SÉRGIO COLCHER

UM META MODELO PARA APLICAÇÕES E SERVIÇOS DE COMUNICAÇÃO ADAPTÁVEIS COM QUALIDADE DE SERVIÇO

TESE DE DOUTORADO

DEPARTAMENTO DE INFORMÁTICA

Rio de Janeiro, 22 de Novembro de 1999

SÉRGIO COLCHER

UM META MODELO PARA APLICAÇÕES E SERVIÇOS DE COMUNICAÇÃO ADAPTÁVEIS COM QUALIDADE DE SERVIÇO

Tese apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para a obtenção do título de Doutor em Ciências em Informática.

Orientador: Prof. Luiz Fernando Gomes Soares

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 22 de Novembro de 1999

Esta tese é dedicada aos meus pais, que sempre me apoiaram em todos os momentos de minha vida, me ensinando a lutar pelos sonhos e a ser uma pessoa mais digna. Dedico-a também à memória do Prof. Sérgio Carvalho, que infelizmente nos deixou antes de ter podido participar deste marco em minha vida. Mas seus ensinamentos, simpatia, irreverência e dedicação me acompanharão para sempre.

Sérgio Colcher

Agradecimentos

O primeiro agradecimento vai para o meu orientador Prof. Luiz Fernando Gomes Soares. Esta tese jamais teria se concretizado se não fosse por seus cuidados, incentivos, críticas, discussões e inspirações. Além disso, não apenas foi ele sempre um exemplo de dedicação, corretude e dignidade, mas também foi, sem dúvida, aquele que proporcionou todo o crescimento profissional pelo qual tenho passado nesses anos em que convivemos.

O segundo agradecimento é não menos importante. Ele vai ao amigo Luiz Fernando Gomes Soares, que sempre me apoiou e confiou em mim e em meu trabalho de forma incondicional, nunca deixando que eu desanimasse, estando sempre presente mesmo quando as coisas "apertaram". Entre outras coisas, me ensinou que é preciso ser paciente, humilde e que "nacemu e tamo pronto pa morrê".

Aos colegas do Laboratório TeleMídia, agradeço por todos os momentos agradáveis de convívio, sempre fazendo do ambiente de trabalho um local aprazível. Em especial, esta tese jamais teria se realizado sem o trabalho e dedicação do colega e amigo Antônio Tadeu Azevedo Gomes, que teve a paciência de me ouvir e de me ajudar quando eu estava realmente precisando de ajuda. Não posso me esquecer de Marcus Antônio A. Rodrigues, Rogério Rodrigues e Débora Muchaluat, amigos sempre dispostos e prontos a ajudar.

Aos colegas do DI da PUC-Rio que, depois desses anos de convívio e dificuldades pelas quais passamos, também considero amigos. Em especial, gostaria de agradecer a Alexandre Plastino e a Simone de Lima Martins por todo o apoio e pela convivência enriquecedora.

Aos amigos do "chopp de quarta" que já é uma instuição nesse departamento, pelos momentos de descontração e alegria. Em especial, gostaria de citar Lorenzo Ridolfi, Marcelo Duarte, Simone Martins, Rogério Rodrigues, Eduardo Uchôa e Rodrigo Uchôa.

A todo o Departamento de Informática, seus professores e funcionários, pela formação de alto nível que proporcionam a seus alunos.

Por fim, à Embratel, cujo apoio dado ao Laboratório TeleMídia permitiu que esta tese fosse realizada, e ao CN$_p$q, pelo suporte financeiro dado a esse trabalho.

RESUMO

UM dos maiores desafios apresentado às prestadoras de serviços de comunicação tem sido o de projetar uma infra-estrutura única, que permita tanto a integração da mais variada gama de diferentes serviços, como a criação de um ambiente no qual esses mesmos serviços possam ser rápida e facilmente criados, alterados e continuamente adaptados a novas necessidades ou condições. A variedade de requisitos e a rapidez dos avanços tecnológicos têm sido grandes empecilhos, especialmente quando a estratégia adotada consiste em tentar acomodar os diversos serviços utilizando-se como base apenas a configuração de algumas opções pré-definidas. O presente trabalho visa propor um modelo que forneça abstrações para uma forma mais dinâmica de programação e adaptação dos aspectos de QoS e comunicação de grupo (multicasting) em serviços de comunicação multimídia. Uma das principais características do modelo é seu caráter recursivo, baseado em um *estilo arquitetural* da forma *pipes e filtros*. Pipes correspondem à representação explicita da conexão entre componentes (os filtros) que, por sua vez, representam os elementos de processamento da implementação de um serviço. Especificações de QoS poderão ser associadas a pipes, que passam então a ser denominados *MediaPipes*. MediaPipes permitirão manipular e programar, de forma *abstrata*, os aspectos de comunicação e QoS desejados ao mesmo tempo em que tornarão possível, quando desejado, expor os detalhes de implementação dessas abstrações de forma a torná-las alvo das adaptações necessárias.

