítulo encontra-se estruturado da seguinte forma. Primeiramente, o cenário implementado é apresentado de modo a estabelecer a infra-estrutura de rede, os equipamentos e as funcionalidades esperadas do sistema operacional utilizado. Em seguida, são relatados os procedimentos que foram necessários para a preparação da infra-estrutura do cenário, como a implementação de interfaces de baixo nível, modificações no *kernel* e configuração do sistema. Por fim, a descrição da instanciação dos frameworks é o assunto abordado, abrangendo detalhes sobre várias fases da provisão de QoS.

4.1 Descrição do cenário

A Figura 4.1 ilustra o cenário de uso, que é composto por várias sub-redes Ethernet interligadas por roteadores, formando um provedor Intserv. Pressupõe-se, ainda, que agentes de negociação RSVP (Braden, 1997) estão implementados em cada estação das sub-redes e nos roteadores.

Tanto as estações finais quanto os roteadores são gerenciados pelo sistema operacional Linux, com a versão 2.2.18 do *kernel*. Essa versão, além de ser muito estável, apresenta mecanismos para a manipulação do enfileiramento de pacotes junto às interfaces de saída para a rede. Denominado como controle de tráfego do

Linux, esse conjunto de mecanismos foi apresentado na Seção 2.5.6 e será utilizado para a reserva de recursos nos buffers de comunicação por ele controlados. Apesar de não ter importância para a descrição da instanciação, cada *kernel* está estendido pela arquitetura LRP descrita na Seção 2.5.5, de forma a tornar o recebimento de pacotes de rede mais eficiente.

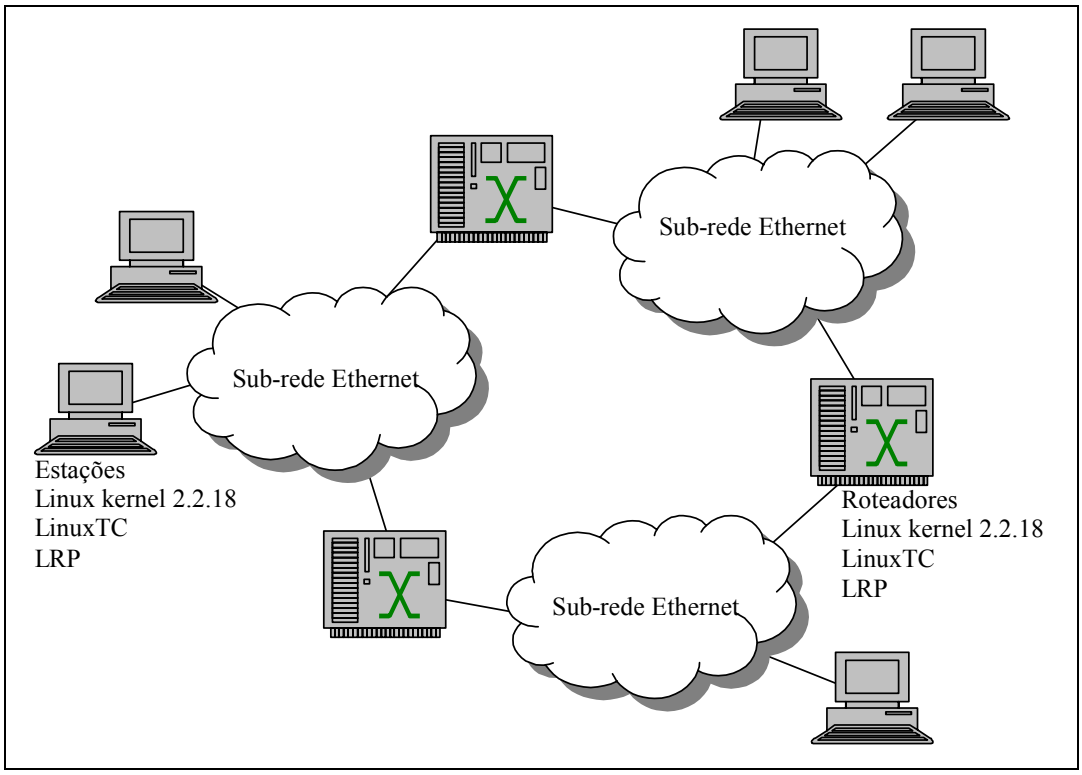

Figura 4.1 - Cenário de provisão de serviços Intserv

4.2 Infra-estrutura desenvolvida

Por intermédio do programa *tc*, o controle de tráfego do Linux (LinuxTC)[17] oferece métodos muito ricos, mas pouco dinâmicos, para a configuração dos seus componentes. Isso porque os desenvolvedores desse subsistema não projetaram uma interface de programação para que aplicações e protocolos de negociação pudessem configurar as características de desempenho da comunicação pela rede.

[17] É necessário configurar e recompilar o kernel para habilitar o LinuxTC, pois a configuração padrão não o inclui. Para isso, podem ser utilizadas as interfaces de configuração *menuconfig* (modo texto) ou *xconfig* (modo gráfico) localizadas junto ao código do kernel, por meio dos comandos *make menuconfig* ou *make xconfig*.

Recentemente, um projeto internacional de código aberto chamado *TCAPI* (Olshefski, 2001) foi criado pela IBM com o objetivo de preencher essa lacuna. Escrita em linguagem C, a versão 1.2 da interface *TCAPI* foi utilizada para a configuração do LinuxTC no cenário. Contudo, essa ferramenta mostrou-se deficiente por não disponibilizar muitas operações importantes oferecidas pelo LinuxTC (criação de filtros RSVP, remoção de qualquer disciplina, "classe" [18] ou filtro) e, ainda, por conter algumas falhas de programação. Devido à facilidade de acesso ao código da interface *TCAPI*, várias modificações puderam ser feitas até que o funcionamento ideal com as funções necessárias fosse atingido. Tais alterações foram submetidas à apreciação do administrador do projeto e todas foram aprovadas e integradas ao software original.

