3 DESCRIÇÃO DO FRAMEWORK

Este capítulo apresenta os aspectos relacionados à mobilidade que foram incorporados ao trabalho definido por Gomes (1999). Tais aspectos envolvem duas questões principais, a predição da mobilidade e o estabelecimento de reservas antecipadas.

O modelo de provisão de QoS proposto por Gomes é representado por meio de *frameworks* arquiteturais. A motivação para se representar conceitualmente esse modelo e as modificações propostas neste trabalho através de *frameworks* reside na facilidade de identificação dos pontos de flexibilização através dos quais o modelo genérico pode ser especializado, pelo projetista, para cenários específicos, de forma a tornar possível a validação da arquitetura proposta. A adoção de *frameworks* (Fayad et al., 1999) garante a reutilização e, ao mesmo tempo, permite a personalização do modelo apresentado em diferentes contextos, o que se deve à sua capacidade de modularizar, combinar e estender os componentes de sistemas de software.

Os *frameworks para provisão de QoS em redes móveis sem fio* propostos neste trabalho adotam o modelo de provisão de qualidade de serviço apresentado por Gomes (1999) introduzindo mais um elemento, que será o responsável por oferecer suporte à mobilidade do usuário. Tem-se então os seguintes *frameworks*:

- *Framework* para Parametrização de Serviços.
- *Frameworks* para Compartilhamento de Recursos.
- *Frameworks* para Orquestração de Recursos.
- *Framework* para Gerenciamento de Mobilidade.

Na descrição conceitual dos frameworks foi adotada a linguagem UML (1997) (*Unified Modeling Language*) para sua especificação, documentação e visualização, sendo empregado o paradigma orientado a objetos na sua modelagem e implementação. Nos diagramas de classe foram utilizadas cores distintas para diferenciar as classes-base (em cinza) daquelas que representam possíveis instanciações dos pontos de

flexibilização desses elementos (em branco). Nesses diagramas, foi ainda utilizada uma grafia diferenciada para representar as classes abstratas (itálico) e as concretas (regular).

3.1 Framework para Parametrização de Serviços

O mecanismo de provisão de qualidade de serviço se baseia na representação correta dos parâmetros envolvidos na definição da QoS. Os parâmetros podem estruturar as informações relacionadas, por exemplo, com a descrição do estado interno do sistema, a caracterização da carga dos usuários e a especificação da QoS desejada pelos mesmos. Na definição de um contrato de serviço, espera-se que o usuário requisitante forneça a caracterização do tráfego correspondente aos fluxos que irá introduzir no sistema e os parâmetros de QoS a serem aplicados pelo provedor sobre os referidos fluxos.

Em ambientes móveis sem fio, existem dois fatores que influenciam decisivamente a manutenção dos parâmetros de QoS definidos durante a negociação do contrato de serviço: a instabilidade das conexões sem fio, apresentando desvanecimentos constantes do enlace, e a própria mobilidade dos usuários, o que resulta em mudanças freqüentes na topologia da rede.

Nos mecanismos tradicionais de reserva de recursos, os parâmetros que representam a QoS desejada pelo usuário são representados de forma pontual, ou seja, cada um desses valores é definido por um único ponto em um espaço n-dimensional, com as coordenadas definindo as características do serviço. O sistema irá atender ou não a uma dada requisição com base nos valores dos descritores que foram associados à reserva, sem que esses valores possam ser alterados de uma forma dinâmica em resposta à disponibilidade de recursos verificada na rede. Como se pode observar, não existe uma flexibilidade quanto à definição dos valores que representam os parâmetros de QoS. O que se tem, em alguns casos, são refinamentos dos algoritmos de controle de admissão para forçar uma certa tolerância quanto aos valores atribuídos à reserva. Entretanto, os algoritmos utilizados atualmente possuem um baixo grau de complexidade para evitar uma carga adicional de processamento nos níveis inferiores da hierarquia de provisão, onde se lida diretamente com os

recursos. Com essa limitação, não há como responder às constantes flutuações na disponibilidade de recursos, que são peculiares aos enlaces sem fio.

Esse problema pode ser ilustrado nos serviços integrados através de objetos *TSpec*, que representam os descritores de tráfego das classes de serviço garantido e de carga controlada. Em um objeto *TSpec*, o tráfego a ser gerado é representado, de forma pontual, por uma tupla contendo cinco parâmetros (*r*, *b*, *p*, *m*, *M*) que descrevem, respectivamente, a taxa média de dados, o tamanho máximo da rajada de dados, a taxa de pico, a unidade mínima de policiamento[24] e o tamanho máximo do pacote a ser transmitido. Ao aceitar, por exemplo, uma reserva definida pelo serviço garantido, a rede está se comprometendo a oferecer o serviço exclusivamente dentro dos valores especificados por essa tupla, sem admitir variações ocasionadas por alterações na disponibilidade de recursos da rede, que podem ocorrer durante a provisão do serviço.

Em ambientes móveis existe a necessidade de se definir os valores assumidos pelos parâmetros como intervalos de níveis aceitáveis de QoS, para que flutuações na disponibilidade de recursos sejam tratadas transparentemente, sem que os mecanismos de renegociação sejam acionados, podendo ser feita a sintonização da QoS, evitando dessa forma a interrupção no fornecimento do serviço. Esse procedimento traz um benefício adicional, já que, devido à escassez de largura de banda nesses ambientes, o tráfego de mensagens de controle deve ser minimizado.

Como mencionado na Seção 1.3, o uso de intervalos de QoS oferece um mecanismo para se efetuar adaptações na QoS solicitada pelo usuário, fazendo com que as reservas se ajustem às flutuações na disponibilidade de recursos na rede. Para ilustrar o que foi dito, retomaremos a categoria de serviço garantido do exemplo anterior. Os valores atribuídos à tupla (*r*, *b*, *p*, *m*, *M*) poderiam ser representados por intervalos de tolerância, os quais definiriam uma faixa de valores aceitáveis [r_{min}, r_{max}], [b_{min}, b_{max}], etc., que descreveriam a reserva. Essas faixas de valores seriam utilizadas em possíveis negociações entre os elementos do sistema. Desse modo, uma reserva

[24] A variável *m* descreve o tamanho mínimo permitido aos pacotes de dados para que as ações de policiamento associadas aos fluxos possam ser aplicadas. Caso um pacote tenha um tamanho inferior ao valor definido em *m*, ele será contado como tendo *m* bytes. As ações de policiamento são aplicadas aos pacotes que excedem a taxa de tráfego pré-definida em um contrato de serviço, fazendo com que esses pacotes sejam descartados ou reclassificados com outro nível de prioridade.

não seria rejeitada caso os valores máximos correspondentes à tupla (*r*, *b*, *p*, *m*, *M*) não pudessem ser admitidos. Esses valores poderiam ser degradados, tendo como limite os valores mínimos definidos, até se adequarem aos percentuais de recursos disponíveis na rede.

Usando o conceito de intervalos de QoS, é possível utilizar os parâmetros negociados de uma forma mais eficiente, maximizando, desse modo, o desempenho do provedor de serviços, ao mesmo tempo em que se reduz o número de interrupções por *handoffs* e rejeições de novas conexões.

3.1.1 Componentes do Framework para Parametrização de Serviços

A Figura 3.1 retrata o *framework para parametrização de serviços* através das classes e dos respectivos relacionamentos que o constituem. As classes abstratas *ServiceCategory* e *Parameter* representam, respectivamente, as hierarquias de categorias de serviços e de derivação de parâmetros. O objetivo desse *framework* é desacoplar as categorias de serviços dos parâmetros que as constituem, garantindo dessa forma uma maior flexibilidade na descrição dessas categorias. Através do *pattern* estrutural Bridge (Gamma et al., 1995), as categorias de serviços são representadas como conjuntos de parâmetros, o que garante sua extensibilidade. Como se pode ver na Figura 3.1, as classes *ServiceCategory* e *Parameter* estão associadas pelo relacionamento de agrupamento *parameterList*. Os métodos `getParameter()` e `addParameter()`, disponibilizados em *ServiceCategory*, são responsáveis por, respectivamente, retornar uma instância de subclasses de *Parameter*, identificadas pelo atributo `param_descriptor`, e adicionar novos parâmetros que definirão novas categorias de serviços, ou mesmo adequar-se às já existentes. Assim como os parâmetros, as categorias de serviços são identificadas por descritores representados pelo atributo `categoryDescriptor`.

A extensão introduzida por Mota et al. (2001) inclui na classe o atributo `style`, que define o estilo de compartilhamento de uma reserva, podendo ser acessado através dos métodos `setStyle()` e `getStyle()` definidos na classe *ServiceCategory*. Um estilo pode representar um serviço compartilhado (*shared*) ou não (*fixed*) entre um agregado de fluxos especificado pelo usuário.

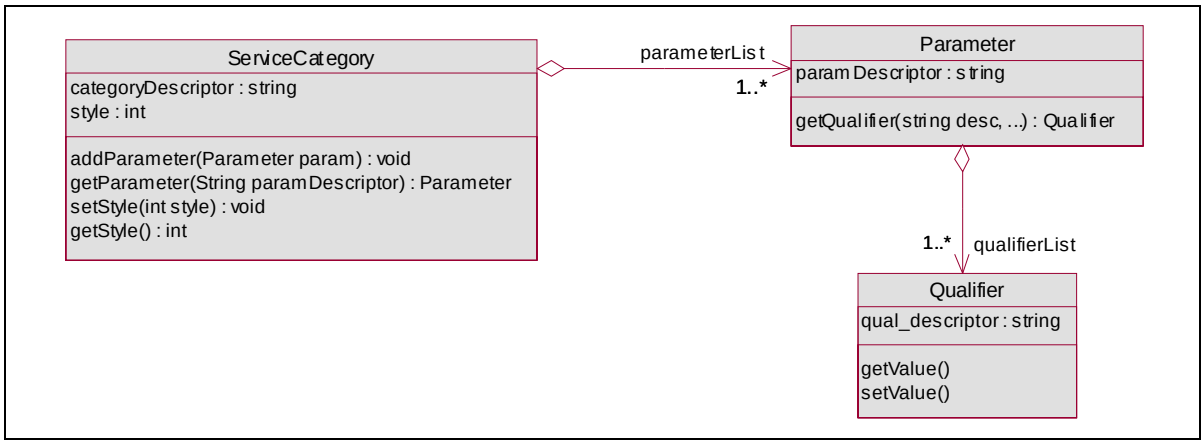

Figura 3.1 *Framework* para Parametrização de Serviços

A classe abstrata *Qualifier* modela os qualificadores de parâmetros. Mais uma vez o *pattern* estrutural Bridge é utilizado, agora para representar os parâmetros como um conjunto de qualificadores, o que se consegue através do relacionamento de agrupamento `qualifierList`. Essa representação oculta dos mecanismos de provisão de QoS certos detalhes referentes à estruturação dos parâmetros, tornando tais mecanismos mais independentes desses parâmetros. Isso é obtido pelo método `getQualifier()`, definido na classe *Parameter*, que retorna uma instância de uma subclasse de *Qualifier* identificada através do atributo `qual_descriptor`. A utilização de qualificadores de parâmetros (*Qualifier*) é identificada como um ponto de flexibilização do *framework*, o qual deve ser preenchido para representar os intervalos de QoS.

3.1.2 Exemplo de Aplicação do Framework para Parametrização de Serviços

A seguir será apresentado um exemplo de aplicação do *framework para parametrização de serviços* em um provedor que implementa o modelo de serviços integrados em um ambiente que oferece suporte à mobilidade dos usuários. O cenário modelado representa uma rede infra-estruturada na qual o encaminhamento de pacotes é gerenciado pelo protocolo IP Móvel e a reserva de recursos é efetuada pelo protocolo MRSVP. Como foi apresentado na Subseção 2.2.1.1, esse protocolo introduz três novas classes de serviços ao modelo *intserv*: *Mobility Independent Guaranteed*, *Mobility Independent Predictive* e *Mobility Dependent Predictive*. As classes de serviços são representadas através da hierarquia de derivações permitida pelo *framework para parametrização*. A classe *ServiceCategory* foi especializada na classe

IntservServiceCategory, a qual foi especializada nas classes *MobilityDependentServiceCategory* e *MobilityIndependentServiceCategory*, que indicam, respectivamente, se a manutenção do serviço depende ou não da localização do usuário. Essas classes, por sua vez, foram especializadas nas classes abstratas *GuaranteedService* e *PredictiveService*, representando os serviços garantido e preditivo (Clark et al., 1992). As categorias de serviços definidas no MRSVP são representadas através das classes concretas *MIGServiceCategory*, *MIPServiceCategory* e *MDPServiceCategory*, que instanciam a hierarquia de classes descrita anteriormente, como pode ser observado na Figura 3.2.

A classe *Parameter* corresponde ao topo da hierarquia de parametrização de serviços. As classes concretas *MSpec*, *TSpec*, *RSpec* e *AdSpec* são apresentadas na Figura 3.2 como instanciações da classe *Parameter*. Esses parâmetros serão adicionados às instâncias de *ServiceParameter* definindo os tipos de serviços oferecidos. A classe *MSpec* descreve a especificação de mobilidade do usuário, sendo um parâmetro interno ao sistema e, como tal, acessado somente pelos objetos instanciados a partir de *ServiceCategory*. Os parâmetros *TSpec* e *RSpec* estão associados respectivamente à descrição do tráfego e à caracterização do fluxo. O parâmetro *AdSpec*, que representa o desempenho típico de provedores de serviços *intserv*, também é representado por uma classe apropriada, com atributos definidos de acordo com a especificação do modelo.