ABSTRACT

ONE of the main chalenges posed to the telecommunications services operators is to develop a single infrastructure to support the integration of a wide range of services and to allow for the rapid creation, alteration and continuous adaptation of these services to new evolving conditions or demands. The variety of requirements and the rapid evolving technology impose many difficulties, specially when the goal is to develop services based on a limited number of pre-defined options. This work presents a model in which communication services platforms will be constructed to support some kind of *adaptability* in order to meet the wide range of different demands, the presence of run-time changes in applications requirements and the rapid evolving technology. The model provides some basic abstractions for the representation of multicasting and Quality of Service (QoS) within multimedia communication services and is based on the *pipes-and-filters* architectural style. Pipes correspond to the explicit representation of connections between components (the filters) that, in turn, represent the active processing elements of a service implementation. QoS specifications can be asscociated to pipes, which are then called *MediaPipes*. MediaPipes correspond to an abstract programming entity for the QoS treatment and also allow for the exposure of internal implementation details whenever adaptations are necessary.

Sumário

Lista de Figuras

Capítulo 1

Introdução

NO atual estágio de desenvolvimento e competitividade do setor de telecomunicações, a habilidade de se oferecer uma vasta gama de serviços com características diferenciadas e perfeitamente sintonizadas com os requisitos das aplicações tem se tornado imprescindível [79]. Nos tempos atuais, serviços devem estar prontamente disponíveis em resposta às necessidades dos usuários, numa velocidade bem maior do que em um passado não muito distante. Em um cenário como esse, diversas atividades como planejamento, projeto, implementação e implantação têm se demonstrado tão complexas a ponto de justificar a criação de uma "engenharia de serviços de telecomunicações" [139, 138], fruto da competitividade e globalização da era da informação.

O primeiro objetivo deste capítulo é a caracterização dos problemas existentes nessa nova área que pode ser, de certa forma, considerada como parte da intersecção de outras áreas como telecomunicações, comunicação de dados, engenharia de software e sistemas multimídia. Os novos desafios que têm sido lançados à indústria de telecomunicações e às operadoras de serviços (advindos, principalmente, da desregulamentação dos mercados, da crescente demanda dos consumidores, da variedade de serviços vislumbrados, etc.) parecem justificar essa convergência, revelando problemas inerentes a essa combinação e, ao mesmo tempo, ausentes em cada uma daquelas áreas isoladas. Em última análise, justamente da complexidade proveniente dessa interdisciplinaridade é que surge a motivação para a realização do presente trabalho, cujo objetivo maior é fornecer um ponto de apoio e referência para um maior desenvolvimento de uma área que encontra-se, hoje, ainda em seus primórdios. Ao longo do capítulo, esse objetivo se tornará mais claro e uma visão geral da tese será apresentada.

1.1 Motivação

A grande evolução da tecnologia digital observada nas últimas décadas foi a maior responsável pela explosão de serviços multimídia em sistemas de processamento e comunicação de dados. Tal evolução permitiu, entre outros avanços, o surgimento de dispositivos de armazenamento de grande capacidade, a proliferação de dispositivos de entrada e saída de alta qualidade para a aquisição e apresentação das diversas mídias, o projeto de circuitos integrados dedicados a compressão de informação com alta qualidade e em tempo real, e o desenvolvimento de serviços e dispositivos de comunicação para redes capazes de transmitir informação digital de qualquer natureza (vídeo, texto, voz, etc.) de forma integrada [123]. Recursos, dispositivos, algoritmos

e pastilhas de circuitos integrados como os mencionados acima já estão, hoje, largamente disponíveis no mercado, sendo comum encontrá-los nas configurações básicas dos computadores pessoais.

A disponibilidade desses recursos alimentou a imaginação dos desenvolvedores quanto às novas possíveis aplicações. A integração de serviços em redes de comunicação foi motivada não só pela possibilidade e viabilidade de se obter economias na utilização de recursos por meio do compartilhamento de linhas de transmissão, equipamentos de comutação etc, mas também pela expectativa quanto aos novos serviços de banda larga e aplicações multimídia distribuídas que poderiam ser desenvolvidos.