Como discutido na Seção anterior, é inviável a reprogramação de todo o controle de tráfego do Linux para que ele se torne integralmente adaptável e possa, assim, ser utilizado para descrever um exemplo de instanciação do framework de adaptação para a criação de novos serviços. Por isso, adotou-se a tática de se introduzir políticas anteriormente não suportadas pelo LinuxTC no *kernel* do sistema, que fossem adaptáveis desde sua concepção. Uma dessas políticas é o conjunto de estratégias de admissão adaptáveis (framework de negociação de QoS), que, se implementada no *kernel*, pode ser utilizada por controladores de admissão definidos no espaço do usuário (como é o caso) ou no próprio *kernel*. Dessa forma, optou-se por transferir as estratégias de admissão para o espaço do *kernel* através da utilização dos módulos do sistema descritos na Seção 2.6. Essas estratégias devem ser codificadas em um formato bem definido, como se segue (Figura 4.2):

- Função `strat_check()`: é o método acessível pelo controlador de admissão no espaço do usuário para a verificação da viabilidade da admissão em si. O parâmetro é o endereço de memória dos argumentos fornecidos através da interface `ioctl`, cujo conteúdo deve ser copiado para o espaço do *kernel* (função do *kernel* `copy_from_user()`).

[18] O elemento "classe" do LinuxTC aparecerá sempre entre aspas ao longo do texto, para que não seja confundido com as classes do modelo orientado a objetos em que se baseia a modelagem da arquitetura.

```
#include <linux/module.h> /* criacao de modulo */
#include <linux/kernel.h> /* programacao no espaco do kernel */
#include <linux/fs.h>     /* operacoes com arquivo (ioctl) */
#include <asm/uaccess.h>  /* copy_to/from_user */

#include <net/adm_strategy.h> /* administracao das estrategias */

#define STRAT_NAME "Exemplo de estratégia 1.0"

static int example_check(unsigned long);

struct adm_strategy example_strategy = { STRAT_NAME, example_check };

/* Funcoes visiveis pelo kernel, quando registradas via struct */
/* apenas *_check deve estar aqui */
static int example_check(unsigned long arg) {
    int feasible;
    ...
    return(feasible);
}

#ifdef MODULE
/* Funcoes executadas no carregamento e remocao do modulo */
int init_module(void) {
    if (register_strategy(&example_strategy)) {
        /* Erro ao inserir o modulo */
        ...
        return -EIO;
    }
    ...
    return 0;
}

void cleanup_module(void) {
    ...
    if (unregister_strategy(&maxflows_strategy)) {
        /* Erro ao remover o modulo, não há como retornar o erro*/
        ...
    }
}
#endif
```

Figura 4.2 - Modelo de implementação de estratégias de admissão através de módulos em linguagem C

- Função `init_module()`: responsável por realizar quaisquer operações necessárias no momento do carregamento do módulo, como solicitar que a estratégia seja registrada pelo *kernel* (nova função do *kernel* `register_strategy()`, cujo parâmetro é uma estrutura com o nome da estratégia e um ponteiro para a função `check()`). Com o registro da estratégia, o *kernel* se torna capaz de identificar uma chamada `ioctl()` a ela direcionada. Outros exemplos de operações que poderiam ser necessárias durante a inserção de um módulo são a inicialização de variáveis locais e a alocação de memória do espaço do *kernel*.

- Função `cleanup_module(void)`: responsável por executar quaisquer operações necessárias no momento da remoção de um módulo, como

remover o registro da estratégia (nova função do *kernel* `unregister_strategy()`, onde a mesma estrutura acima descrita é fornecida como parâmetro para a identificação da estratégia).

O *kernel* 2.2.18 foi modificado tanto para prover as funções de gerenciamento dos módulos de estratégias de admissão, como para oferecer uma interface de comunicação entre controlador e estratégia por meio de um novo dispositivo[19], batizado como `adm_strat`. O método `check()` das estratégias deve ser invocado pelo controlador por meio de uma chamada `ioctl()` nesse dispositivo. Tal função recebe como parâmetros um código sinalizando o método requerido e a estratégia correspondente, além de um ponteiro para os argumentos do método. Dessa forma, o controlador de admissão terá um trecho de código (em C) semelhante ao da Figura 4.3, ao testar efetivamente a viabilidade da reserva.

```
int feasible = 0;
...
{
      int fd;
      int cmd; /* Código para metodo e estrategia a serem utilizados */
      struct check_parameters params;  /* parametros do metodo check */
      int strat; /* estrategia a utilizar, conforme a
                    categoria de serviço solicitada */
      ...
      cmd = ADM_STRAT_CHECK * ADM_STRAT_FNBASE /* Codificacao de check */
               + strat;         /* Codificacao da estrategia requerida */
      if ((fd = open("/dev/adm_strat", O_RDONLY)) == -1) {
            /* Erro ao acessar o dispositivo */
            ...
      }
      else {
            /* Chamada da funcao check da estratégia no kernel,
               de forma codificada */
            feasible = ioctl(fd, cmd, &params);
            ...
      }
      ...
}
...
if (feasible) {
      /* Pode ser gerada uma pre-reseva */
}
...
```

Figura 4.3 - Trecho de código de um controlador de admissão para o acesso aos módulos de estratégias de admissão

As novas funcionalidades foram incluídas no utilitário de configuração do *kernel*, na porção referente ao LinuxTC:

[19] No Linux, dispositivos são tratados como arquivos, podendo ser acessados por chamadas como `open()`, `read()` e `ioctl()`.

```
[*] Flow Admission Control Strategies (NEW)
      (6) Maximum number of concurrent strategies (NEW)
      < > SimpleSum Admission Strategy (NEW)
      < > MaxUsers AdmissionStrategy (NEW)
```

4.3 Instanciação da arquitetura

A instanciação da arquitetura QoSOS sobre o cenário de uso proposto levou ao desenvolvimento de duas interfaces de programação para aplicações (API) em Java, através das quais foram modelados muitos dos mecanismos descritos. A primeira API, denominada *QueuingQoS*, oferece a solicitação de serviços com QoS por parte dos agentes de negociação de rede, promovendo controle de admissão e criação de recursos virtuais sobre o subsistema de enfileiramento de pacotes de rede. A segunda API, denominada *AdaptQoS*, permite que ações de adaptação sejam requisitadas pelos administradores de sistemas operacionais especificamente sobre as categorias de serviço acessíveis pela interface *QueuingQoS*.