O parâmetro *MSpec* é o responsável por determinar se a provisão de QoS para o fluxo do usuário depende ou não da sua localização atual, o que define a necessidade de se efetuarem reservas imediatas e/ou antecipadas (pré-alocação de recursos). Um objeto *MSpec* contém informações espaciais e temporais referentes à mobilidade do usuário. Com base na precisão dessas informações, o mecanismo de controle de admissão pode reservar recursos que irão garantir, com algum nível de certeza, a continuidade das conexões enquanto o usuário se desloca entre células distintas.

As classes de serviço introduzidas pelo MRSVP, como ilustrado na Figura 3.2, são representadas em um nível mais elevado como dependentes ou independentes da localização, o que é definido pelo parâmetro *MSpec*, podendo oferecer serviços garantidos ou preditivos. A classe de serviço *Mobility Independent Guaranteed* representa aplicações intolerantes e está associada aos parâmetros *TSpec* e *RSpec*. Já as classes *Mobility Independent Predictive* e *Mobility Dependent Predictive*, que representam aplicações

tolerantes, estão associadas somente ao parâmetro *TSpec*, pois apenas os valores de caracterização dos fluxos de tráfego do usuário são informados.

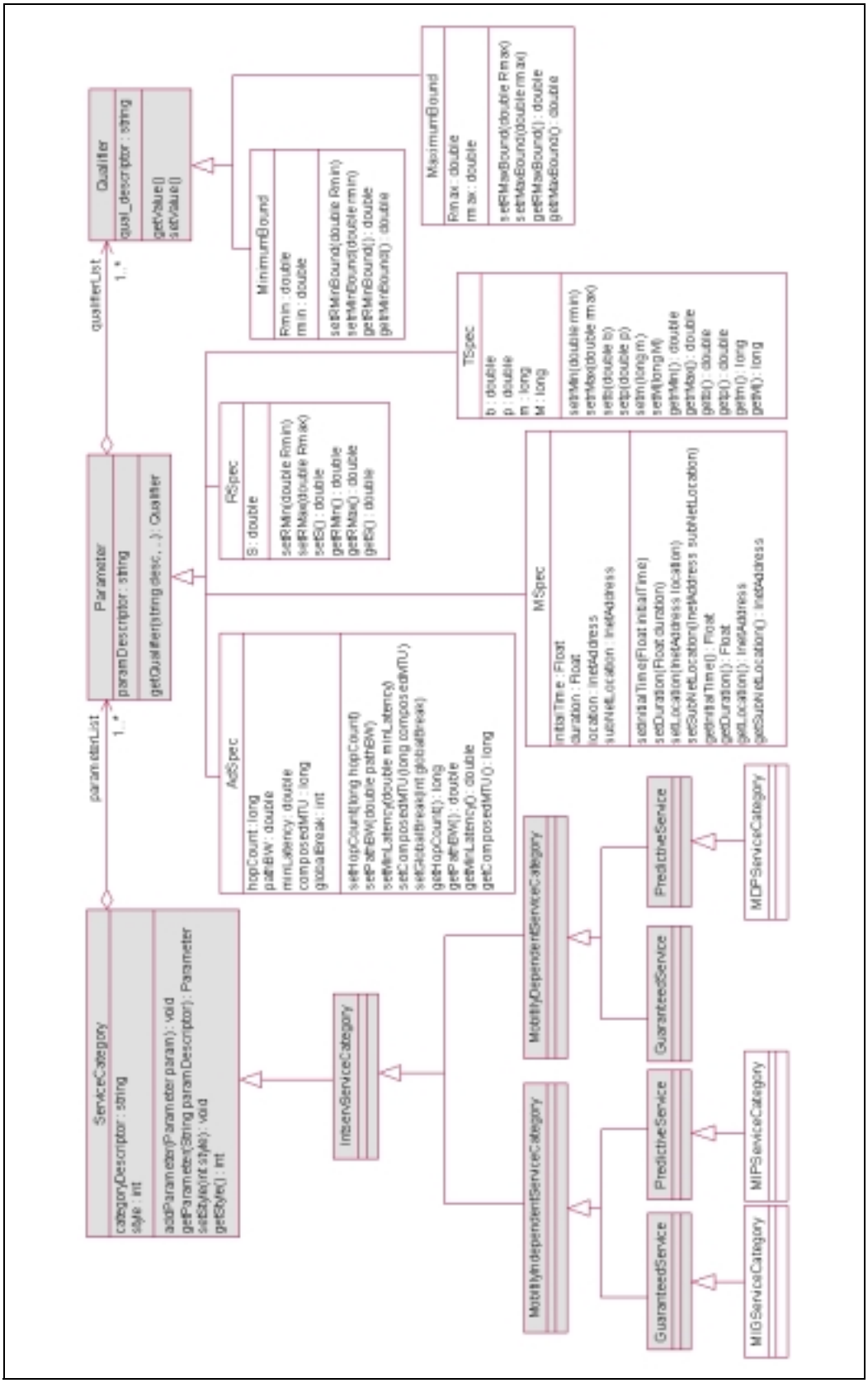

Figura 3.2 Exemplo de aplicação do *Framework* para Parametrização de Serviços

No *framework para parametrização de serviços*, a introdução do conceito de intervalos de QoS é feita através da classe abstrata *Qualifier*. As classes *MinimumBound* e *MaximumBound* representam respectivamente os limites inferior e superior apresentados na definição de intervalos de QoS. Na especialização modelada pelo exemplo apresentado na Figura 3.2, pode-se observar que somente os descritores das requisições do serviço (atributo `R`, que representa a taxa de dados, do objeto *RSpec*) e os descritores de tráfego (atributo `r`, que representa a taxa média de dados, do objeto *TSpec*) receberam esse nível de detalhamento[25].

3.2 Frameworks para Compartilhamento de Recursos

Como as estações móveis podem experimentar grandes variações de QoS devido à mobilidade, para se obter garantias de serviço independentes da mobilidade, uma estação móvel precisa efetuar reservas antecipadas de recursos nas diversas localidades que ela possa vir a visitar durante o tempo de vida de sua conexão. O que se propõe é determinar com antecedência a necessidade de se utilizar um determinado recurso e efetuar a sua reserva antecipadamente, enriquecendo as funcionalidades do esquema tradicional, que efetua somente reservas imediatas. Além de se garantir que os recursos sejam reservados independentemente dos possíveis deslocamentos do usuário, é interessante que se mantenha um controle temporal sobre essas reservas, definindo quando a reserva deve ser efetuada e por quanto tempo ela será mantida.

Para garantir tanto a alocação quanto a pré-alocação de recursos, será empregado o conceito de *recursos virtuais*[26], apoiando esta proposta no gerenciamento de recursos efetuado pela *árvore de recursos virtuais* descrita por Gomes (1999) e nas idéias de reserva de recursos antecipada introduzidas por Wolf & Steinmetz (1997).

[25] Isso se deve ao fato de que, no cenário implementado, somente os descritores de serviços associados à taxa de dados influenciarão na adaptação dos parâmetros que mapeiam a qualidade solicitada pelos usuários da aplicação. Entretanto, nada impede que, em outras instanciações, os demais atributos dos objetos *TSpec* e *RSpec* estejam associados a objetos instanciados a partir da classe *Qualifier*, os quais definiriam intervalos de tolerância.

[26] Recursos virtuais correspondem às parcelas de tempo, associadas a um ou mais fluxos de tráfego do usuário, nas quais foi dividida a utilização dos recursos de um provedor.

Ao se admitir uma nova chamada ou uma chamada originada por *handoff*, faz-se a sua associação a recursos virtuais controlados por um escalonador de recursos virtuais específico, o qual será definido pela categoria de serviço solicitada pelo usuário. Cada escalonador de recursos virtuais apresenta estratégias de admissão e de escalonamento relacionadas à categoria de serviço do recurso e às políticas de QoS adotadas.

A Figura 3.3 ilustra um exemplo de uma árvore de recursos virtuais em um ambiente móvel, utilizando o MRSVP, com o escalonamento de fluxos baseado nas classes garantida e preditiva, o qual será retomado na Subseção 3.2.2, onde é ilustrada uma instanciação do *framework*. O algoritmo de escalonamento considera somente os fluxos associados às reservas ativas no enlace[27]. A estratégia associada ao serviço garantido pode utilizar, por exemplo, a disciplina de escalonamento WFQ (*Weighted Fair Queuing*). A largura de banda que não é utilizada pelo serviço garantido é utilizada pelo serviço preditivo através de filas de prioridade, com os fluxos sendo escalonados, por exemplo, pelo algoritmo FIFO (*First In First Out*).

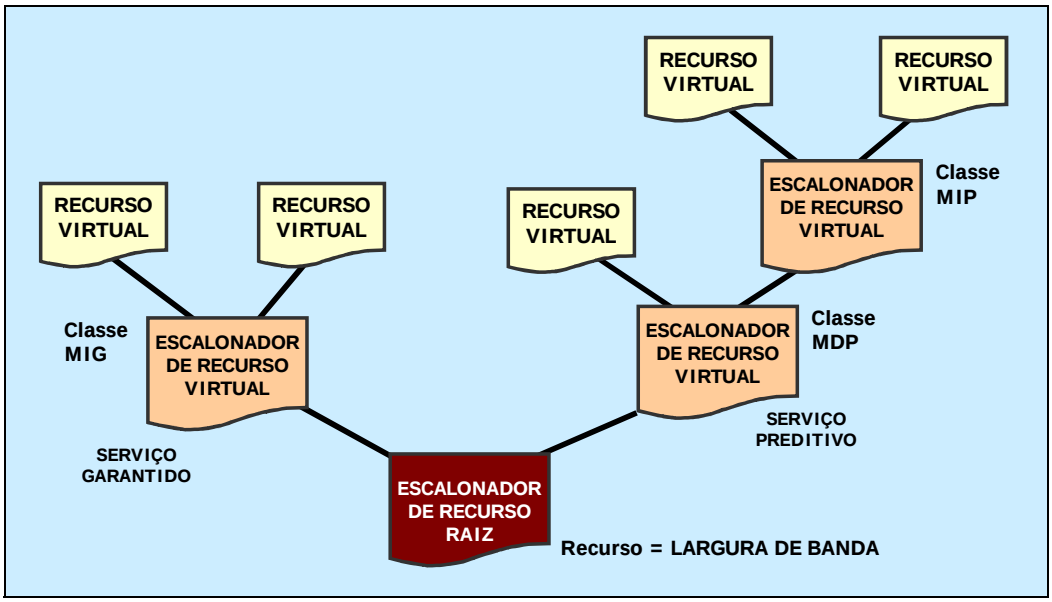

Figura 3.3 Exemplo de árvore de recursos virtuais em um roteador com suporte à mobilidade

Os mecanismos envolvidos com a criação de recursos virtuais são modelados no *framework para alocação de recursos*, enquanto as funções e estratégias envolvidas no

[27] Maiores detalhes sobre o algoritmo de escalonamento podem ser obtidos no trabalho de Talukdar et al. (Talukdar et al., 1998).

controle de tráfego (classificação, policiamento e escalonamento) estão representadas no *framework para escalonamento de recursos*. Esses *frameworks* atuam durante as fases de estabelecimento e manutenção de contratos de serviços, respectivamente, e, por modelarem funções bastante relacionadas (criação, configuração e escalonamento de recursos), são referenciados, em conjunto, como *frameworks para compartilhamento de recursos*.

3.2.1 Componentes dos Frameworks para Compartilhamento de Recursos

A Figura 3.4 traz uma visão conjunta dos componentes de escalonamento e alocação de recursos. Nenhuma classe foi acrescentada ou novas funcionalidades adicionadas em função da mobilidade dos usuários. O modelo definido por Gomes (1999) e adaptado por Mota et al. (2001) se adequa perfeitamente aos ambientes móveis sem fio, sendo adotado integralmente neste trabalho. A seguir serão apresentadas, de uma forma compacta, as classes e os respectivos relacionamentos que constituem o *framework para compartilhamento de recursos*.

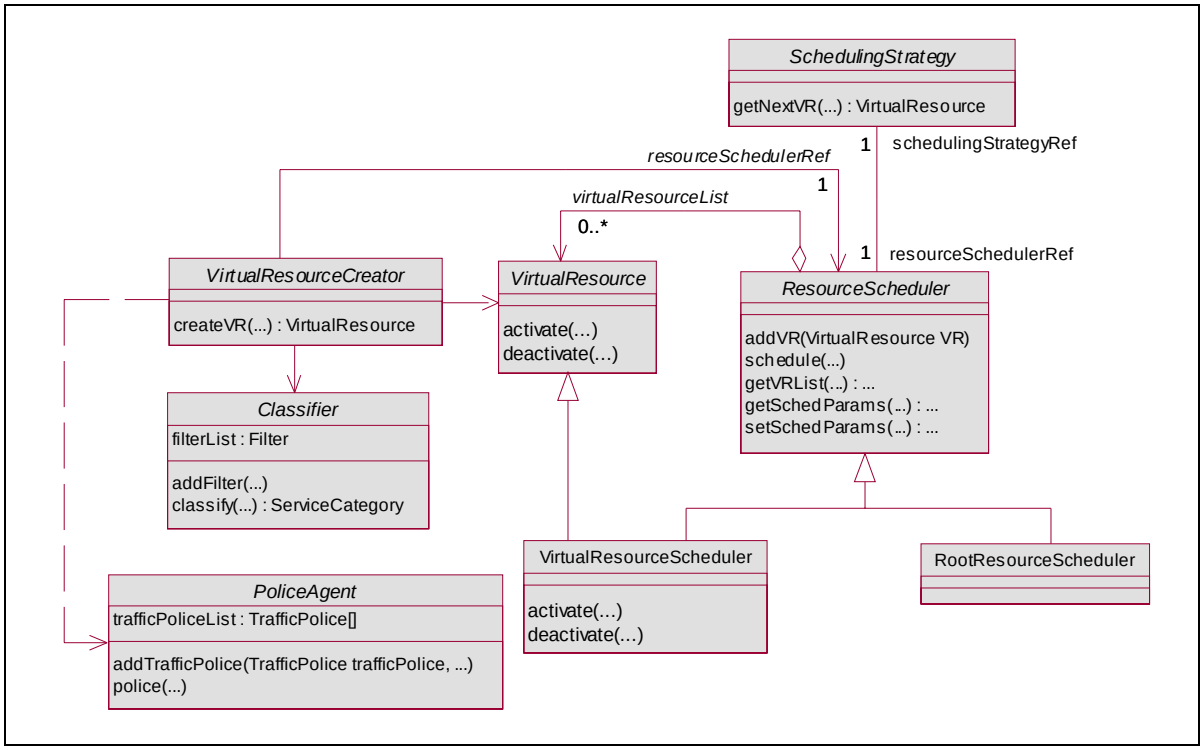

Figura 3.4 Visão conjunta dos *Frameworks* para Compartilhamento de Recursos

Na Figura 3.4, o mecanismo de escalonamento de recursos é modelado pelas classes abstratas *VirtualResource*, *VirtualResourceScheduler* e *RootResourceScheduler*. Essas

classes representam, respectivamente, os recursos virtuais, os escalonadores de recursos virtuais e os escalonadores de recurso raiz[28]. O *pattern* estrutural Adapter (Gamma et al., 1995) é utilizado, com algumas modificações, para definir os escalonadores de recursos através de uma única interface de escalonamento genérica, tornando-os uniformes, simbolizada pela classe abstrata *ResourceScheduler*.