Vários cenários foram delineados de forma a caracterizar os efeitos da tecnologia de integração de serviços na sociedade. Imagina-se, por exemplo, que médicos em hospitais localizados em regiões diferentes serão capazes de discutir o diagnóstico de um paciente usando informações de raio X, histórico da evolução do caso, vídeos, banco de dados sobre acompanhamentos de casos semelhantes, etc. Os participantes dessa teleconferência multimídia deverão ser ainda capazes de trocar comentários (falados) e, simultaneamente, apontar os diversos ítens sob discussão. Todos os fluxos de voz, imagens e texto deverão ser comunicados a todos os participantes e todo o sincronismo necessário (por exemplo, uma imagem em movimento sendo referenciada e comentada oralmente) deverá ser preservado [124]. Outros cenários semelhantes podem também ser construídos com base em aplicações não menos importantes como educação à distância [84, 26], entretenimento [28, 27] e comércio eletrônico [47].

Cenários como esse sugerem o alto grau de complexidade exigido dos serviços de comunicação e as dificuldades associadas ao projeto dessas aplicações. Em relação aos serviços, a grande dificuldade consiste em tratar de toda a diversidade de tipos de fluxo, cada qual com suas próprias características, e os relacionamentos de sincronização que podem existir entre eles. Três fatores podem ser destacados como agravantes dessas dificuldades: a questão da *qualidade de serviço*, a questão da *comunicação de grupo*, e a questão da *rápida evolução tecnológica*. Alguns detalhes sobre essas questões encontram-se, respectivamente, nas Seções 1.1.1, 1.1.2 e 1.1.3. Desses fatores, emerge a necessidade dos *serviços adaptáveis*, capazes de evoluir ou sofrer modificações de acordo com requisitos específicos. Na Seção 1.1.4, apresentam-se alguns aprofundamentos sobre essa questão.

A idéia de serviços e aplicações adaptáveis surge para lidar com a diversidade dos serviços mas, em contrapartida, impõe novas questões. O desenvolvimento das aplicações será necessariamente mais complexo do que era o desenvolvimento de aplicações que se utilizam de serviços prontos (com opções predefinidas e em número limitado). Mecanismos que permitam lidar com essa complexidade deveriam ser fornecidos de forma expor essa complexidade adicional apenas nos momentos em que as adaptações se fizerem necessárias e, no restante do tempo, fornecer abstrações de alto nível que permitam a utilização da infra-estrutura de serviços de uma forma mais simples. Alguns comentários sobre tais mecanismos serão apresentados na Seção 1.1.5.

1.1.1 Qualidade de Serviço

As características e requisitos de comunicação exigidos pelos diversos tipos de mídia são, por si só, muito diferentes. Como se isso não bastasse, a dimensão de "qualidade" (relacionada à alguma forma de percepção humana) associada a cada mídia é não apenas dependente de seu tipo (áudio, vídeo, etc.), mas também das características específicas da aplicação ou serviço em questão. Ao ouvir música, por exemplo, exige-se, em geral, uma qualidade de áudio bastante

superior do que aquela desejada numa conversa telefônica. Assim, serviços de comunicação multimídia devem atender aos requisitos de qualidade impostos por qualquer combinação dos tipos de mídia e aplicações. O aumento na diversidade dessas combinações tem gerado a demanda por um número grande de serviços, cada um com uma característica própria de qualidade, denominada *qualidade de serviço (Quality of Service — QoS).*

A complexidade de se implantar um sistema de comunicação que ofereça suporte a todos esses serviços é muito grande, principalmente ao se levar em conta que novos serviços podem ser constantemente exigidos em resposta a novas formas de codificação ou compressão das mídias, ou a novas classes de aplicações que podem mudar os requisitos de qualidade já existentes.

1.1.2 Comunicação de Grupo

A comunicação de grupo tem sido apontada como capacidade fundamental para o suporte adequado a aplicações multimídia [98, 112]. A criação dos sistemas multimídia pode ser, na realidade, encarada como um avanço na forma de colaboração e comunicação humana através de máquinas computacionais. É de se esperar que essas aplicações serão amplamente beneficiadas por serviços de comunicação de grupo (multicast). O cenário da aplicação de vídeo-conferência médica sugerido anteriormente já exemplifica essa necessidade. Várias outras aplicações são naturalmente associadas a essa capacidade, incluindo aplicações de trabalho cooperativo em geral. Além disso, um dos maiores mercados das redes com integração de serviços deverá ser o de vídeo sob demanda (Video on Demmand — VoD), no qual a distribuição seletiva assume um papel fundamental.

A distribuição das informações de forma eficiente aos participantes ou usuários de um sistema é um dos maiores desafios apresentados aos prestadores de serviços de comunicação, ainda mais quando QoS pode variar não mais apenas com o tipo de mídia e aplicações, mas também com a capacidade que cada equipamento de usuário (transmissor ou receptor) possui. O provedor de serviço deverá dispor dos mecanismos adequados de roteamento e controle de recursos (como buffers, banda passante, etc.) que, ao mesmo tempo, possam garantir a qualidade desejada a cada participante em um serviço ou aplicação de multicast e ainda otimizar a utilização desses recursos.