A Figura 4.4 permite uma visão geral dessa instanciação, ilustrando como os componentes da interface *QueuingQoS* interagem e modificam a estrutura do controle de tráfego, ações essas demarcadas por linhas tracejadas. A interface *AdaptQoS*, por sua vez, tem seu fluxo de controle caracterizado pelas linhas pontilhadas, agindo apenas sobre a estrutura adaptável de estratégias de admissão. O fluxo dos pacotes de rede entre as filas do controle de tráfego está representado pelas linhas contínuas.

4.3.1 Iniciação do sistema

As tarefas pertinentes à fase de iniciação do sistema estão implementadas por um programa em separado, acionado no momento da inicialização do sistema (boot). Ele é capaz de montar a estrutura inicial da árvore de recursos virtuais sobre as filas do nível de enlace (denominação dada às filas controladas pelo LinuxTC) que fazem parte do subsistema de rede das estações. As parcelas de alocação da largura de banda dessas filas associadas às categorias de serviço

(inclusive melhor esforço, que não tem qualquer garantia negociada) são passadas pela linha de comando do programa.

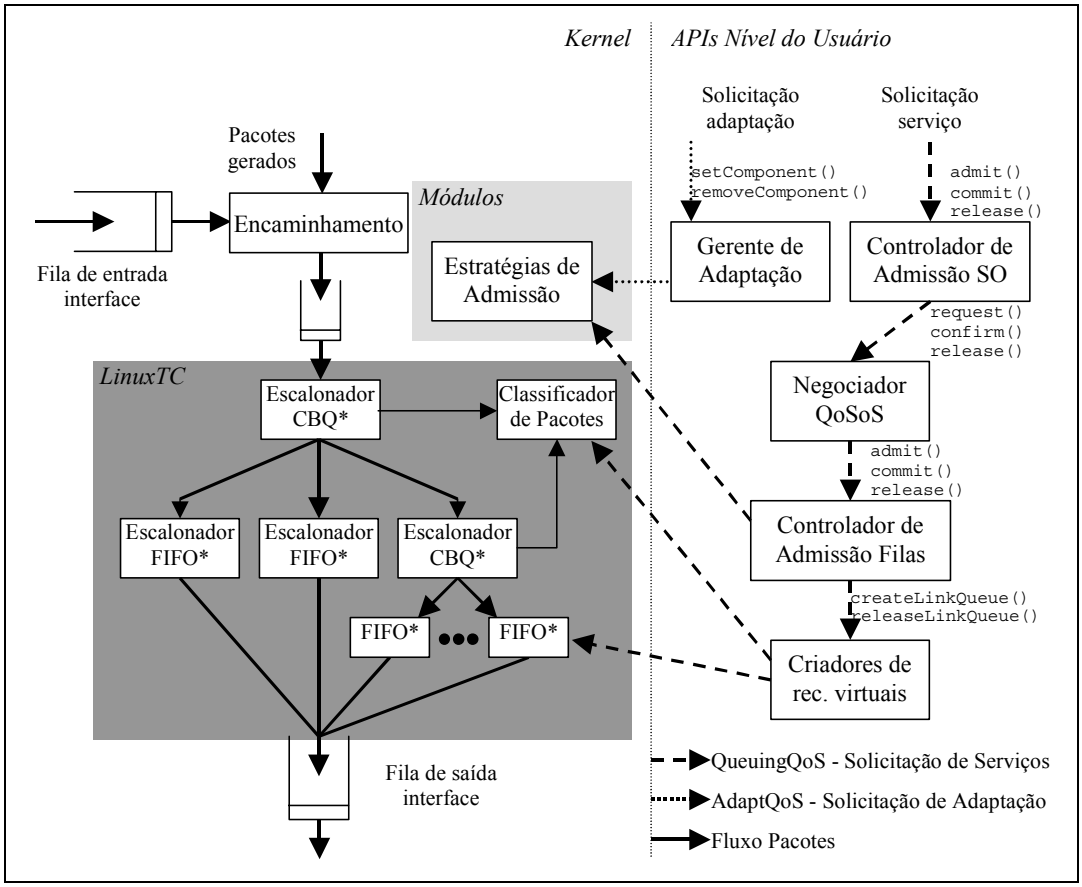

Figura 4.4 - Visão geral da implementação

A Figura 4.5 ilustra um exemplo de configuração inicial da árvore de recursos virtuais, onde o escalonador raiz utiliza a estratégia de escalonamento CBQ (Floyd, 1995), capaz de atribuir parcelas de largura de banda aos seus escalonadores filhos. A categoria de serviço garantido, detentora de 30% da capacidade da fila de enlace, também foi associada à estratégia CBQ, para redistribuir sua largura de banda entre seus fluxos. A categoria de carga controlada tem disponível uma parcela de 50% do enlace e utiliza a estratégia FIFO para o escalonamento dos pacotes, já que não fornece garantias severas de alocação de recursos. Por último, a categoria de melhor esforço será utilizada pelos fluxos que não solicitaram a provisão de serviços com QoS, que podem juntos utilizar apenas 20% do recurso. As estratégias de admissão relacionadas a cada categoria de serviço serão descritas posteriormente, na Seção 4.3.5.

Nota-se que o programa de iniciação do sistema poderia estar utilizando um gerenciador de adaptabilidade para a criação dos serviços, mas não foi assim feito devido à necessidade de suporte à adaptação por parte de todas as políticas de QoS distribuídas entre o LinuxTC e a interface do usuário.