No processo de escalonamento de recursos, a compatibilização de interfaces entre *RootResourceScheduler* e *VirtualResourceScheduler* é realizada através do método `schedule()` definido em *ResourceScheduler*. Este método permite que um escalonador desative o recurso virtual filho detentor do recurso real (`deactivate()`) e ative o próximo recurso virtual filho que irá utilizar esse recurso (`activate()`). Em algumas situações – quando, por exemplo, as árvores de recursos virtuais gerenciam múltiplos recursos reais –, os métodos citados anteriormente podem requerer um argumento que identifique a que recurso real o seu disparo está associado.

As classes abstratas *Classifier* e *PoliceAgent* foram introduzidas por Mota et al. (2001) e também fazem parte do *framework para escalonamento de recursos*. A classe *Classifier* diz respeito ao módulo de classificação, o qual é configurado pela adição de filtros, normalmente aplicados sobre campos específicos do cabeçalho dos pacotes. Os filtros são utilizados para retornar a classe de serviços associada ao pacote de dados, o que se faz através da identificação positiva de um padrão específico de cabeçalho. O método `addFilter()` é o responsável pelas associações realizadas entre os filtros e as categorias de serviços que são armazenadas no atributo `filterList`. Um filtro é aplicado aos pacotes de um fluxo de dados quando o método `classify()` é acionado nos agentes que atuam no mecanismo de classificação.

A classe *PoliceAgent* modela o mecanismo de policiamento e tem como atributo principal a lista `trafficProfileList`, que armazena as associações entre os perfis de tráfego do usuário e as ações de policiamento. O método `addTrafficProfile()` adiciona essas associações em `trafficProfileList`.

[28] O escalonador de recurso raiz é o responsável por escalonar diretamente as parcelas de utilização do recurso real, podendo gerenciar mais de um recurso real de um mesmo tipo simultaneamente.

Em *PoliceAgent*, a responsabilidade de verificar o tráfego do usuário, de modo a determinar se um pacote em particular encontra-se conforme ou não ao perfil de tráfego associado ao fluxo ao qual pertence, é delegada ao método `police()`. Caso o pacote não esteja de acordo com o perfil configurado, a ação de policiamento correspondente, contida na associação `trafficProfileList`, será executada.

Ainda na Figura 3.4 temos o *framework para alocação de recursos* sendo modelado através das classes *ResourceScheduler*, *VirtualResourceCreator* e *VirtualResource*, que representam de uma forma abstrata os componentes responsáveis pela criação de recursos virtuais. O *pattern* de criação Factory Method (Gamma et al., 1995) é utilizado para modelar o relacionamento de dependência entre esses componentes, definindo uma interface para a criação de recursos virtuais mas transferindo para as suas subclasses a definição do recurso que será instanciado. O relacionamento mencionado é simbolizado em *VirtualResourceCreator* através do método abstrato `createVR()`, cujas redefinições em cada uma das subclasses de *VirtualResourceCreator* implementam a instanciação de subclasses específicas de *VirtualResource*.

3.2.2 Exemplo de Aplicação dos Frameworks para Compartilhamento de Recursos

Dando continuidade ao exemplo apresentado na Subseção 3.1.2, a Figura 3.5 ilustrará o compartilhamento de recursos em uma rede de serviços integrados com suporte à mobilidade. O recurso considerado será a largura de banda, que é um recurso escasso em ambientes móveis sem fio.

O controle da utilização da largura de banda é efetuado por um objeto da classe *RootPacketScheduler*, que é uma instanciação da classe abstrata *RootResourceScheduler*. Para cada classe de serviços fornecida pelo provedor podem ser definidos escalonadores virtuais que serão representados no *framework para escalonamento* por objetos da classe *VirtualPacketScheduler*, especializada da classe *VirtualResourceScheduler*. Cada um desses escalonadores virtuais pode adotar diferentes estratégias de escalonamento.

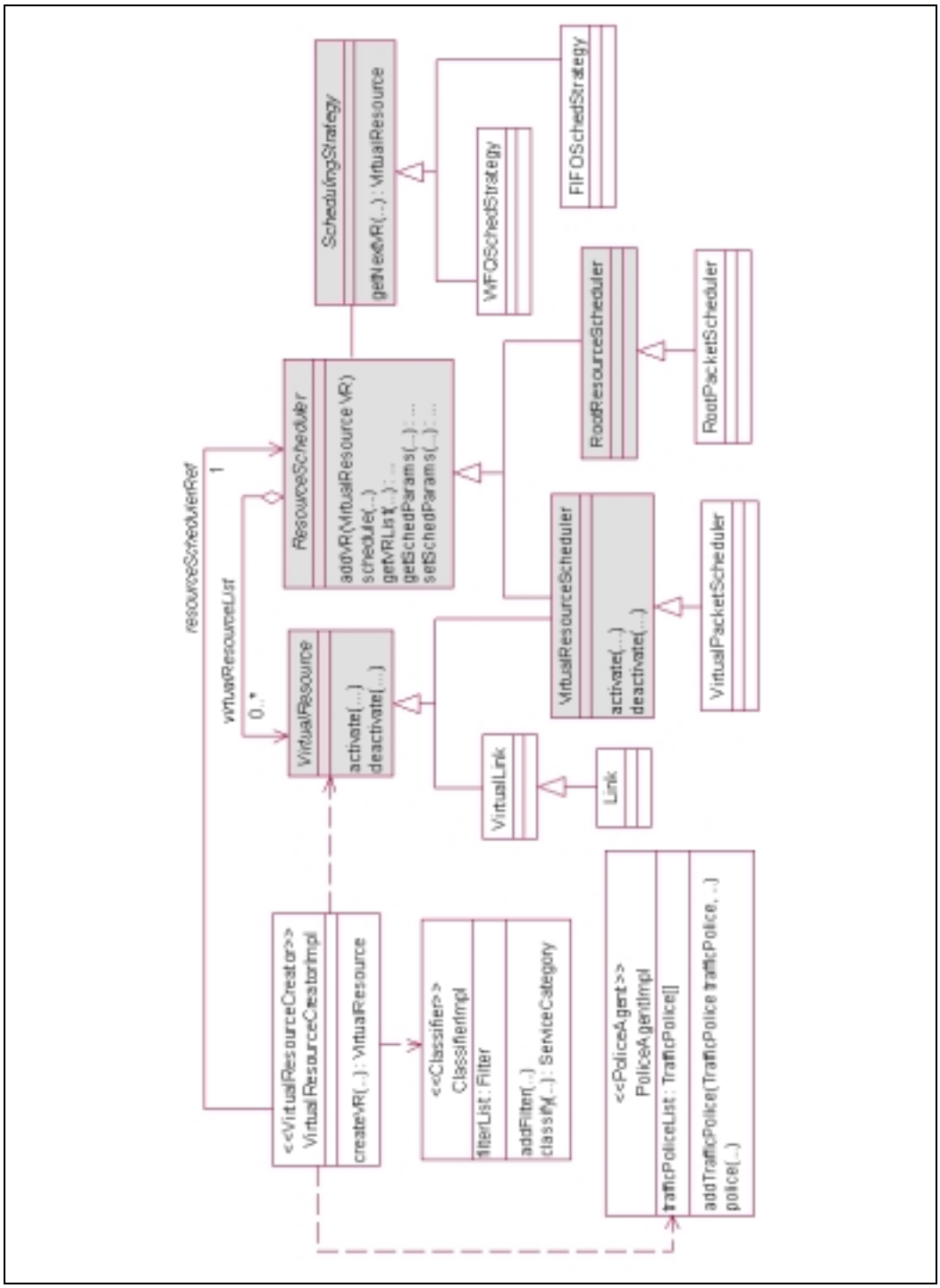

Figura 3.5 Exemplo de aplicação dos *Frameworks* para Compartilhamento de Recursos

Como se pode observar na Figura 3.5, as classes *RootResourceScheduler* e *VirtualResourceScheduler* são derivadas a partir de *ResourceScheduler*. Esta classe define indiretamente, através das suas derivações, o tipo de recurso que será criado. Como foi mencionado na Seção 3.2, as disciplinas de escalonamento utilizadas para os serviços garantido e preditivo são, respectivamente, os algoritmos WFQ (*Weighted Fair Queueing*) e FIFO (*First In First Out*), as quais estão representadas pelas classes

WFQSchedStrategy e *FIFOSchedStrategy*. O componente de criação de recursos virtuais é representado pela classe concreta *VirtualResourceCreatorImpl*. Na Figura 3.5 foi adotada uma notação especial que permite indicar, pela utilização dos símbolos "<<" e ">>", que a classe *VirtualResourceCreatorImpl* é uma implementação de *VirtualResourceCreator*. Uma notação similar foi utilizada para ilustrar as implementações *ClassfierImpl* e *PoliceAgentImpl*.

Com base na categoria de serviços solicitada e nos parâmetros de QoS informados pelo usuário e devidamente mapeados em valores que possam ser interpretados pelo componente de criação (*VirtualResourceCreatorImpl*), o recurso virtual será criado e, em seguida, associado a um escalonador responsável pela categoria negociada. No exemplo ilustrado pela Figura 3.5, os recursos virtuais são representados pela classe concreta *Link*, especializada da classe abstrata *VirtualLink*, que modela percentuais de acesso à largura de banda. A etapa seguinte é iniciada através do disparo do método `addFilter()`, definido na classe *ClassifierImpl*, que se encarregará de adicionar uma nova associação de filtro à categoria de serviços solicitada na reserva. Por fim, o perfil de tráfego definido pelo usuário, bem como a ação de policiamento correspondente a esse perfil, serão informados ao mecanismo de policiamento (*PoliceAgentImpl*), sendo passados como parâmetro através do método `addTrafficDesc()`, concluindo, dessa forma, a fase de estabelecimento de contrato de serviços.

3.3 Frameworks para Orquestração de Recursos

Na orquestração de recursos, os *agentes de negociação* são os elementos responsáveis pela coordenação do processo de provisão de QoS, que pode ser efetuado de uma forma distribuída ou centralizada. Na abordagem distribuída, os agentes de negociação presentes em cada um dos subsistemas atuam em conjunto de modo a definir a melhor forma de garantir a provisão da QoS solicitada. Já na abordagem centralizada, o agente de negociação recebe a solicitação e a encaminha para um agente central, que se encarrega de tomar as decisões relativas ao fornecimento do serviço.

Quando o *controlador de admissão* recebe uma nova requisição, ele se comunica com o agente de negociação de modo a definir quais os subsistemas que serão

acionados para garantir o fornecimento do serviço, verificando a disponibilidade de recursos nos mesmos. Ao se admitir uma nova solicitação, deve ser feito um balanceamento entre os fluxos admitidos previamente e os fluxos solicitantes, de modo a se manter a QoS requisitada por aqueles, levando-se em conta o estado interno atual do sistema, a caracterização da carga e a especificação da QoS. A admissão dos fluxos do usuário resultará na *alocação dos recursos* necessários para a sua manutenção, o que ocasionará mudanças no estado interno do sistema[29]. Caso o sistema não possua recursos disponíveis para atender à solicitação do usuário, o pedido será rejeitado.

Após a admissão dos fluxos do usuário pelo sistema, este deve garantir que o usuário atenha-se à carga caracterizada e que a QoS negociada seja mantida durante todo o tempo de uso do serviço. É essa garantia que define um contrato de serviços e, como em todo contrato, ambas as partes interessadas devem colaborar, ou seja, o usuário deve ater-se à carga caracterizada e o sistema 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. Quando admitidos, os fluxos passam a ser *monitorados* continuamente, o que poderá acionar tanto os mecanismos de *renegociação* quanto os de *sintonização da QoS*[30].