1.1.3 Evolução Tecnológica

A complexidade exigida dos serviços de comunicação é ainda maior quando se considera que a evolução das tecnologias básicas de transmissão e comutação tem ocorrido de forma cada vez mais veloz, fazendo com que protocolos e mecanismos presentes nos sistemas de comunicação se tornem, muitas vezes, inadequados às novas necessidades de desempenho [45]. Mecanismos reativos de controle de congestionamento, por exemplo, que são aplicáveis a redes com baixas taxas de transmissão, deixam de ser razoáveis quando as taxas de transmissão crescem o suficiente para tornar o tempo de transmissão significativamente menor do que o de propagação

nos meios físicos [124].[1] Além disso, sistemas de comunicação podem, muitas vezes, não ser suficientemente escaláveis para atender prontamente a um aumento inesperado de demanda. O exemplo clássico advém do crescimento da Internet que, superando todas as expectativas iniciais, levou rapidamente ao quase esgotamento dos endereços IP disponíveis, provocando uma corrida na direção da nova versão IPv6. Porém, dada a forma com que as implementações foram e ainda são feitas hoje em dia, a migração da Internet deverá se estender por um período da ordem de anos, durante os quais será inevitável uma convivência entre as diferentes versões (conseqüentemente implicando na existência esquemas de compatibilização e interoperabilidade) até que a nova versão esteja completamente implantada.

É claro que se os avanços tecnológicos tendem a ser cada vez mais rápidos, um ambiente no qual modificações são predominantemente lentas torna-se, no mínimo, um contra-senso. A rapidez na obsolescência de equipamentos, protocolos e serviços tem que passar a ser tratada como ponto estratégico das operadoras, pois representa o perigo da não preservação de alguns altos investimentos, tanto em equipamentos como em desenvolvimento, treinamento, etc.

1.1.4 A Necessidade de Adaptabilidade

O ideal almejado pelas operadoras de serviços é projetar uma infra-estrutura de comunicação que exiba duas características que são, de certa forma, antagônicas: (i) a integração da mais variada gama de diferentes serviços em um único e eficiente sistema de comunicação, e (ii) a criação de um ambiente no qual esses mesmos serviços possam ser rápida e facilmente criados, alterados e continuamente adaptados às novas necessidades e condições.

Torna-se cada vez mais difícil encontrar uma arquitetura única, genérica o suficiente para acomodar os diversos serviços e adaptações através de simples parametrizações ou opções predefinidas. A alternativa, recentemente considerada, é a definição de modelos nos quais arquiteturas de protocolos e serviços podem ser dinamicamente programados ou adaptados ao longo de seu ciclo de vida. Essa *adaptabilidade* é uma característica desejável tanto para acomodar e absorver os avanços tecnológicos que acontecem cada vez mais rapidamente, como para permitir a criação de novos serviços e aplicações que apresentam requisitos específicos de QoS.

Uma conseqüência direta desse modelo é que, como os diferentes requisitos podem ser oriundos de aplicações específicas, parte da responsabilidade de implementação ou configuração dos sistemas de comunicação poderá ser delegada a projetistas das próprias aplicações. De fato, alguns autores, como Lazar [80, 79] e Campbell [17], consideram que o projeto, implementação e configuração de serviços de comunicação deverá se tornar tão corriqueiro quanto o projeto da aplicação em si, estando ambas as tarefas interligadas e completamente integradas. Essa "programabilidade" dos serviços, segundo Lazar, deverá ser uma característica essencial em qualquer ambiente distribuído moderno.

[1] Considere dois nós interligados por um meio físico a uma distância de 100 km. Assumindo pacotes de 500 bits (uma célula ATM tem 424 bits) e um tempo de propagação de 5 μseg por quilômetro, se a taxa de transmissão for de 1 Mbps, o tempo de transmissão de uma pacote é igual a $\frac{500\,bits}{1\,Mbps} = 500\,\mu$seg. Como o tempo de propagação entre os nós também é de $500\,\mu$seg, o mecanismo reativo é viável, pois é possível ajustar a transmissão baseando-se em informações de realimentação do receptor, que se consegue enviar antes que um número grande de pacotes já tenham sido encaminhados pelo transmissor. Mas se a taxa de transmissão for de 1Gbps, por exemplo, o tempo de transmissão passará a ser de $0.5\,\mu$seg, o que significa dizer que, quando os primeiros bits de um pacote estiverem chegando no receptor, outros 1000 pacotes já estarão a caminho [124, pág. 621]. Nesse caso, a realimentação será inócua.