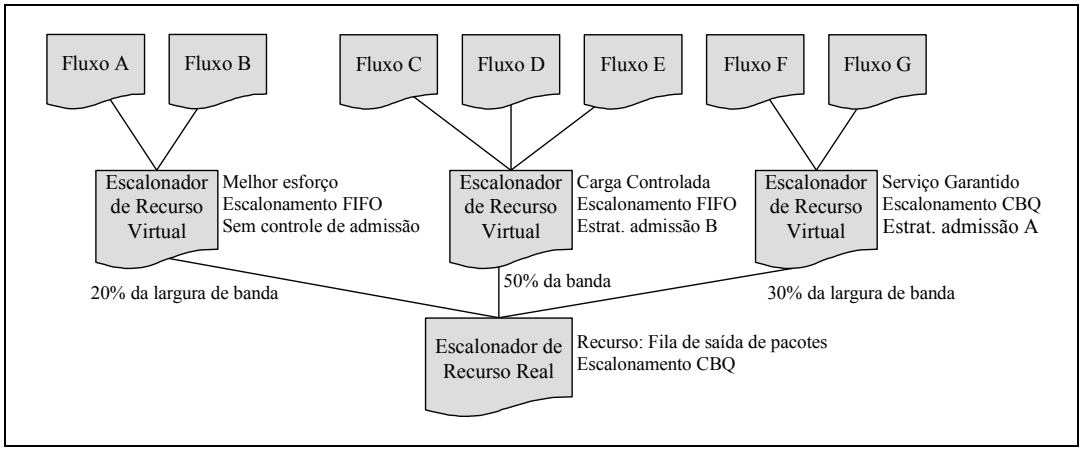

Figura 4.5 - Árvore de recursos virtuais da fila de pacotes

4.3.2 Parametrização de serviços

O framework de parametrização de serviços foi instanciado seguindo a proposta de (Mota, 2001), com o objetivo de manter a coerência necessária entre as duas implementações. A Figura 4.6 ilustra a hierarquia de serviços e de parâmetros originada pela especialização do referido framework.

As categorias de serviço garantido e de carga controlada são representadas pelas classes `GuarServiceCategory` e `CLServiceCategory`, respectivamente. O parâmetro RSpec (reservation specification) deve ser adicionado apenas a objetos da classe `GuarServiceCategory` por denotar os requisitos de qualidade a serem reservados pelo sistema, compreendendo a taxa de dados (R, em bytes/s) e o termo de folga (s, referente à diferença entre o retardo obtido a partir da reserva de R e o retardo máximo permitido, em milissegundos). Já o parâmetro TSpec (traffic specification) é de comum utilização por ambas categorias e descreve a caracterização do tráfego gerado pelo o usuário. O valor de r corresponde à taxa de dados em bytes/s, b é o tamanho do bucket em bytes, p é a taxa de pico em bytes/s, m denota o tamanho mínimo da unidade policiada em bytes e M é o tamanho máximo do pacote em bytes.

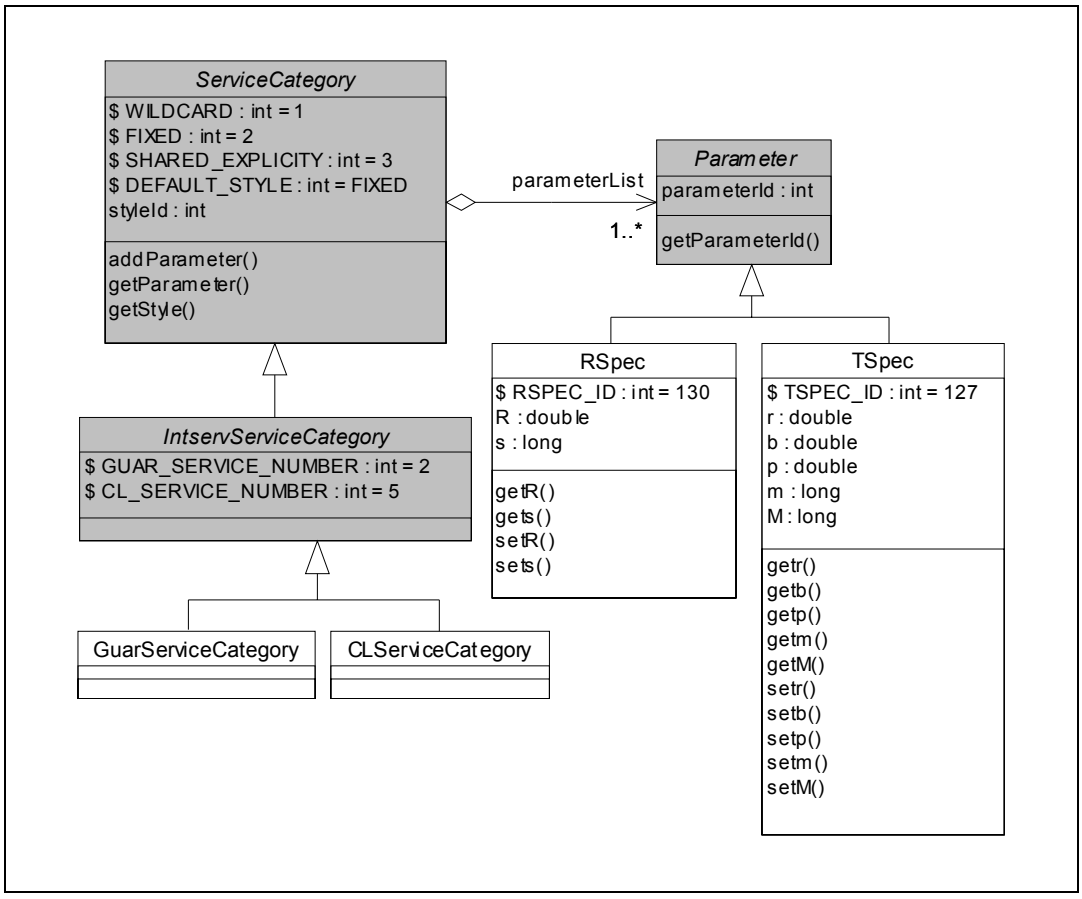

Figura 4.6 - Aplicação do framework para Parametrização de Serviços

Não há a necessidade de definição de outros parâmetros de maior relacionamento com o recurso a ser alocado, já que o algoritmo CBQ e os policiadores do LinuxTC são configurados por valores de mesmo significado que aqueles definidos pelas estruturas RSpec e TSpec.