Em uma rede móvel sem fio, ao se estabelecer um SLA (*Service Level Agreement*) entre o provedor e o usuário do serviço, deve-se efetuar a reserva não somente na localidade e no instante atuais (reserva imediata). Nesses ambientes, os recursos devem ser também reservados em todas as células para as quais o nó móvel poderá se deslocar, estabelecendo-se reservas ao longo de todo o novo caminho, levando-se em consideração o aspecto temporal, ou seja, quanto tempo irá durar a sessão do usuário e o momento no qual a reserva deverá ser iniciada em cada localidade visitada. Pode-se observar que a provisão de garantias determinísticas de QoS fim-a-fim em ambientes móveis sem fio durante o tempo de vida de uma conexão não transcorre sem que haja um possível desperdício de recursos. Além das flutuações na

[29] O ciclo de solicitações e respostas entre os usuários e o sistema é de responsabilidade do mecanismo de *negociação da QoS* (Subseção 3.3.1).

[30] Mecanismo responsável 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.

disponibilidade de recursos, nem sempre é possível determinar com antecedência os deslocamentos do usuário móvel e muito menos quando esses deslocamentos serão efetuados. O objetivo das pesquisas nesta área tem sido encontrar um equilíbrio entre níveis aceitáveis de QoS e reservas antecipadas, o que reduziria a interrupção de conexões em andamento e a não admissão de novas conexões, garantindo uma melhor utilização de recursos na rede e no enlace sem fio.

O mecanismo de orquestração tem a responsabilidade de manter as garantias de QoS das conexões previamente admitidas, assegurando a otimização da utilização dos recursos da rede. Como foi mencionado na Seção 3.1, uma política de adaptação nos parâmetros da aplicação auxilia no escalonamento dos fluxos durante o *handoff* ou em períodos de desvanecimento, com base nas informações do sistema. Caso os mecanismos de adaptação não possam ser empregados, ou sejam empregados sem sucesso, então o mecanismo de renegociação é acionado.

Em ambientes móveis, o controle de admissão deve ser efetuado considerando a prioridade dada às conexões originadas por *handoff*. O objetivo do algoritmo de controle de admissão deve ser admitir tantos usuários móveis quanto possível, mantendo a QoS contratada. Como exemplo, no trabalho de Talukdar et al. (1998) é apresentado um mecanismo de controle de admissão para redes de serviços integrados considerando as classes de serviços introduzidas pelo MRSVP.

Um ponto importante a ser tratado em ambientes móveis são *as políticas de controle* que verificam as permissões do usuário para efetuar reservas em cada localidade. Essas permissões podem sofrer alterações de acordo com a localização atual do usuário. No trabalho de Chalmers & Sloman (1999) as requisições de QoS são descritas como *políticas de autorização* responsáveis pela definição dos serviços ou recursos que o usuário pode acessar, tendo como base as restrições referentes à sua localização atual.

Em ambientes móveis deve existir uma maior flexibilidade nos mecanismos de manutenção de contratos, sendo aceitável uma degradação suave do serviço. É necessário que se saiba lidar com mudanças abruptas na qualidade do serviço oferecido de uma localidade para outra e até mesmo com curtos períodos de desconexão (desvanecimento). Essa exigência determina que os mecanismos que acionam a renegociação e a sintonização da QoS esperem um intervalo de tempo até

disparar essas operações, a fim de evitar que flutuações passageiras interfiram no contrato estabelecido (Chalmers, 1998). Vale ressaltar que a disponibilidade do enlace sem fio sofre flutuações não somente devido à mobilidade do usuário, mas também às condições do canal (Choi, 1999). Quando a capacidade do sistema de prover recursos é incrementada, seja devido à liberação de recursos ou ao restabelecimento das condições do canal, o mecanismo de sintonização deve ser acionado em sentido inverso, desta vez com o objetivo de melhorar as condições oferecidas aos usuários.

O *framework para orquestração de recursos* é o responsável pela admissão/rejeição de conexões através do mecanismo de controle de admissão, que é alimentado pelo escalonador de recursos com informações referentes à utilização atual do sistema e ao tipo de solicitação efetuada. Os mecanismos responsáveis pelo estabelecimento e manutenção de contratos de serviços são representados conceitualmente através dos *frameworks para negociação* e *sintonização da QoS*, respectivamente, sendo referenciados em conjunto como *frameworks para orquestração de recursos*.

3.3.1 Componentes do Framework para Negociação da QoS

A Figura 3.6 ilustra o *framework para negociação da QoS*, trazendo as classes que o constituem e os seus respectivos relacionamentos. A solicitação da QoS, efetuada por um usuário do subsistema, é iniciada através do método `admit()` da classe abstrata *AdmissionController*. O *pattern* estrutural Facade (Gamma et al., 1995) é utilizado para representar o processo de requisição de serviços oferecido pelos métodos `admit()` e `commit()` de *AdmissionController*, ocultando toda a complexidade envolvida no mecanismo de estabelecimento de contratos de serviços. O método `admit()` 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.

O método `request()` da classe abstrata *QoSNegotiator*, ao ser acionado, alimenta o agente de negociação com os parâmetros associados à provisão do serviço, como os descritores dos fluxos, a categoria de serviços desejada e o identificador do usuário, entre outros. No caso específico dos ambientes móveis, pode-se acrescentar a especificação de mobilidade do usuário, cujo conteúdo irá definir as próximas localidades que serão visitadas pelo nó móvel, podendo se definir, ainda, em que instante esses deslocamentos ocorrerão e quanto tempo

durarão. Será com base nesses parâmetros que o negociador poderá verificar se as restrições impostas pelas políticas configuradas no provedor irão permitir ou não o fornecimento do serviço.

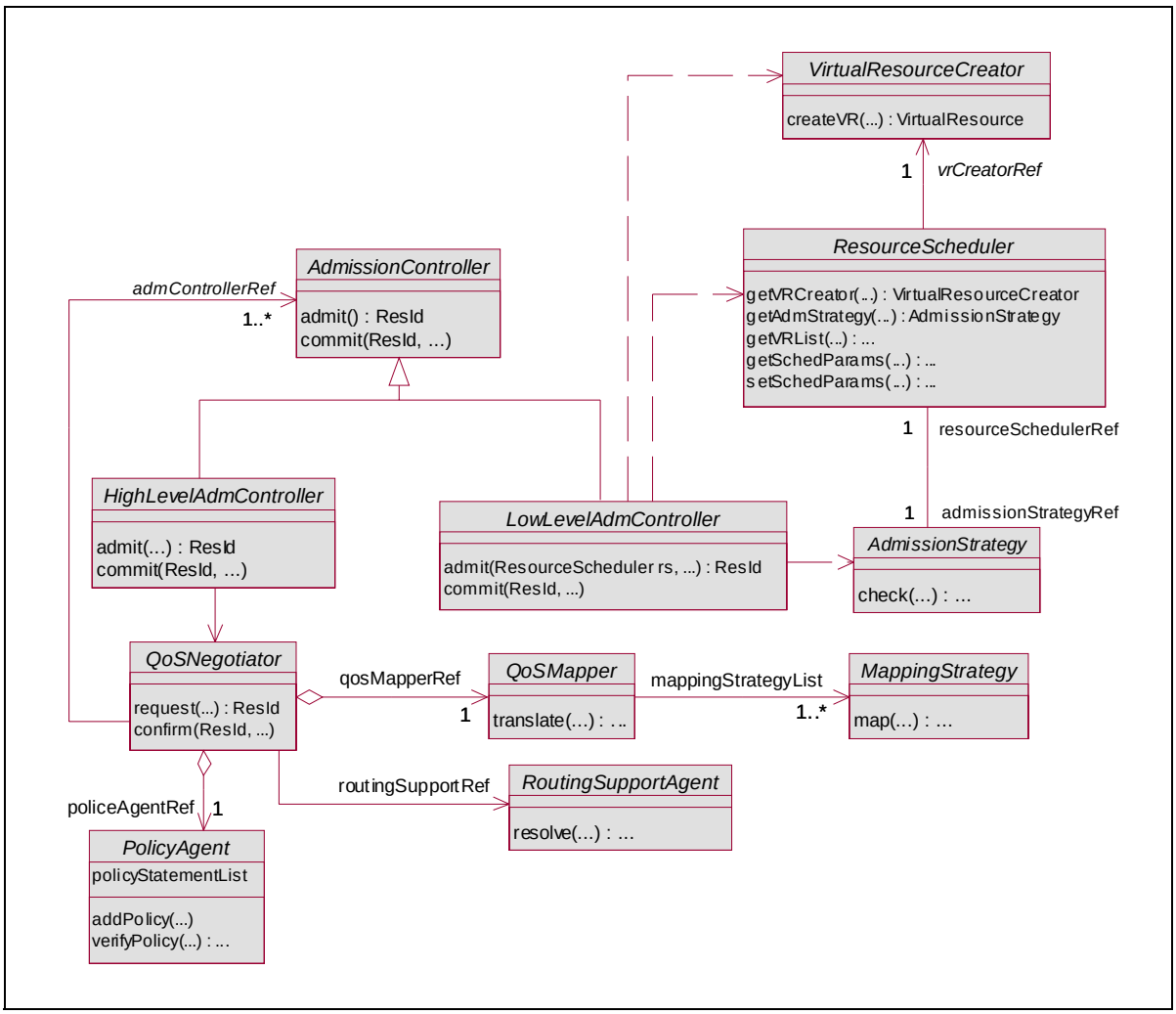

Figura 3.6 *Framework* para Negociação da QoS

Os parâmetros informados ao negociador da QoS serão confrontados com as restrições estabelecidas pelas políticas de controle que foram configuradas no provedor do serviço, o que irá determinar o fornecimento ou não do serviço ao usuário. Esse controle é realizado através do método `verifyPolicy()`, implementado em objetos da classe abstrata *PolicyAgent*. Quando se oferece suporte à mobilidade, há uma preocupação maior com as restrições de acesso que podem ser encontradas quando as estações se deslocam entre diferentes domínios administrativos. Esses deslocamentos podem implicar em alterações na forma como um serviço é oferecido por questões relacionadas, por exemplo, à segurança ou mesmo à tarifação. Como se pode observar, novas políticas de controle podem ser

adicionadas em uma lista denominada `policyStatementList` através do método `addPolicy()`, o que caracteriza mais um ponto de flexibilização do *framework*.

Considerando a solicitação do usuário e as verificações referentes às políticas de controle implementadas no provedor, o agente de negociação terá a responsabilidade de definir os subsistemas mais apropriados à realização do serviço solicitado. Geralmente, a escolha desses subsistemas é realizada com o auxílio de mecanismos de roteamento com QoS (Mota et al., 2001), representados no *framework para negociação da QoS* através da classe *RoutingSupportAgent*. O método `resolve()` irá retornar ao agente de negociação uma lista com os provedores adjacentes que atendem às restrições definidas pela combinação dos parâmetros solicitados pelo usuário e das políticas de controle do provedor. A relação `admControllerRef` entre as classes *QoSNegotiator* e *AdmissionController* representa a possibilidade de se implementar o mecanismo de negociação de forma distribuída ou centralizada.

O próximo passo no processo de negociação é estipular com que parcela cada subsistema irá contribuir na provisão do serviço solicitado como um todo. Para tanto, faz-se necessária a tradução dos parâmetros informados pelo usuário aos agentes de negociação, os quais atuam em um nível mais alto da abstração, em valores compreensíveis ao nível de visão da QoS correspondente ao subsistema que será acionado. Esse papel é desempenhado pelos mapeadores de QoS, representados na Figura 3.6 pela classe abstrata *QoSMapper*. As requisições de serviços, representadas pelos parâmetros de serviço, são traduzidas em estratégias de mapeamento através do método `translate()` (*QoSMapper*), o qual retorna uma lista de categorias de serviços compatíveis. Uma variação do *pattern* comportamental Strategy (Gamma et al., 1995) é utilizada para representar a relação de associação (`mappingStrategyList`) entre as classes abstratas *QoSMapper* e *MappingStrategy*. As estratégias de mapeamento são representadas abstratamente pela classe *MappingStrategy*. A especialização dessa classe é obtida através da implementação do método abstrato `map()`.

Caso o provedor esteja em um nível mais externo na hierarquia de provedores (nível mais alto da abstração), o seu *AdmissionController* será instanciado a partir da classe abstrata *HighLevelAdmController*, a qual acionará o agente de negociação através do método `request()` (*QoSNegotiator*). O mecanismo de negociação irá, por sua vez, identificar todos os provedores intermediários que podem se envolver no

fornecimento do serviço requisitado, acionando os seus controladores de admissão através do método `admit()` (*AdmissionController*). Se o provedor estiver manipulando diretamente o recurso, então o *AdmissionController* será instanciado a partir da classe abstrata *LowLevelAdmController*, a qual se encarregará de verificar, através do método `admit()`, a disponibilidade de criação do novo recurso diretamente na árvore de recursos virtuais, utilizando o método `check()` da classe *AdmissionStrategy*. Essa classe representa, através de uma abstração, as estratégias de admissão associadas a cada categoria de serviço, e sua instanciação corresponde à implementação do método `check()`. Novamente, é apresentada uma variação do *pattern* Strategy representando a relação de dependência (`admissionStrategyRef`) entre o gerente do recurso e as possíveis estratégias de admissão a serem adotadas. A escolha das estratégias de admissão representa um ponto de flexibilização do *framework*, podendo ser preenchido, no caso das redes móveis, por algoritmos que levem em consideração os limites máximos e mínimos definidos através dos intervalos de QoS.

As classes *AdmissionStrategy* e *ResourceScheduler*, que representam, respectivamente, as estratégias empregadas no controle de admissão e os escalonadores de recursos, estão relacionadas no *framework para negociação da QoS* pelas associações `admissionStrategyRef` e `resourceSchedulerRef`. As implementações dessas classes com relação a uma categoria de serviço específica estão intimamente relacionadas, já que a estratégia de admissão deverá atuar diretamente no escalonador correspondente à categoria dos fluxos que ela está tentando admitir. Essa informação é fornecida para a estratégia de admissão através de um dos argumentos passados na chamada ao método `admit()`.