1.1.5 Abstração e Reificação

Nesse novo ambiente de desenvolvimento sugerido na Seção 1.1.4, serviços deverão ser projetados levando em consideração a possibilidade de que adaptações poderão ser feitas ao longo das várias fases do ciclo de vida dos serviços (incluindo sua operação), tanto por usuários (que poderão adaptar serviços de acordo com suas necessidades de QoS) como por desenvolvedores dos equipamentos e protocolos básicos (que poderão, eventualmente, atualizar a implementação dos serviços em função de avanços ou modificações em padrões, correções de problemas ou atualização de versões).

O ciclo de vida dos serviços, que outrora se apresentava em fases mais ou menos distintas de construção, implantação, operação e remoção, deve ser revisto de forma a refletir o dinamismo e a possível superposição ou embaralhamento dessas fases. De forma análoga, as responsabilidades de operadoras, consumidores e desenvolvedores, que antes eram bem definidas em relação a essas fases, também deverão sofrer modificações.

Em particular, desenvolvedores das aplicações podem passar a ser, além de consumidores dos serviços, também participantes do processo de evolução da infra-estrutura desses serviços de comunicação. A complexidade desse processo poderá se tornar um empecilho se ele não estiver apoiado em um modelo adequado, que integre as duas visões (programação de aplicações e implementação de serviços) [24] e que, ao mesmo tempo, as revele ou esconda nos momentos oportunos. Mais especificamente, ao dedicar-se aos aspectos mais ligados às questões de alto nível, provenientes das dificuldades e requisitos puramente funcionais das aplicações, desenvolvedores deverão dispor de mecanismos que os permitam manipular e programar, de forma *abstrata*, os aspectos de comunicação e QoS desejados. Por outro lado, em determinadas circunstâncias, deverá poder ser possível ser exposto aos detalhes de implementação dessas abstrações, para *reificá-las*[2] de forma a torná-las alvo das adaptações necessárias.

1.2 Objetivos

O presente trabalho visa propor um modelo que forneça mecanismos (ou abstrações) para a representação e programação de aspectos de QoS e comunicação de grupo (multicasting) em serviços de comunicação multimídia. Uma das principais características do modelo é seu *caráter recursivo*, que faz com que a realização dos mecanismos mencionados seja modelada por meio da utilização de um outro serviço, de um nível de abstração mais baixo, mas que contém os mesmos mecanismos de abstração sobre QoS e multicasting do nível superior. Sob a ótica desse modelo, o desenvolvimento de aplicações multimídia será considerado como a construção de um serviço comunicação do nível mais alto de abstração.

Aplicações multimídia poderão ser construídas sobre os serviços de comunicação especificados que, por sua vez, poderão ser sucessivamente construídos sobre serviços mais simples. Dessa estratégia, espera-se obter dois benefícios: (i) um mecanismo para o controle da complexidade de forma que os aspectos da implementação dos serviços de comunicação do nível inferior poderão ser ora revelados ao nível superior (com o intuito de efetuar as adaptações desejadas), ora abstraídos (de forma a esconder os detalhes em abstrações de alto nível que simplificam a utilização desses serviços); e (ii) uma diminuição da carga cognitiva, conseqüência da utilização de um mesmo modelo para os diferentes níveis de abstração.

[2] Reificação: transformação em “coisa” [63]; do lat. res, rei, “coisa” + -ficar + -ção [62].

Ao se considerar o aspecto da adaptabilidade, não se pode negligenciar o fato de que o impacto no processo de construção e operação dos serviços não se verifica apenas sob os aspectos técnicos, ou de suporte computacional, mas também organizacional. Não se tem por objetivo, neste trabalho, propor uma nova metodologia de desenvolvimento nem tão pouco um novo paradigma para operação e administração de serviços. Considerou-se fundamental, porém, salientar os pontos em que um ciclo de vida (que inclui a utilização de algum método de desenvolvimento) será influenciado pela possibilidade e dinamismo das adaptações que poderão ser efetuadas sobre os serviços. Sob esse aspecto, o modelo deverá fornecer uma base conceitual por meio da qual certas propriedades de serviços de comunicação adaptáveis poderão ser compreendidas e comparadas.

Qualquer implementação de um ambiente de suporte a serviços deverá conter, porém, outras propriedades não capturadas pelo modelo aqui proposto. De fato, a intenção deste trabalho não será a de propor um modelo que cubra todos os aspectos necessários a uma implementação, mas sim a de estabelecer uma base conceitual que tornará possível o desenvolvimento de outros modelos mais específicos. Dentro dessa linha de raciocínio, é importante frisar que o presente modelo *não deve ser interpretado* como um "modelo definitivo" que, quando implementado, irá sobrepujar ou tornar quaisquer outros obsoletos. Ao contrário, o modelo será, propositadamente, incompleto demais para ser diretamente implementado, refletindo o fato de que o objetivo foi sempre o da generalidade. Esse é um dos sentidos em que ele pode ser considerado um "meta modelo".