4.3.3
Compartilhamento de recursos

Os mecanismos do controle de tráfego do Linux puderam ser diretamente modelados pelo framework de escalonamento de recursos. A instanciação da arquitetura ilustrada pela Figura 4.7 reflete, inclusive, os nomes dos métodos encontrados internamente no LinuxTC, na forma como foram definidos em (Almesberger, 1999).

A classe `DeviceQueuingDiscipline` representa a disciplina de enfileiramento raiz associada a uma interface de rede e responsável por receber os

comandos `wakeup()` para iniciar a seqüência de escalonamento de pacotes (função `dequeue()`). Todas as outras disciplinas relacionadas com a mesma interface são modeladas pela classe `InnerQueuingDiscipline` e também utilizam a função `dequeue()` para dar prosseguimento ao escalonamento.

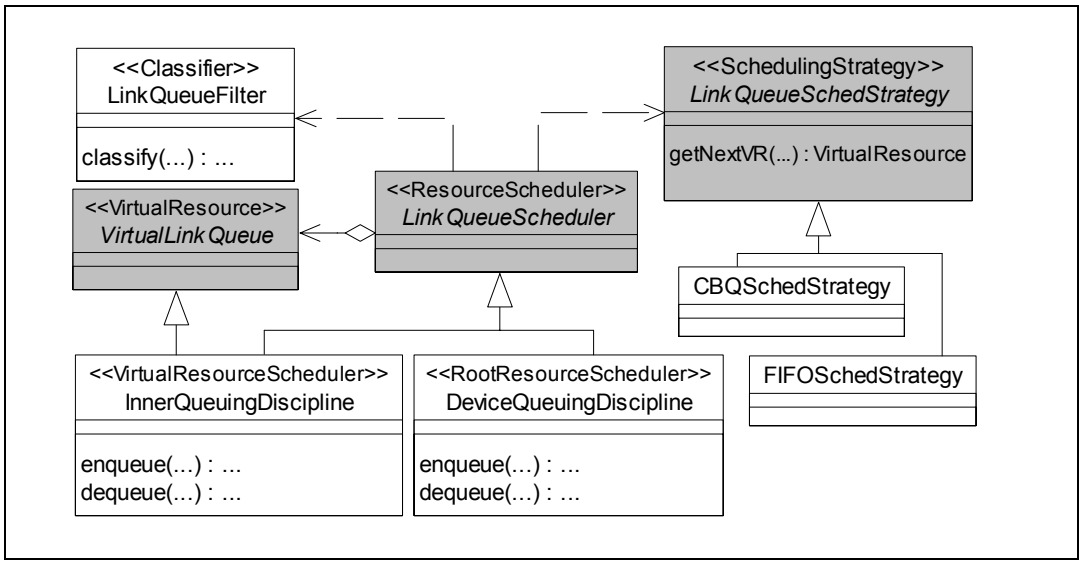

Figura 4.7 - Aplicação do framework de escalonamento de recursos

Quando um pacote é totalmente processado pela pilha de protocolos de rede, é colocado à disposição da disciplina de enfileiramento associada à interface de saída selecionada pelos procedimentos de encaminhamento. A seguir, essa disciplina raiz deve submeter o pacote à análise dos filtros classificadores, modelados por `LinkQueueFilter`, que identificam a "classe" a que pertence o pacote. A disciplina raiz, então, invoca o método `enqueue()` da disciplina folha da "classe" indicada. A seqüência de métodos `enqueue()` continua até que seja encontrada uma disciplina sem "classes", como as disciplinas regidas pela estratégia FIFO.

Nota-se que o elemento "classe" representa apenas uma conexão entre as disciplinas, definindo a parcela de alocação atribuída a cada uma delas. Se uma "classe" não possui uma disciplina folha, a estratégia FIFO é considerada como sua disciplina. As estratégias de escalonamento oferecidas pelo LinuxTC que são utilizadas na provisão de serviços integrados estão representadas na modelagem pelas classes `CBQSchedStrategy` e `FIFOSchedStrategy`.

Já o elemento filter do LinuxTC difere um pouco do mecanismo de classificação modelado pela arquitetura QoSOS, uma vez que ele agrega as funções de classificação e policiamento, como será visto posteriormente. Ao mesmo tempo, cada filtro possui uma única regra para a classificação e um único perfil de tráfego para policiamento.

A seqüência de chamadas `dequeue()` define o escalonamento de cada pacote armazenado em todas as disciplinas. Ao receber a sinalização pelo método `wakeup()`, a disciplina raiz seleciona, de acordo com sua estratégia, aquela "classe" que deve ser atendida e faz a chamada da função `dequeue()` para a disciplina folha correspondente. Quando essa seqüência chega a uma disciplina sem "classes", imediatamente é selecionado o pacote a ser entregue ao driver da interface.

Uma característica importante do LinuxTC é que os pacotes não são copiados de uma disciplina para a outra, como se estivessem sendo transmitidos entre filas. Na realidade, cada disciplina possui uma fila de ponteiros para os *skbufs*[20], os quais permanecem armazenados nos mesmos endereços durante todo o processo de enfileiramento.

Entre os mecanismos modelados pela instanciação do framework de alocação de recursos, ilustrada pela Figura 4.8, somente os componentes de criação de recursos virtuais não estão presentes no subsistema de controle de tráfego do Linux e precisaram ser implementados no espaço do usuário, como parte da API de solicitação de serviços. Foram instanciados dois componentes de criação de modo a corresponderem cada um a uma categoria de serviço específica, já que as necessidades de reserva entre elas são diferentes.