Para que os recursos virtuais previamente alocados através do método `admit()` sejam de fato criados, é necessário que o usuário invoque o método `commit()`. Como pode ser observado, esses dois métodos de *AdmissionController* são complementares. O método `commit()` acionará o método `createVR()` (*VirtualResourceCreator*), o qual será o responsável, em última instância, pela criação do recurso na árvore de recursos virtuais.

3.3.2 Exemplo de Aplicação do Framework para Negociação da QoS

A Figura 3.7 dá continuidade ao exemplo apresentado na Subseção 3.1.2 e ilustra uma instanciação do *framework para negociação da QoS* em uma rede de serviços integrados com suporte à mobilidade, adotando o modelo de negociação distribuído. A instanciação é feita através da implementação de negociadores, mapeadores da QoS e mecanismos de policiamento.

A implementação dos negociadores é feita através de instanciações da classe *IntservQoSNegotiator*. A troca de informações adicionais que são representadas por estruturas do protocolo de sinalização, o MRSVP (no exemplo da Figura 3.7), é implementada na classe *MRSVPIntservQoSNegotiator*.

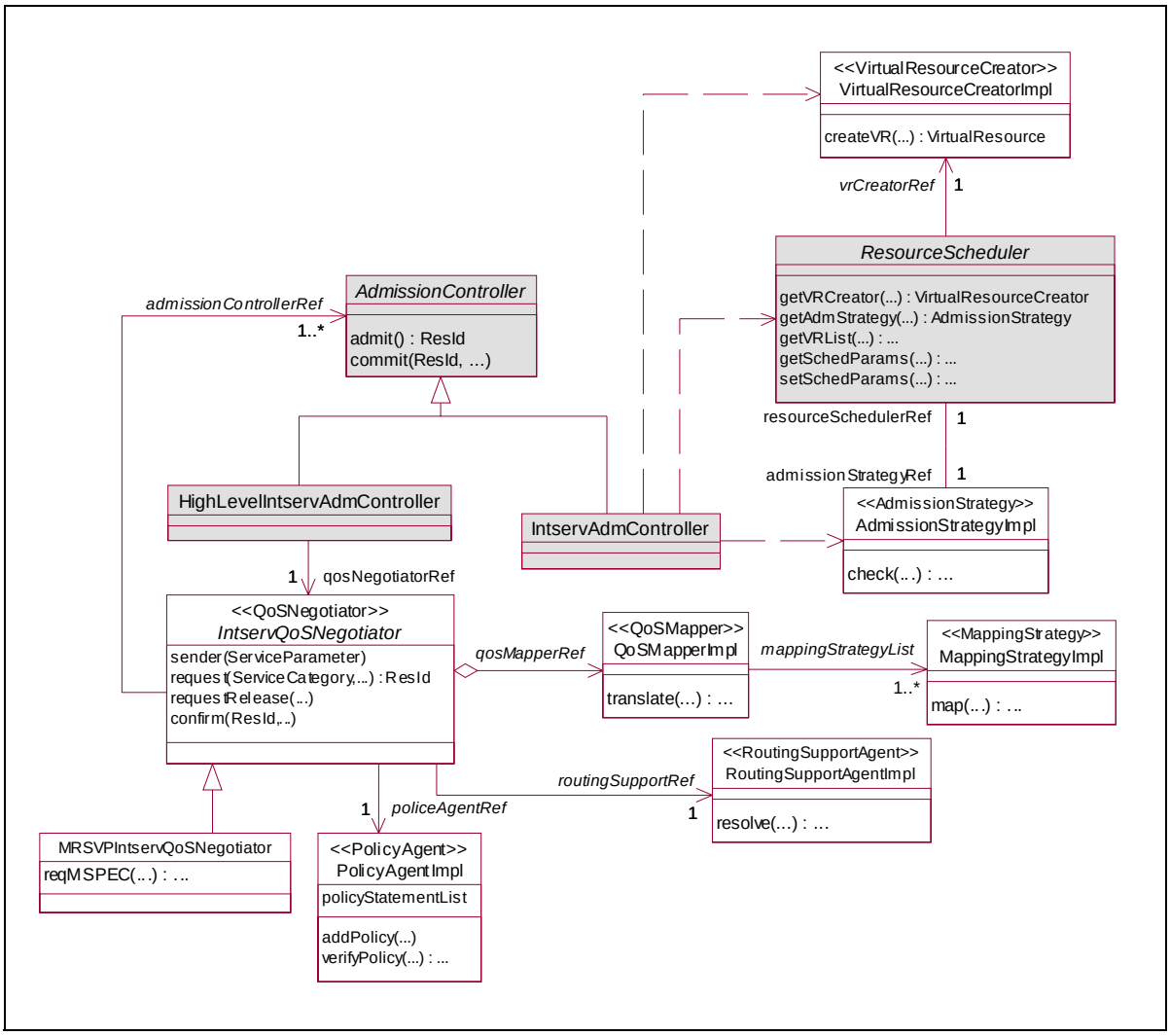

Figura 3.7 Exemplo de aplicação do *Framework* para Negociação da QoS

O mapeador é implementado através da classe *QoSMapperImpl*, que se encarregará de traduzir os parâmetros passados na requisição do serviço, os quais determinam as classes de serviços definidas pelo MRSVP (MIG, MIP e MDP), em valores que possam ser compreendidos pelos escalonadores de recursos locais.

O mecanismo de policiamento é implementado na classe *PolicyAgentImpl* e irá informar aos negociadores da QoS se o usuário que solicitou o serviço tem permissão de acesso aos recursos locais. Durante os seus deslocamentos, os usuários podem alcançar regiões que não correspondam à sua área de cobertura ou mesmo regiões que ofereçam o serviço com algumas restrições e mesmo tarifações diferenciadas.

A definição de intervalos de QoS e a possibilidade de efetuar reservas de recursos de forma antecipada devem ser consideradas na fase de negociação da QoS. A implementação da estratégia de admissão irá refletir esses novos conceitos. Os parâmetros de QoS e as categorias de serviços definidos na fase de parametrização (Seção 3.1) podem trazer informações a respeito da mobilidade do usuário, como os seus próximos deslocamentos, o momento em que esses deslocamentos ocorrerão e a sua duração, além de informações sobre os intervalos de tolerância em relação aos recursos solicitados. O tratamento dessas informações representará um processamento adicional para o módulo de controle de admissão, que inicialmente só deveria verificar a disponibilidade de recursos para os novos fluxos considerando a sua classe de serviço. Ao constatar a impossibilidade de se admitir um novo fluxo com a QoS máxima, o controlador de admissão verificará se possui recursos suficientes para atender a esse fluxo com uma QoS inferior, dentro do intervalo informado na fase de parametrização. Caso isso não seja possível, o controlador de admissão deverá recalcular os valores para os quais os fluxos anteriores foram admitidos utilizando os intervalos de QoS de cada fluxo, de modo a liberar recursos suficientes para admitir o novo fluxo.

No exemplo ilustrado na Figura 3.7, a pré-alocação de recursos será efetuada pela implementação do MRSVP, o qual se encarregará de utilizar as informações passadas através do MSPEC do nó móvel e acionar os agentes de negociação em

cada localidade a ser visitada[31] através do método `request()` modificado. Em ambientes que oferecem suporte à mobilidade, a reserva só será efetuada de fato se todos os agentes de negociação em cada localidade na especificação de mobilidade do nó móvel retornarem uma resposta positiva.

3.3.3 Componentes do Framework para Sintonização da QoS

A Figura 3.8 representa o *framework para sintonização da QoS*, trazendo as classes que o constituem e os seus respectivos relacionamentos. Como pode ser observado, a sua estrutura é bem similar àquela apresentada no *framework para negociação da QoS* (Figura 3.6). Assim como o mecanismo de negociação, a sintonização pode ser efetuada de forma distribuída ou centralizada, o que é simbolizado pela relação `adjustControllerRef` definida em *QoSTuner*.

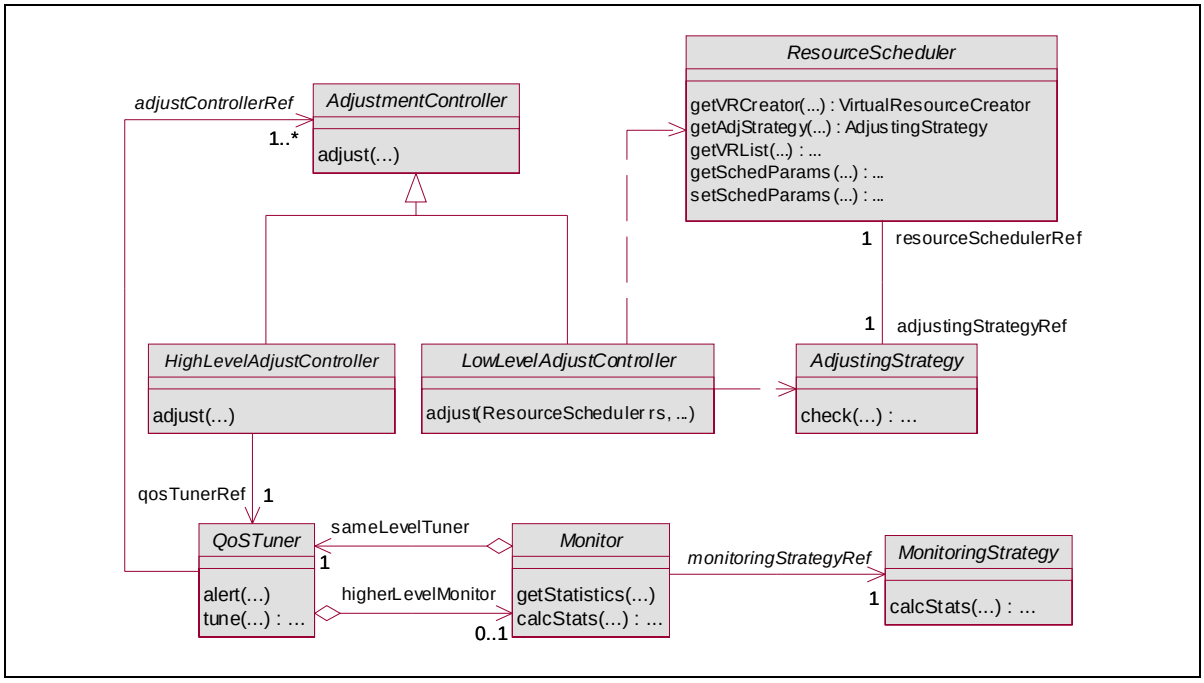

Figura 3.8 *Framework* para Sintonização da QoS

A sintonização da QoS é acionada quando são recebidos alertas, disparados pelo mecanismo de monitoramento quando este detecta condições inadequadas para a manutenção dos fluxos admitidos. As classes abstratas *Monitor* e *MonitorStrategy*

[31] Maiores detalhes sobre o funcionamento do MRSVP podem ser encontrados no trabalho de Talukdar et al. (2001).

representam, respectivamente, os monitores de fluxos e as suas estratégias de monitoramento. Essas classes estão associadas através do relacionamento `monitoringStrategyRef`. O método `getStatistics()` presente em *Monitor* é o responsável por obter as medições periódicas referentes à QoS que está sendo fornecida aos usuários do sistema e por enviar alertas ao seu agente de sintonização, disparando o método `tune()` no objeto instanciado a partir de *QoSTuner*. O cálculo dessas medições é efetuado em *MonitorStrategy* pelo método `calcStats()`. Os alertas são enviados antes que os contratos estabelecidos previamente sofram algum tipo de degradação, permitindo o acionamento de medidas preventivas, garantindo, dessa forma, a manutenção do nível da QoS contratada. A interação entre os mecanismos de sintonização e de monitoramento é simbolizada no *framework* para sintonização através dos relacionamentos `sameLevelTuner` e `higherLevelMonitor` entre as classes *QoSTuner* e *Monitor*. Esses relacionamentos indicam a possibilidade do *sintonizador* se comunicar com monitores de níveis distintos, podendo acionar o método `getStatistics()` de um monitor de nível N+1 caso o acionamento do método `tune()` no nível N não tenha obtido sucesso.

O método `tune()`, que é disparado pela chegada de alertas dos monitores de fluxo, irá acionar o método `adjust()` em instâncias de *AdjustmentController*. O que se vê aqui é semelhante ao que ocorre no *framework para negociação*: caso se esteja lidando diretamente com o gerente do recurso virtual, são criadas instâncias de *LowLevelAdjustController*; caso se esteja lidando com um agente de sintonização de um nível superior da hierarquia, instâncias de *HighLevelAdjustController* serão acionadas até que não existam mais provedores intermediários. O método `adjust()` é o responsável pelo disparo do método `setSchedParams()` em *VirtualResource*, que irá redimensionar a parcela de utilização confiada a um recurso virtual. Antes que qualquer alteração seja efetuada, será aplicado um teste para verificar a sua viabilidade. Esse teste é semelhante às estratégias de admissão que foram implementadas em *AdmissionStrategy*, só que nessa etapa o recurso virtual já existe, tendo sido criado na fase de negociação. De modo similar a escolha das estratégias de admissão, a escolha do teste efetuado pelo mecanismo de sintonização também representa um ponto de flexibilização do *framework*, podendo redimensionar as parcelas de utilização do recurso levando em consideração os limites máximos e mínimos definidos através dos intervalos de QoS. Vale ressaltar que esse

redimensionamento será aplicado a todos os provedores intermediários responsáveis pela sintonização da QoS dos fluxos do usuário.