1.3 Visão Geral da Estratégia Adotada

Vários trabalhos têm enfocado a forma com que serviços de comunicação são desenvolvidos, operados e mantidos. Alguns deles tratam do problema sem propor mudanças efetivas no ciclo de vida dos serviços, preferindo enfocar somente o lado da rapidez no processo de criação dos serviços. Nessa perspectiva, para adaptar ou criar novos serviços, desenvolvedores seguem o ciclo de vida normal, construindo novos serviços, desativando os antigos e colocando os novos em funcionamento. O processo de construção é porém acelerado através do uso de técnicas de engenharia de software que, de uma forma geral, procuram aumentar o reuso dos produtos gerados nas fases que precedem a operação. Produtos reaproveitáveis incluem todo o tipo de documentação, juntamente com as devidas decisões de projeto e implementação, além do próprio código.

As técnicas que têm sido utilizadas para incrementar o reuso e agilizar o processo de desenvolvimento de serviços incluem:

- a utilização de *design patterns* [53, 13], aplicados ao domínio específico da construção de serviços, protocolos e software de comunicação [116, 115, 117, 41, 42, 88, 133],
- a criação de linguagens específicas para construção de protocolos e serviços de comunicação [1, 102, 101], e
- o desenvolvimento de bibliotecas específicas [113, 118, 65, 92, 95], juntamente com a introdução de mecanismos de suporte nos sistemas operacionais [66, 38, 39, 91, 114, 60].

Porém, todas essas técnicas não lidam diretamente com a possibilidade de adaptações mais dinâmicas, realizadas ao longo da operação de um serviço.

No presente trabalho, procurou-se uma estratégia que permitisse aliar as técnicas acima mencionadas com outras que influenciassem no ciclo de vida dos serviços e permitissem formas de adaptação mais dinâmicas. Um dos pontos de partida foi o estudo da utilização de *frameworks* na modelagem de serviços de comunicação [57, 112] que, embora ainda não tenham sido, nessa área específica, tão explorados quanto os design patterns, já tiveram sua importância reconhecida em algumas instâncias [61].

Um framework [104, 105] difere de uma aplicação pelo fato de que algumas de suas partes (denominadas *pontos de flexibilização* ou *hot-spots*) são explicita e propositadamente deixadas incompletas ou marcadas como "sujeitas a variações". Através do respectivo preenchimento dessas "lacunas" ou de modificações nos pontos marcados, pode-se obter a adaptabilidade do serviço. O momento dentro do ciclo de vida de um serviço em que hot-spots são completados, que pode variar desde as fases de construção e implementação até as fases de operação, determina o seu grau de dinamismo. Os hot-spots complementados nas fases de construção (mais precisamente na fase de *instanciação*[3] do framework) dão origem a serviços específicos que, por sua vez, ainda podem apresentar outros hot-spots a serem complementados, modificados ou estendidos em tempo de operação.

A experiência com a modelagem de dois frameworks, um que trata dos mecanismos de provisão de QoS [57] e outro que trata dos mecanismos de multicast [112], revelou alguns aspectos peculiares, tanto da modelagem de frameworks relacionados à comunicação, como da forma de implementação dos hot-spots nesse tipo de serviço. Tornou-se evidente a necessidade de um modelo que facilitasse a representação explícita da cooperação entre os componentes definidos pelo framework. Essa representação deverá servir como a base para a especificação das abstrações de alto nível sobre os aspectos de QoS e multicast. Além disso, a utilização de camadas para a especificação de serviços foi também identificada como um aspecto importante, além de ser uma decorrência natural da forma com que protocolos e serviços de comunicação existentes são comumente modelados.

O modelo baseia-se em um estilo arquitetural [54, 90] da forma *pipes e filtros* [13, 89]. Pipes correspondem à representação explícita da conexão entre componentes (filtros) que, por sua vez, representam os elementos de processamento da implementação do serviço. Pipes podem ser ponto-a-ponto ou ponto-a-multiponto. Aos pipes serão associadas as especificações de QoS que serão utilizadas para definir e regular a forma de comunicação entre os filtros que os utilizam. Pipes que são associados a especificações de QoS passam a ser denominados *MediaPipes* [25, 24, 23].

Pipes também fornecerão a abstração em relação aos mecanismos de implementação dos serviços utilizados para honrar os compromissos de QoS, que irão depender do tipo de ambiente em que a comunicação estará acontecendo e das próprias características de qualidade requisitadas. A estruturação em camadas será utilizada como base para os mecanismos de abstração e reificação dos serviços, que permitirão, respectivamente, a programação em alto nível e as adaptações do serviço. A Figura 1.1 ilustra a abstração de um MediaPipe (em um nível *N*) com sua estrutura (no nível *N-1*) revelada.