A alocação de recursos para os fluxos que requerem a categoria de serviço garantido é realizada por um objeto da classe `GuaranteedLQFactory`, através do método `createGuarLink()`. Essa operação pode ser dividida em duas etapas: a criação dos filtros de classificação e policiamento e a reserva efetiva da

[20] Skbufs são os buffers de memória para armazenamento dos pacotes de rede no espaço de endereçamento do kernel, muito semelhantes aos mbufs utilizados em outros sistemas operacionais.

largura de banda solicitada. Um único elemento "filtro" do LinuxTC realiza tanto o casamento de sua regra com os dados contidos no cabeçalho, como o policiamento dos pacotes casados seguindo sua regra de perfil de tráfego. Assim, o componente de criação da categoria de serviço garantido solicita a criação de um filtro de classificação e policiamento (através da interface TCAPI, pois o objeto está definido no espaço do usuário) pela função `change()` de um objeto da classe `LinkQueueFilter`. A reserva da largura de banda é feita pela criação de mais uma "classe" junto à disciplina de enfileiramento encarregada de escalonar os pacotes da categoria de serviço garantido. Isso é feito (novamente por intermédio da interface TCAPI) através da função `change()` presente nas especializações da classe `LinkQueueScheduler`.

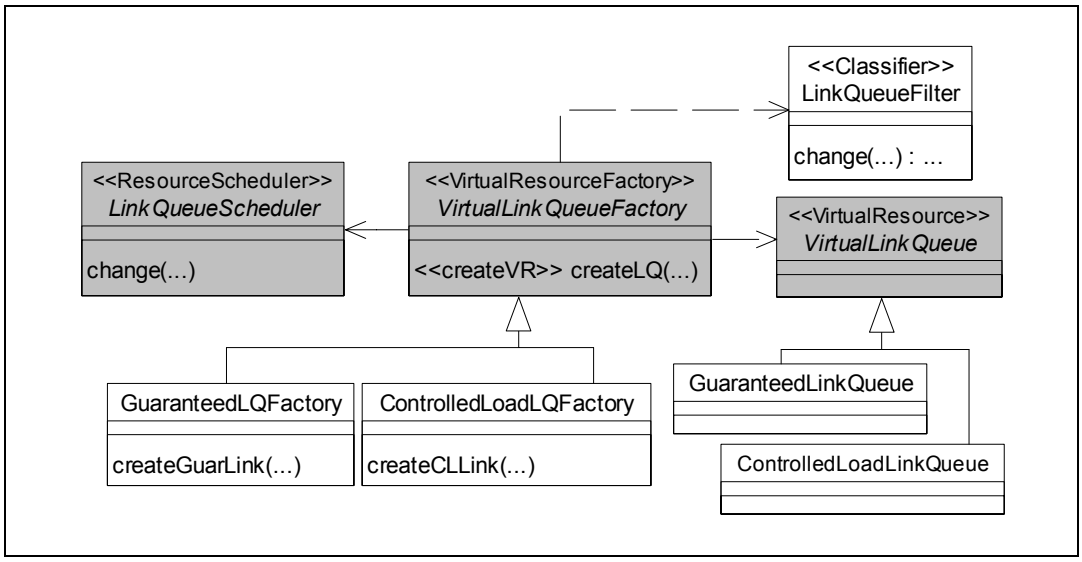

Figura 4.8 - Aplicação do framework para Alocação de Recursos

Com relação ao componente de criação de recursos virtuais associado à categoria de serviços de carga controlada, é suficiente que o filtro de classificação e policiamento seja configurado, pois não existe a reserva efetiva de largura de banda para fluxos dessa categoria.

4.3.4 Solicitação de serviços

O processo de solicitação de serviços ao subsistema de enfileiramento de pacotes do sistema operacional somente acontece depois que os agentes de negociação de rede estabeleceram as parcelas de participação na orquestração de

recursos de cada estação envolvida. Por isso, a API *QueuingQoS* disponibiliza os métodos públicos do controlador de admissão QoSOS para que os agentes de negociação do nível de rede possam solicitar serviços com QoS. O controlador de admissão QoSOS aqui instanciado provê apenas uma interface entre o usuário, o negociador QoSOS e o controlador de admissão das filas de comunicação. Conforme proposto para este cenário de uso, nenhum tipo de orquestração de recursos interna ao sistema operacional é realizada pelo negociador.

As primitivas que compõem a API *QueuingQoS* estão especificadas na Tabela 4.1. As classes `QoSSession` e `FilterSpec` foram definidas por (Mota, 2001) e reaproveitadas, sendo que a primeira representa as informações sobre o destino de um fluxo e a segunda é a identificação da origem de um fluxo.

Solicitação de Serviço `long admit(QoSSession, InetAddress, FilterSpec, ServiceCategory)`	
`QoSSession`	Sessão de QoS
`InetAddress`	Endereço da interface local a ter recursos reservados
`FilterSpec`	Identificação do fluxo especificado
`ServiceCategory`	Categoria de serviço especificada
`Retorno`	Se maior que 0, identificador para a pré-reserva feita pelo controlador de admissão, caso contrário significa inviabilidade ou falha da solicitação
Confirmação de Serviço `boolean commit(long)`	
`Long`	Identificador da pré-reserva a ser confirmada
`Retorno`	Sucesso (true) ou falha (false) da confirmação
Encerramento do Serviço `boolean release(QoSSession, InetAddress, FilterSpec, ServiceCategory)`	
`QoSSession`	Sessão de QoS
`InetAddress`	Endereço da interface local a ter recursos liberados
`FilterSpec`	Identificação do fluxo especificado na solicitação
`ServiceCategory`	Categoria de serviço especificada na solicitação
`Retorno`	Sucesso (true) ou falha (false) da liberação dos recursos

Tabela 4.1 - Primitivas da API de solicitação de serviços

O método `release()` é o único que não foi mencionado na descrição da arquitetura. A partir dele, o usuário solicita a liberação de todos os recursos alocados para a provisão de QoS sobre o fluxo identificado pelos parâmetros `QoSSession` e `FilterSpec`. O retorno informa se a operação foi bem sucedida ou não.

4.3.5
Estabelecimento de contratos de serviço

A Figura 4.9 ilustra a instanciação do framework de negociação de QoS para o cenário proposto. Para atender uma solicitação de serviço feita por meio da primitiva `admit()`, o controlador de admissão do sistema operacional (classe `QoSOSAdmissionController`) encaminha os parâmetros fornecidos ao negociador de QoS (classe `QoSOSNegotiator`). O negociador deve repassar os parâmetros solicitados ao controlador de admissão das filas de enlace (classe `LinkQueuingAdmissionController`, por meio do método `admit()`). Isso pode ser feito diretamente, como dito anteriormente.