Considerando o conceito de intervalos de QoS, quando o teste de viabilidade efetuado por `adjust()` indicar a impossibilidade de se efetuarem ajustes no sistema, pode-se reduzir os parâmetros que mapeiam a QoS contratada junto aos usuários para valores intermediários entre o valor máximo e o valor mínimo tolerável pela aplicação. Isso evita que o usuário tenha o seu contrato rompido e precise negociar novos valores ou definir um novo caminho, delegando aos agentes de sintonização autonomia para fazer alterações em nome da aplicação do usuário, desde que os valores sejam redimensionados dentro do intervalo informado na solicitação do serviço. A adoção dessa abordagem é possível já que as estratégias de ajuste correspondem a um ponto de flexibilização, no *framework para sintonização*, a ser instanciado a partir da classe *AdjustingStrategy*.

3.3.4 Exemplo de Aplicação do Framework para Sintonização da QoS

Admitindo que a fase de negociação tenha transcorrido com sucesso, tem-se então a fase de manutenção da QoS fim-a-fim, que é feita através do monitoramento dos fluxos. Essa manutenção é de responsabilidade do *framework para sintonização da QoS* e a sua implementação está ilustrada na Figura 3.9, que dá continuidade ao exemplo apresentado na Subseção 3.3.2.

O estado interno do sistema é monitorado por um objeto de *MonitorImpl* que é uma instanciação da classe abstrata *Monitor*. Durante o monitoramento, quando são detectados fluxos sendo atendidos com níveis de QoS inferiores aos que haviam sido contratados na fase de negociação, são gerados alertas e o agente de sintonização é acionado através do método `tune()`. O agente de sintonização é instanciado a partir da classe *HighLevelIntservAdjustController* e sua função é reconfigurar o mecanismo de escalonamento, definindo novos valores dentro dos intervalos de QoS informados para cada fluxo[32], de modo a manter a QoS negociada. Caso o processo de sintonização não obtenha resultados positivos, o agente de sintonização irá

[32] Esse processo é semelhante ao que foi descrito na Subseção 3.3.2.

encaminhar o alerta recebido, a partir do mecanismo de monitoramento, para o agente de negociação, o qual definirá uma rota alternativa por onde serão encaminhadas as mensagens de caminho ativas e passivas (*Path*) do MRSVP. Como conseqüência, uma nova negociação será iniciada entre os nós intermediários que fazem parte do novo caminho estabelecido.

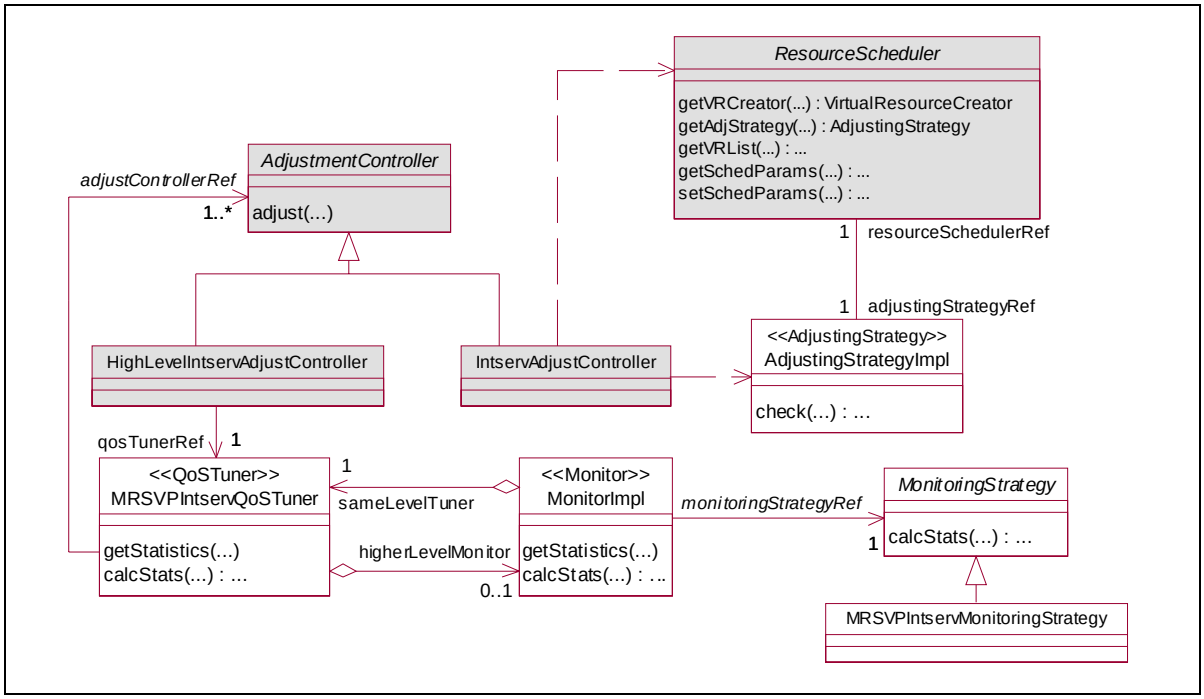

Figura 3.9 Exemplo de aplicação do *Framework* para Sintonização da QoS

3.4 Framework para Gerenciamento de Mobilidade

Diversos estudos vêm sendo desenvolvidos com o objetivo de oferecer mecanismos de predição dos deslocamentos do usuário para auxiliar no processo de gerenciamento da mobilidade. Esses mecanismos se propõem a disponibilizar, com uma certa antecedência, estimativas relacionadas aos possíveis deslocamento das estações móveis e, em alguns casos, informações de sincronismo, como o tempo de permanência da estação em cada célula e o tempo total da conexão. Essas informações podem ser representadas através de grafos cujos vértices correspondem aos prováveis provedores para os quais o nó móvel pode migrar e as arestas podem ser valoradas indicando o instante em que o deslocamento irá ocorrer ou a sua probabilidade de ocorrência.

Para que se possa determinar antecipadamente os futuros deslocamentos do usuário, de modo a se construir a sua especificação de mobilidade, além de se adotar

uma heurística de predição é necessário manter informações a respeito do histórico de deslocamentos. Na Subseção 2.1.4 foram ilustradas algumas das técnicas empregadas na predição da mobilidade do usuário e o seu estreito relacionamento com a manutenção de históricos de mobilidade. Uma percepção comum aos trabalhos desenvolvidos nessa área é que os movimentos dos usuários apresentam um certo nível de regularidade, e é essa regularidade, registrada nos históricos de mobilidade, que torna possível a utilização de mecanismos de predição.

Na verdade, os arquivos de especificação de mobilidade mencionados por Talukdar et al. (2001) e Pajares et al. (1999) podem ser vistos como um caso particular de uma base de dados administrada pelo gerenciador de mobilidade, que seria o responsável pela manutenção dessas informações. Nesse caso, as informações relacionadas à localização do nó móvel são atualizadas localmente pelo próprio nó, o que caracteriza uma abordagem distribuída. Uma outra possibilidade seria a implementação de um gerenciador centralizado que controlaria informações referentes aos registros de localização de todos os nós (Pitoura & Samaras, 1998).

O *framework para gerenciamento de mobilidade* incorpora o mecanismo de gerenciamento de localização, que se traduz na manutenção do histórico de mobilidade do usuário, e o de predição da mobilidade, que emprega heurísticas para definir antecipadamente a trajetória do nó com base no seu histórico.

3.4.1 Modelos de Gerenciamento de Mobilidade

Pelo que foi abordado na Subseção 2.1.4, pode-se destacar os principais caminhos que vêm sendo seguidos no sentido de oferecer serviços baseados no gerenciamento da mobilidade do usuário. Na provisão de QoS em redes móveis sem fio, faz-se necessário identificar o itinerário seguido pela unidade móvel para que os recursos requisitados possam ser reservados antecipadamente em todas as localidades que serão visitadas. *O framework para gerenciamento de mobilidade* oferece essas informações aos *frameworks para orquestração* de modo que os recursos sejam contratados somente nas células para as quais o nó móvel possa se deslocar, evitando que os mesmos sejam reservados desnecessariamente, o que provocaria uma utilização ineficiente do enlace sem fio e geraria um alto tráfego de mensagens originadas através do protocolo de reserva de recursos.

O gerenciamento de mobilidade pode ser implementado adotando duas abordagens distintas, mas que poderão interagir para atender de forma mais eficiente às demandas por informações referentes à mobilidade do usuário: a (i) *abordagem centralizada* e a (ii) *abordagem distribuída*.

- *Abordagem Centralizada*

  A abordagem centralizada (Figura 3.10) define dois papéis bem claros no mecanismo de gerenciamento de mobilidade: o de agente de mobilidade (AM) e o de gerente de mobilidade (GM). O gerente seria o responsável pela coleta e processamento de informações relativas ao deslocamento das estações móveis, criando históricos de mobilidade e fornecendo predições com a especificação de mobilidade para cada estação, obedecendo diferentes critérios de predição e modelos de mobilidade. Os gerentes podem estar distribuídos através de diferentes *áreas de localização*[33] (ALs). Uma abordagem semelhante é observada quando se utiliza o conceito de *clusters*. A idéia por trás dos *clusters* é semelhante à das ALs dinâmicas (Nacif, 2001), ou seja, agrupar células dinamicamente com base na mobilidade do usuário. Pode-se ainda pensar em uma arquitetura hierárquica de gerentes que englobaria diferentes ALs. Os gerentes estariam interagindo de forma a obter itinerários que abrangessem uma maior área geográfica, de acordo com a necessidade de predição de cada unidade móvel. No modelo centralizado, os gerentes poderiam atuar nas próprias ERBs das ALs ou, utilizando os conceitos introduzidos pelo IP Móvel, poderiam estar associados aos HAs dos nós móveis ou mesmo aos FAs, quando o nó móvel estivesse visitando uma nova localidade.

[33] ALs são o resultado da divisão da área de serviço total em áreas geográficas menores. O planejamento das ALs é uma tarefa complexa e consiste na definição da localização e abrangência de cada AL, ou seja, quais as células que as compõem.

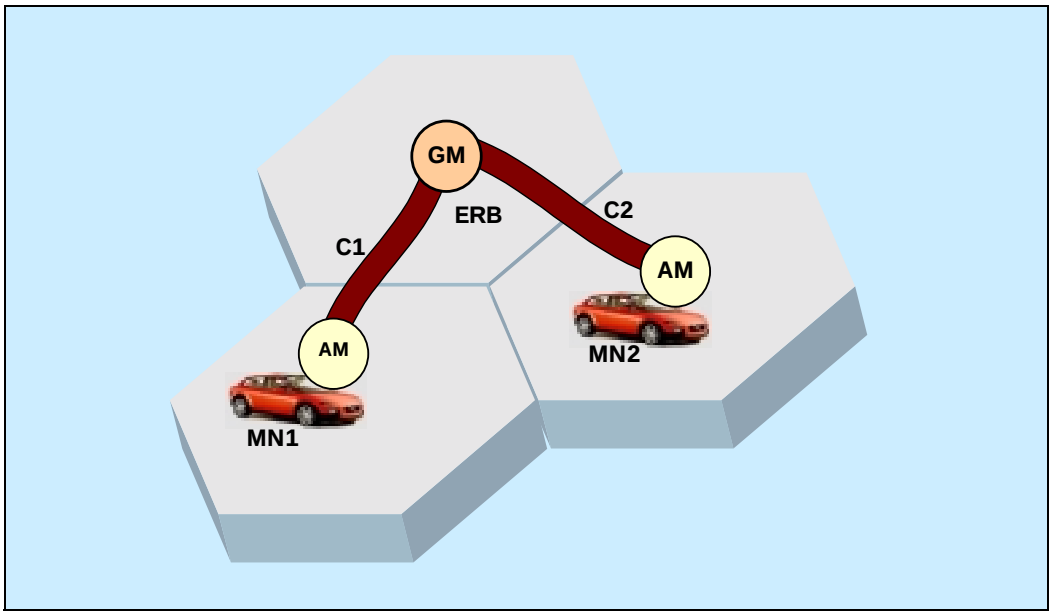

Figura 3.10 Abordagem centralizada de gerenciamento de mobilidade

Nesta abordagem, o agente de mobilidade é responsável pela interação com o gerente. Através do seu agente, a unidade móvel solicita ao gerente de mobilidade a atualização das informações referentes à sua localização atual, podendo ainda requisitar informações referentes aos deslocamentos futuros. Para tanto, o usuário deverá indicar o critério de predição a ser utilizado, ou mesmo o modelo de mobilidade ao qual pertence. Outro ponto que pode ser abordado é a forma como a comunicação entre agentes e gerentes deverá ser efetuada, ou seja, a escolha do protocolo de comunicação e do formato de mensagens utilizado. Vale ressaltar que esses pontos devem ser considerados independentemente do modelo de gerenciamento utilizado.

- *Abordagem Distribuída*

Nesta abordagem (Figura 3.11), o gerente não pode ser apresentado como um componente propriamente dito, mas será uma abstração representada pela colaboração entre gerentes de mobilidade independentes. O gerenciamento da mobilidade é efetuado em cada unidade móvel, através do seu gerente de mobilidade. Este será acionado localmente pelo agente de mobilidade e se encarregará de estabelecer comunicações com tantos gerentes (externos) quanto for necessário para obter as informações solicitadas pelo usuário, ou seja, a sua especificação de mobilidade. Um comportamento semelhante será verificado quando da atualização das informações sobre a localização atual do nó móvel. Pelo que foi exposto,

podemos ter unidades móveis se comunicando para trocar, ou mesmo atualizar, informações sobre deslocamentos em uma determinada AL ou entre ALs distintas. Este cenário, como se pode observar, se adequa a ambientes *ad hoc*, nos quais não existem elementos fixos centrais aos quais se possa confiar a tarefa de gerenciamento da mobilidade dos usuários.