[3] Na língua portuguesa, a palavra "instância" não apresenta o significado que tem sido a ela frequentemente atribuído, referindo-se a "um indivíduo de uma população ou classe" ou "um exemplo tomado de dentro de um conjunto de possibilidades". O verbo instanciar (e, consequentemente, o termo instanciação) nem mesmo existe. Porém, dada freqüência com que esse termo tem aparecido, principalmente em trabalhos na área de orientação a objetos, o autor julgou por bem privilegiar o entendimento ao invés de tentar utilizar outros termos que poderiam até prejudicar leitores já acostumados com esse tipo de jargão.

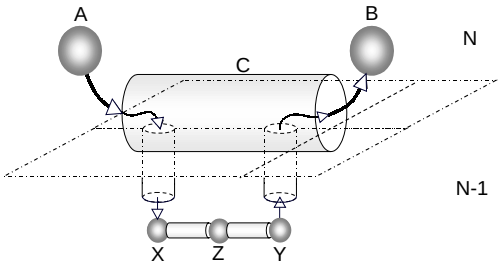


Figura 1.1: Abstração e reificação de MediaPipes: utilizando a recursividade do modelo.

As esferas *A* e *B* correspondem aos componentes (filtros) usuários (de nível *N*) de um serviço de comunicação provido através do MediaPipe *C*. Ao revelar a estrutura de *C*, obtém-se a visão do nível inferior (*N-1*) que, nesse exemplo, contém dois MediaPipes verticais e dois horizontais, além dos componentes *X*, *Y* e *Z*. Todos esses elementos (componentes e MediaPipes) são conjuntamente responsáveis pelo comportamento da abstração do MediaPipe do nível *N*. Cada um dos MediaPipes do nível *N-1* pode ser então, recursivamente, estruturado de maneira similar ao MediaPipe *C*. Os mecanismos de adaptação dos serviços corresponderão aos mecanismos de criação, destruição e modificação (através da reificação e modificação dos elementos internos) de pipes e componentes.

Existe um número considerável de trabalhos em andamento cujo enfoque está nos mecanismos de adaptação e programação de serviços nas áreas de telecomunicações e redes de computadores. Em geral, eles recaem em algumas categorias como as de redes inteligentes (Intelligent Networks — IN) [71, 7], redes programáveis [80, 78, 79, 20, 137] ou redes ativas [130, 129, 136, 135, 106]. Utilizando o modelo proposto, as diferenças entre essas categorias serão comparadas em termos dos elementos alvo das adaptações e da forma com que elas podem ser efetuadas. Essa comparação será feita sob a luz das idéias de *reflexividade* e de *níveis de meta serviços* [75, 8, 10, 30, 85, 127, 108]. Sob essa perspectiva, mecanismos aparentemente incompatíveis ou sem relação poderão se mostrar, por vezes, complementares. Outros mecanismos comuns em redes de comunicação (como sinalização e gerência) também serão analisados sob a luz desse mesmo modelo, fornecendo uma visão que, apesar de compatível com a tradicional, apresenta-se um pouco diferente daquela normalmente utilizada na literatura.

1.4 Organização da Tese

No Capítulo 2, o conceito de serviço de comunicação e seu ciclo de vida serão abordados. Em especial, o significado do tratamento de informações em arquiteturas de comunicação e o impacto causado nos modelos de serviço serão discutidos. Ao longo dessa discussão, alguns trabalhos relacionados à modelagem e à adaptabilidade desses serviços serão comentados.

O Capítulo 3 apresenta o modelo que corresponde à parte central desta tese. Define-se o conceito de um *ambiente de oferecimento de serviços* como o de um conjunto de usuários e os respectivos provedores, estruturados segundo um conjunto de regras básicas. A estrutura interna de provedores é fornecida baseada na definição dois tipos de provedores: os *provedores de infra-estrutura* e os *provedores de acesso*. Componentes compostos também são definidos com base na idéia de *espaços* e *subespaços*. Sobrepondo-se a esse conjunto de regras de estruturação, serviços podem ser construídos, correspondendo a *topologias virtuais* que representam as associações entre componentes. A idéia de adaptação de serviços é definida com base na

criação, modificação ou destruição dessas topologias. As *torres de serviços e meta serviços* são definidas de forma a caracterizar a atuação dos diversos mecanismos de adaptação sobre essa abstração. A adaptabilidade do serviço é finalmente analisada tomando como base os elementos alvo da manipulação, seus responsáveis e o momento dentro do ciclo de vida em que essas manipulações podem ocorrer.