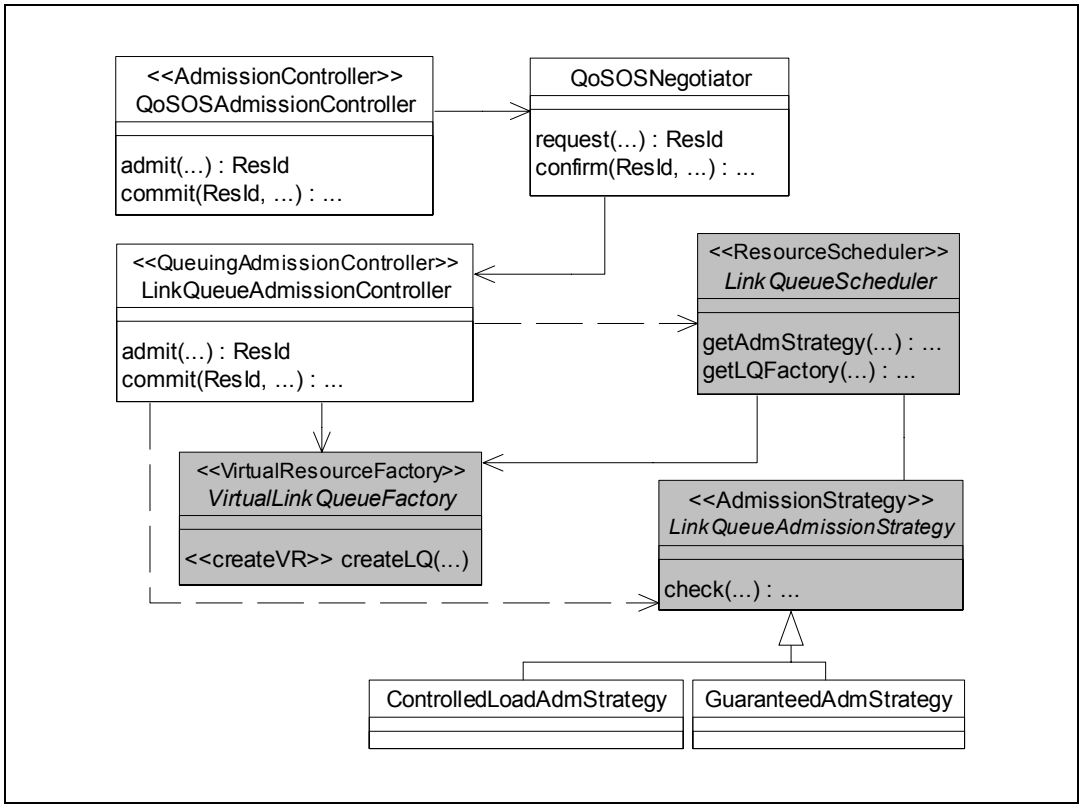

Figura 4.9 - Aplicação do framework para Negociação de QoS

O controlador de admissão das filas verifica a categoria de serviço solicitada e invoca o método `check()` da estratégia de admissão correspondente (classe `LinkQueueAdmissionStrategy`). Esse método analisa os atuais parâmetros de desempenho dos recursos e os compara com a necessidade de QoS solicitada. Se ficar concluído que a solicitação é viável, o controlador de admissão é informado e gera um identificador de pré-reserva, que será utilizado posteriormente na confirmação do serviço. O controlador de admissão das filas retorna o

identificador gerado ao negociador, que repassa ao controlador de admissão QoSOS.

Com o objetivo de confirmar o serviço, o controlador de admissão QoSOS deve invocar o método `confirm()` do negociador, informando o identificador da pré-reserva, que será repassado pelo negociador ao controlador de admissão das filas, utilizando o método `commit()`. Se o identificador ainda for válido, os dados da pré-reserva são recuperados e o componente apropriado de criação de recursos virtuais (uma especialização da classe `LinkQueueFactory`) é acionado, como descrito no framework de alocação de recursos. Essa seqüência de chamadas é ilustrada resumidamente pela Figura 4.4 apresentada anteriormente.

É interessante observar que um trecho do código do controlador de admissão implementado é executado periodicamente para a verificação da validade dos identificadores de reserva. Quando é constatada a expiração de um identificador, todos os dados sobre a reserva são removidos e não mais considerados para a admissão de solicitações futuras.

As estratégias de admissão disponíveis inicialmente correspondem às classes `GuaranteedAdmStrategy` e `ControlledLoadAdmStrategy`. Para que a admissão de um fluxo de serviço garantido seja aprovada, a taxa de dados (R) deve estar disponível no escalonador de recurso virtual. Essa estratégia é conhecida como soma simples e corresponde à estratégia A, especificada na estrutura da árvore de recursos virtuais, apresentada pela Figura 4.5. Já a admissão para carga controlada é regida pela estratégia B da figura, que consiste em testar se a soma dos parâmetros r de todos os fluxos anteriormente admitidos mais o parâmetro r do fluxo que solicitou o serviço não excede a largura de banda alocada à categoria de serviço. Essa estratégia é equivalente à soma simples, mas leva em conta os parâmetros de caracterização do tráfego.

4.3.6 Manutenção de contratos de serviço

Durante a fase de manutenção de serviço, os filtros de policiamento do controle de tráfego do Linux agem de modo a identificar os fluxos submetidos

pelos usuários que excedem a caracterização de tráfego informada no momento da solicitação do serviço. Os filtros são capazes de identificar os pacotes não-conformes e, imediatamente, tomar uma das três decisões, configuradas no momento da criação do filtro: ignorar a notificação e enviar o pacote; promover uma reclassificação do pacote, por exemplo, para uma "classe" de menor prioridade; ou simplesmente descartá-lo.