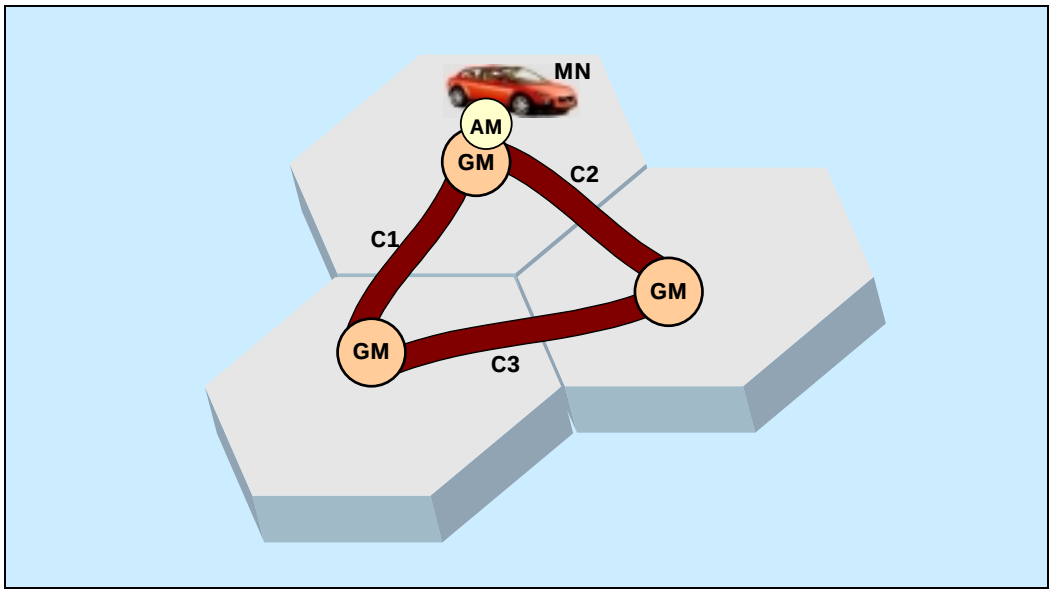

Figura 3.11 Abordagem distribuída de gerenciamento de mobilidade

- *Considerações sobre as Abordagens Centralizada e Distribuída*

Como foi mencionado no início desta subseção, as abordagens centralizada e distribuída podem atuar em conjunto em um cenário de gerenciamento de mobilidade, para atualizar ou ainda fornecer informações relacionadas à mobilidade do usuário. Pode-se avaliar, por exemplo, uma região geográfica na qual existem duas ou mais ALs implementando modelos de gerenciamento de mobilidade distintos, compondo uma hierarquia de ALs. Um gerente de uma AL que fizesse uso de uma estratégia distribuída poderia acionar o gerente de outra AL, com estratégia centralizada, para obter informações a respeito do deslocamento de um usuário específico.

Como foi mencionado na Subseção 2.1.4, o gerenciamento de mobilidade implica na utilização de históricos de mobilidade para alimentar heurísticas de predição e definir modelos de mobilidade. O fato da unidade móvel migrar para regiões as quais não estejam registradas no seu histórico de deslocamentos sugere a utilização conjunta dos dois modelos apresentados. O modelo centralizado poderia ser empregado pelas unidades móveis em

regiões que estivessem retratadas no histórico de deslocamentos, enquanto o modelo distribuído poderia ser adotado quando fosse detectada a ausência de um histórico de deslocamentos para a região visitada. Uma estratégia que poderia ser adotada é a definição da abordagem a ser utilizada em tempo de execução.

Nesta subseção procuramos apresentar as questões envolvidas no processo de definição de uma estratégia de gerenciamento de mobilidade, ilustrando em muitos pontos situações específicas de uma possível solução para avaliar a complexidade envolvida na adoção da mesma.

Um aspecto que foi brevemente mencionado e merece atenção especial são as classes para comunicação que poderão ser utilizadas para a negociação de conteúdos entre agentes e gerentes de mobilidade, representadas na Figura 3.10 como C1 e C2, e entre gerentes de mobilidade de localidades distintas, representadas na Figura 3.11 como C1, C2 e C3. O que se pode observar na literatura é uma forte tendência a se utilizar padrões como o HTTP/1.1 e o CC/PP (1999) (*Composite Capabilities/Preference Profiles*). A combinação do HTTP e do CC/PP oferece suporte à negociação de conteúdos, reduzindo a quantidade de informações trocadas entre os servidores e seus clientes, considerando informações referentes ao contexto do sistema e oferecendo diversas alternativas para a disponibilização de conteúdo. O perfil CC/PP contém alguns nomes de atributos e valores associados que são usados pelo servidor para determinar a forma mais apropriada de entregar um recurso a um cliente. Ele é estruturado para permitir que um cliente descreva as suas capacidades através de referências a um perfil padrão, acessível a um servidor de origem ou outro provedor, e a um conjunto menor de características adicionais que são diferentes do perfil padrão.

A combinação do HTTP/1.1 com o CC/PP permite uma maior flexibilidade em ambientes móveis garantindo que, além de informações referentes à mobilidade do usuário, os agentes e gerentes possam trocar, sem um custo de comunicação adicional, informações relacionadas ao contexto do usuário e do sistema como um todo. Isso irá influenciar o modo como um dado serviço será disponibilizado,

levando-se em consideração, além da localização física do usuário[34], informações gerais de contexto (Schilit & Adams, 1994; Chen & Kotz, 2000).

Pode-se considerar a possibilidade de se adotar diferentes classes para comunicação como um ponto de flexibilização do *framework para gerenciamento de mobilidade*. Tanto o protocolo de comunicação quanto o formato das mensagens podem ser modificados para atender a características específicas de implementação.

3.4.2 Componentes do Framework para Gerenciamento de Mobilidade

A Figura 3.12 traz as classes e os respectivos relacionamentos que constituem o *framework para gerenciamento de mobilidade*. O gerenciador de mobilidade pode ser acionado em duas situações: (i) o nó se deslocou e deseja atualizar o seu histórico de mobilidade ou (ii) o nó deseja obter a sua especificação de mobilidade, informando o critério de predição e a classe para comunicação entre agentes e gerentes que deverá ser adotada. A atualização do histórico é realizada através do método `updateHistory()`, enquanto a solicitação da especificação de mobilidade do usuário é efetuada através do método `reqMSpec()`, ambos pertencentes à classe *MobilityManagement*.

A adoção de um critério de predição em detrimento de outro, a incorporação de perfis de mobilidade ou mesmo a definição de uma classe para comunicação entre agentes e gerentes identificam pontos de flexibilização do *framework para gerenciamento de mobilidade*. Como pode ser observado, a possibilidade de se introduzir novos mecanismos de predição e novos formatos e protocolos para a troca de mensagens é facilitada pela introdução de pontos de flexibilização no *framework* proposto.

[34] Vale ressaltar que a localização do usuário já é uma informação de contexto.

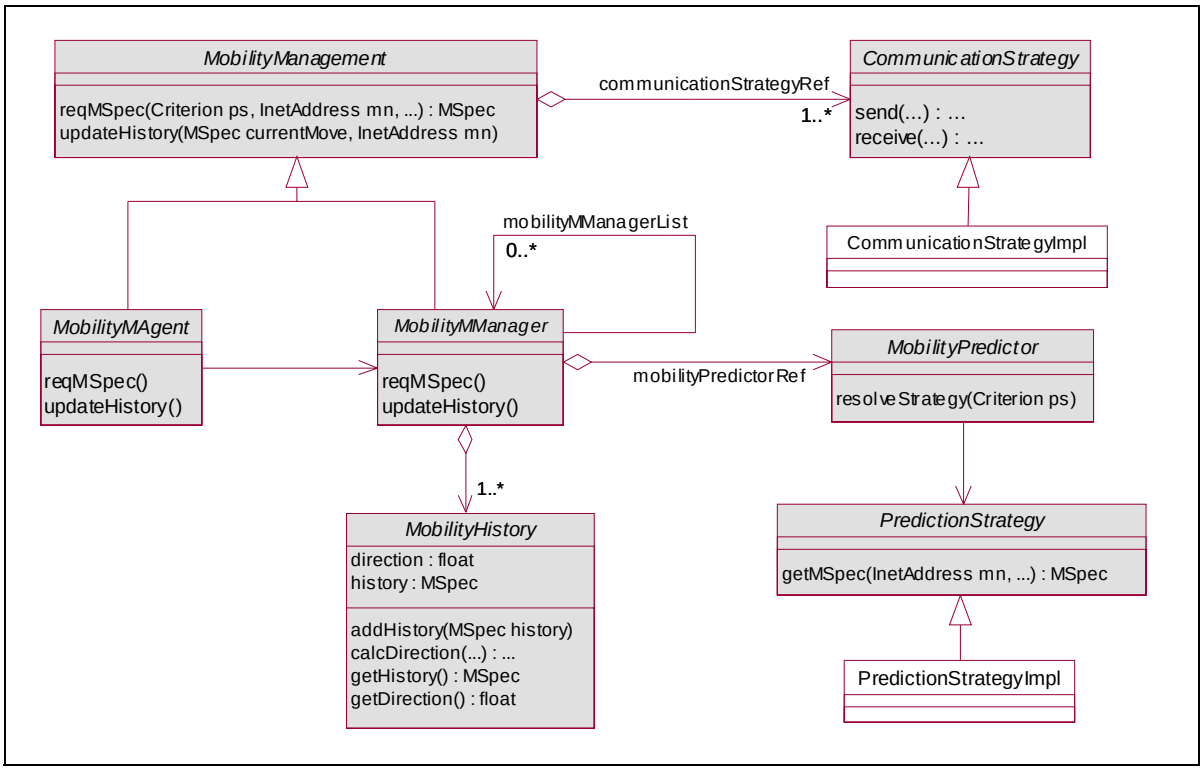

Figura 3.12 *Framework* para Gerenciamento de Mobilidade

O método `updateHistory()` do gerente de mobilidade (*MobilityMManager*) é o responsável pela atualização das informações contidas em *MobilityHistory*. Ao ser acionado, esse método invoca o método `addHistory()` para que as informações sobre o deslocamento atual do usuário possam ser adicionadas ao seu histórico de mobilidade, na lista `historyList`. O método `addHistory()` é o responsável por manter a integridade das informações armazenadas no histórico. Assim, antes que as informações relacionadas a localização atual sejam inseridas para um usuário específico, é necessário que se faça uma verificação entre o resultado apontado pela predição e o deslocamento real efetuado pelo nó (Chan et al., 2000). Essa checagem é importante, visto que, se o resultado da predição tiver incorrido em erro, o histórico deve ser ajustado para evitar que erros gerados por padrões de movimento randômico ou mudanças no padrão de comportamento do usuário se propaguem para as novas predições. Em caso contrário, o histórico é simplesmente atualizado. Esse processo de atualização do histórico pode ser acompanhado através do diagrama de fluxos apresentado na Figura 3.13.

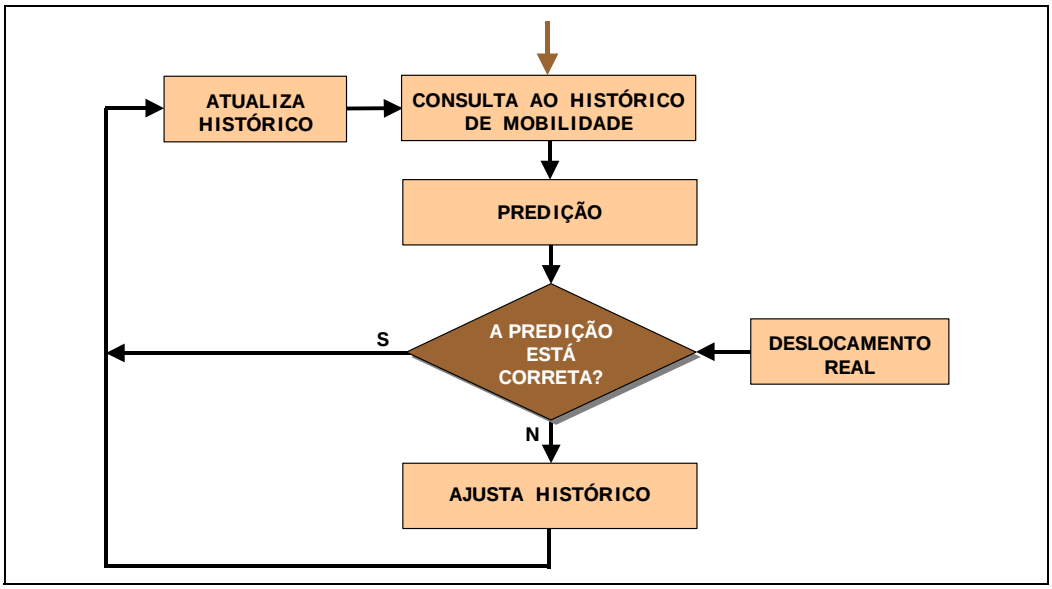

Figura 3.13 Etapas envolvidas no processo de predição

A seqüência de invocação de métodos referente à atualização do histórico de mobilidade do usuário é ilustrada pelo diagrama apresentado na Figura 3.14.