4.3.7 Adaptação de serviços

A Figura 4.10 ilustra a instanciação do framework de adaptação de serviços sobre a infra-estrutura desenvolvida dentro do *kernel*, descrita anteriormente. A classe `LinuxQoSAdaptationManager` representa o gerente de adaptação implementado no espaço do usuário e sua funcionalidade está restrita à inserção e substituição de módulos que implementam estratégias de admissão. Tais módulos são abstraídos sob a forma de componentes adaptáveis através das classes `AdmissionStrategyComponent`, `KernelModulePort` e `ObjectFile`.

Quando o administrador do sistema ou qualquer outro mecanismo de gerência autorizado solicita a inserção/substituição (`setComponent()`) de um componente através da API *AdaptQoS*[21], o gerente de adaptação instanciado se certifica de que a porta de adaptação é o subsistema de módulos do *kernel*, para o qual será submetida a implementação do componente, por meio da chamada `ins_mod`. Esse subsistema possui uma verificação simples de segurança, modelada pela classe `LinuxAdaptationSecurityManager`, que consiste na autenticação das capacidades atribuídas ao solicitante, seja ele um usuário ou um programa (`função capable()`). Além disso, uma função de sondagem (`modprobe()`) pode ser explorada pela classe `LinuxComponentProber`, mas não foi utilizada na implementação.

A associação entre as categorias de serviço e as estratégias de admissão é dada pela posição que o módulo ocupa na lista de módulos carregados (2 para as

[21] A interface *AdaptQoS* corresponde aos métodos públicos disponibilizados pela classe `LinuxQoSAdaptationManager`.

estratégias do serviço garantido e 5 para as estratégias de carga controlada). Assim, de acordo com a categoria informada numa requisição de substituição, o gerente pode remover o módulo já instalado na posição correspondente, substituindo-o pelo novo módulo.

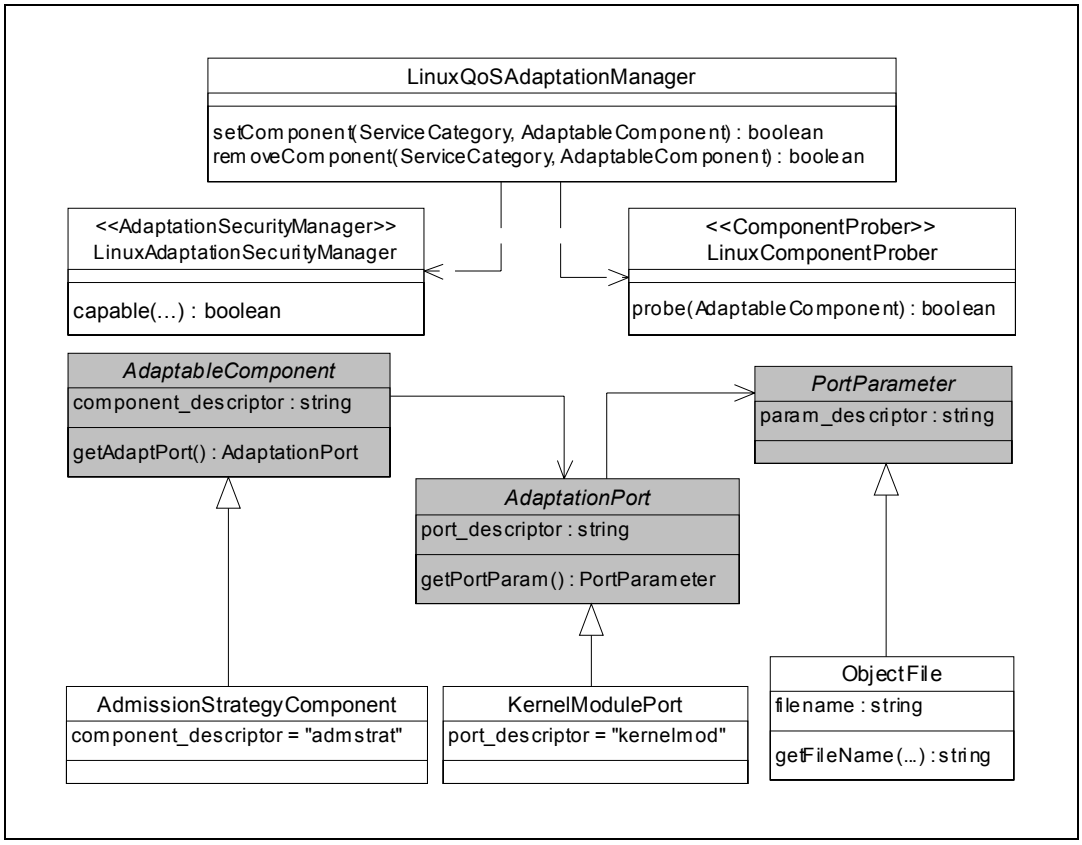

Figura 4.10 - Aplicação do framework para Adaptação de Serviços

4.4 Resumo do Capítulo

Neste Capítulo, foi descrito um exemplo real de aplicação da arquitetura QoSOS em um sistema operacional de propósito geral. Promover um GPOS a um cenário de uso da arquitetura trouxe a necessidade de se efetuar algumas modificações no código do *kernel* e da API de configuração do subsistema alvo da provisão. Deve-se ressaltar que esta não foi uma implementação eficiente, pois outros subsistemas devem participar da orquestração de recursos com o intuito de prover maior confiabilidade e predição sobre o comportamento do sistema como um todo. O objetivo de demonstrar a arquitetura, contudo, foi alcançado, pois o

ambiente desenvolvido permitiu a demonstração de como os frameworks da arquitetura podem ser aplicados em um cenário real.

Esse cenário foi propositadamente constituído como um complemento a um projeto anterior desenvolvido no Laboratório TeleMídia, ocasião em que não foram considerados os mecanismos de provisão de QoS nos sistemas operacionais agora implementados. O reaproveitamento de parte da interface de programação proposta por aquele trabalho possibilitou tanto a fácil integração entre os negociadores de diferentes níveis como o desenvolvimento de uma nova API genérica, estendida para a solicitação de serviços com QoS a sistemas operacionais.