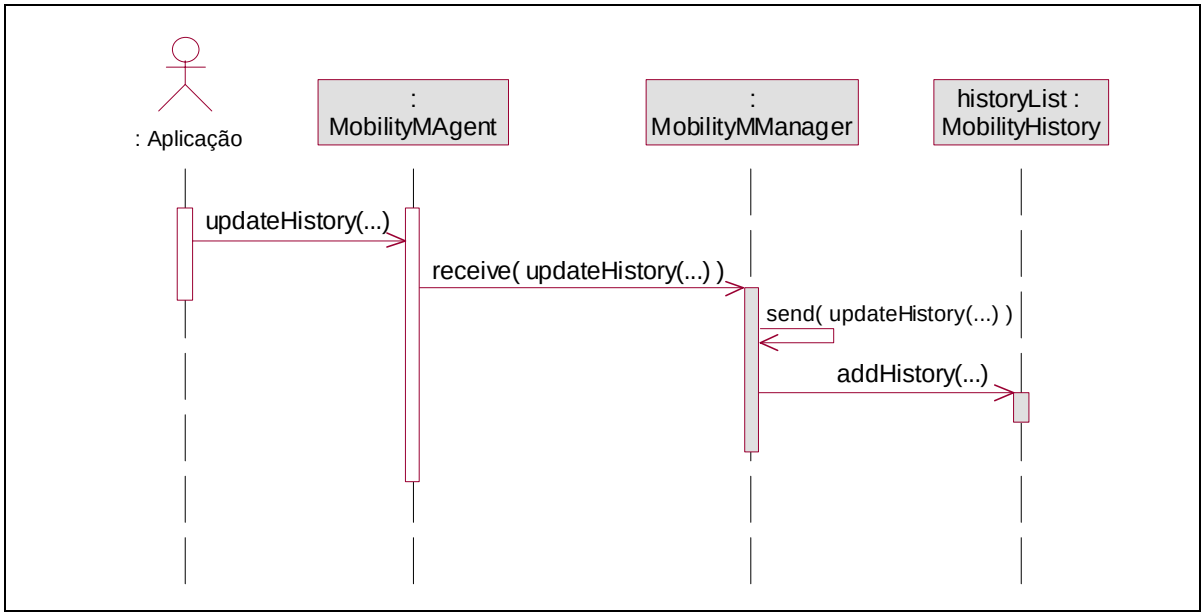

Figura 3.14 Seqüência de invocação de métodos na atualização do histórico de mobilidade do nó

A classe abstrata *MobilityManagement* tem os seus métodos `reqMSpec()` e `updateHistory()` redefinidos nas subclasses *MobilityMAgent* e *MobilityMManager*. A partir dessas redefinições podem ser representadas as responsabilidades de cada um desses componentes no processo de gerenciamento de mobilidade, através dos níveis de abstração envolvidos. O relacionamento recursivo de associação `mobilityMManagerList` (classe *MobilityMManager*) indica a possibilidade de se implementar o gerenciamento de mobilidade de forma centralizada ou distribuída. A

relação de agregação `communicationStrategyRef` entre as classes abstratas *MobilityManagement* e *CommunicationStrategy* define a estratégia de comunicação a ser adotada entre agentes e gerentes através dos métodos `send()` e `receive()`, permitindo que diferentes protocolos de comunicação e formatos de mensagens sejam utilizados.

O gerente de mobilidade, representado pela subclasse *MobilityMManager*, faz referência às estratégias de predição através da associação `mobilityPredictorRef` com a classe abstrata *MobilityPredictor*. Ao ter o método `reqMSpec()` disparado, o gerente chama o método `resolveStrategy()` para definir, a partir da indicação do usuário, qual estratégia de predição deve ser utilizada. O gerente passa então a alimentar a estratégia de predição escolhida (acionada pelo método `getMSpec()` de *PredictionStrategy*) com o histórico de mobilidade do usuário, representado pela classe *MobilityHistory*. Uma adaptação ao *pattern* comportamental Strategy (Gamma et al., 1995) pode ser verificada através do relacionamento entre as classes *MobilityMManager* e *MobilityPredictor*, o que permite que uma instância da classe *MobilityMManager* seja configurada com um objeto da classe *PredictionStrategy* especificado na solicitação do usuário (`reqMSpec(Criterion ps,...)`). A Figura 3.15 ilustra através de um diagrama a seqüência de invocação dos métodos envolvidos no processo de predição da mobilidade do usuário.

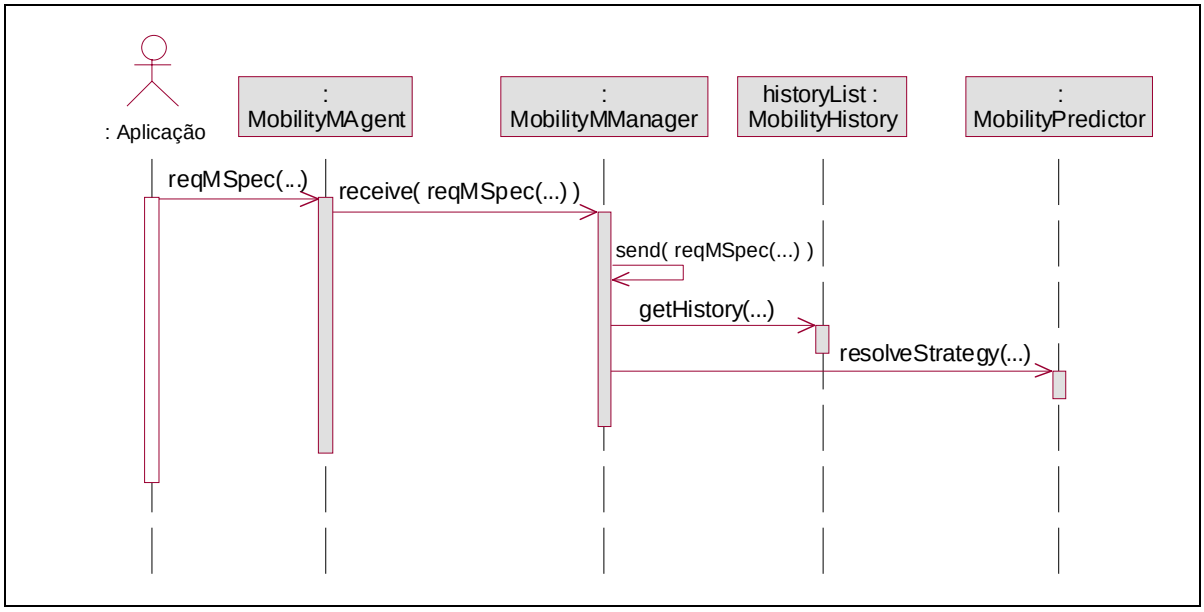

Figura 3.15 Seqüência de invocação de métodos na predição da mobilidade

3.4.3 Exemplo de Aplicação do Framework para Gerenciamento da Mobilidade

A Figura 3.16 dá continuidade ao exemplo apresentado na Subseção 3.1.2 de um ambiente *intserv* móvel com estações e roteadores que oferecem suporte ao IP Móvel e ao MRSVP, ilustrando um caso de uso do *framework para gerenciamento de mobilidade*. Iremos acompanhar o acionamento do mecanismo de predição de mobilidade originado por um nó móvel que deseja efetuar uma reserva de recursos usando o MRSVP. Como foi mencionado na Subseção 2.2.1, o MRSVP pressupõe a existência de um arquivo de especificação de mobilidade do usuário para que se possam estabelecer reservas antecipadas nas localidades descritas nesse arquivo. Como foi visto, a garantia de uma provisão de QoS fim-a-fim, no MRSVP, depende da precisão dessa informação. Neste exemplo, o *framework para gerenciamento de mobilidade* estará atuando de modo a fornecer as especificações de mobilidade do usuário para que as suas reservas antecipadas possam ser efetuadas.

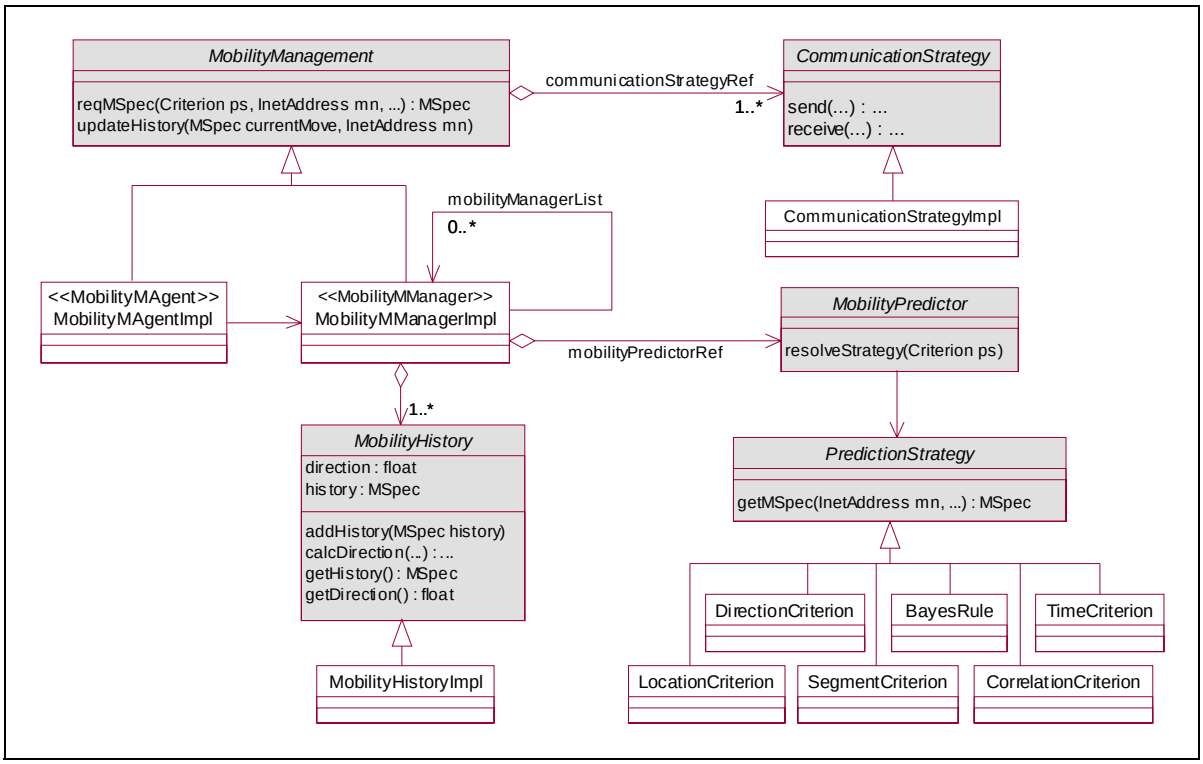

Figura 3.16 Exemplo de aplicação do *Framework* para Gerenciamento de Mobilidade

Durante a requisição da QoS, ao definir a categoria de serviço à qual pertence o seu fluxo, o nó móvel irá disparar o processo de predição a partir do método `reqMSpec()` contido na instanciação de *MobilityMAgent*. Internamente, cada nó móvel faz a solicitação a partir do objeto *MobilityMAgentImpl* e este se encarrega de acionar o objeto *MobilityMManagerImpl*, que é o responsável pelo gerenciamento das

informações de mobilidade do usuário. A comunicação entre agentes e gerentes é definida através do objeto *CommunicationStrategyImpl*. A implementação dos métodos `send()` e `receive()` definirá o protocolo de comunicação e o formato das informações trocadas. No exemplo da Figura 3.16, uma possível instanciação dos métodos de *CommunicationStrategy* pode ser realizada, como comentado na Subseção 3.4.1, através da utilização do HTTP 1.1 e do CC/PP.

O gerente de mobilidade representado pelo objeto *MobilityMManagerImpl* é o responsável pela manutenção das informações contidas no histórico de mobilidade do usuário. O deslocamento efetuado pelo nó móvel será informado ao gerente através da implementação do método `updateHistory()` (*MobilityMManager*), que se encarregará de invocar o método `addHistory()` do objeto *MobilityHistoryImpl* para que as informações sobre o deslocamento atual do usuário possam ser verificadas e posteriormente adicionadas ao histórico de mobilidade do usuário, representado pela lista `historyList`.

Através do critério de predição definido pelo usuário na sua requisição, o gerente de mobilidade (*MobilityMManagerImpl*) irá consultar as informações de modo a retornar os possíveis deslocamentos futuros do usuário. O que ocorrerá é a tradução do critério passado pelo usuário, através do método `reqMSpec()`, no *MobilityPredictor* pelo método `resolveStrategy()`. A partir da definição da estratégia de predição adotada, o gerente definirá então a especificação de mobilidade do usuário, retornando o seu *MSpec* através do método `getMSpec()` da instanciação de *PredictionStrategy*. Como se pode observar na Figura 3.16, a classe abstrata *PredictionStrategy* foi instanciada de modo a adotar um dos critérios de predição mencionados na Subseção 2.1.4.1.

3.5 SUMÁRIO

Neste capítulo foram apresentados os *frameworks para provisão de QoS em ambientes genéricos de processamento e comunicação*. Uma instanciação dos *frameworks* foi ilustrada levando em consideração as características inerentes aos ambientes móveis, motivando o estabelecimento de reservas antecipadas, a atribuição de uma maior flexibilidade nos algoritmos de controle de admissão através da introdução do

conceito de intervalos de QoS e a aplicação das políticas de controle, definindo a permissão de acesso dos usuários móveis nas diferentes localidades visitadas.

Para que a pré-alocação de recursos fosse realizada de uma forma eficiente foi proposto um mecanismo capaz de predizer os deslocamentos dos nós, modelado conceitualmente através do *framework para gerenciamento de mobilidade*. Esse novo mecanismo foi incorporado ao modelo genérico sendo o responsável pela administração das informações referentes à mobilidade dos usuários e pela predição de seus futuros deslocamentos. Também foram avaliadas as abordagens centralizada e distribuída e uma possível colaboração entre as duas.