Os elementos básicos e as mencionadas restrições apresentadas no Capítulo 3 formam uma espécie de "linguagem básica" para construção de serviços, a partir da qual estruturas pré-montadas podem ser definidas para modelar necessidades comuns em serviços de comunicação (como a provisão da QoS e a comunicação de grupo). Essas estruturas pré-montadas serão, ao mesmo tempo, usuárias da linguagem básica e geradoras de linguagens mais específicas. Sua definição corresponderá à definição de frameworks, dois dos quais são apresentados nos Capítulos 4 e 5.

O framework apresentado no Capítulo 4 modela, de uma forma genérica, o funcionamento dos mecanismos de provisão de QoS em sistemas de processamento e comunicação. Trata-se de uma revisão e extensão dos conceitos apresentados em [57] e [56]. Os principais pontos que mereceram considerável atenção, e que diferiram da abordagem anteriormente apresentada nesses trabalhos, dizem respeito a dois aspectos: (i) a representação dos frameworks tornou-se consideravelmente mais simples com a adoção de uma notação que provilegia o nível conceitual e não apresenta mais os detalhes da implementação de patterns específicos dedicados a adaptabilidade. Essa estratégia segue as idéias apresentadas por Fontoura [49, 48], embora a notação utilizada não seja exatamente a mesma; (ii) os diversos mecanismos são analizados sob a luz do modelo definido no Capítulo 3, estabelecendo quais desses mecanismos podem ser considerados meta serviços, sobre quais serviços eles atuam e de que forma o fazem.

O Capítulo 5 é, em vários aspectos, semelhante ao Capítulo 4. Trata-se de uma revisão do framework para tratamento de comunicação de grupo apresentado em [112, 111] nas mesmas bases de generalidade que o anterior.

O Capítulo 6 é dedicado a uma avaliação geral deste trabalho. Entre outras coisas, os sucessos, fracassos e dificuldades serão apontados. Vários pontos neste trabalho merecem uma revisão detalhada. Alguns deles podem se beneficiar de desenvolvimentos oriundos de outras área de ciência da computação como as de aspectos formais e dos novos trabalhos que vêm sendo constantemente desenvolvidos na área de engenharia de software, principalmente sobre arquiteturas de software e desenvolvimento de frameworks. Em vários pontos, o modelo do Capítulo 3 pode ser considerado incompleto ou inadequado. Alguns deles, como já mencionado na Seção 1.2, são propositadamente deixados em aberto como forma de generalidade do modelo. Outros, porém, foram resultado das dificuldades encontradas para sua definição, merecendo ser destacados como pontos em que esse modelo deve ser revisado. Por fim, procura-se delinear como este trabalho pode ser continuado, ressaltando, principalmente, a necessidade de desenvolvimento de casos uso que exercitem os conceitos apresentados e ajudem no processo de refinamento contínuo ao qual este modelo deverá ser submetido.

1.5 Sumário

A sofisticação das novas aplicações multimídia distribuídas que têm sido vislumbradas sugere o alto grau de complexidade exigido dos serviços de comunicação e as dificuldades associados ao projeto dessas aplicações. Grande parte das dificuldades consiste em oferecer serviços capazes de manter as características necessárias a toda a diversidade de tipos de fluxo.

O grande ideal das operadoras de serviços de comunicação é projetar uma infra-estrutura que permita tanto a integração da mais variada gama de diferentes serviços em um único sistema de comunicação, como a criação de um ambiente no qual esses mesmos serviços possam ser rápida e facilmente criados, alterados e continuamente adaptados às novas necessidades e condições. Porém, a variedade de requisitos e a rapidez dos avanços tecnológicos têm sido grandes empecilhos, especialmente quando a estratégia adotada consiste na procura de uma arquitetura única, genérica o suficiente para acomodar os diversos serviços, utilizando-se como base apenas a configuração de algumas opções pré-definidas. Uma alternativa recentemente considerada consiste na definição de modelos em que serviços podem ser dinamicamente programados ou adaptados ao longo de seu ciclo de vida.

O presente trabalho visa propor um modelo que forneça mecanismos (ou abstrações) para a representação e programação de aspectos de QoS e comunicação de grupo (multicasting) em serviços de comunicação multimídia. Tal modelo deverá servir como base para a análise e comparação de mecanismos de adaptação e programação de serviços em redes de comunicação. A intenção do modelo será a de estabelecer uma base conceitual que tornará possível o desenvolvimento de outros modelos mais específicos. Conseqüentemente, o modelo será propositadamente incompleto demais para ser diretamente implementado, refletindo o fato de que o objetivo foi sempre o da generalidade, podendo, nesse sentido, ser considerado um “meta modelo”.

Uma das principais características do modelo é seu caráter recursivo, baseado em um estilo arquitetural da forma *pipes e filtros*. Pipes correspondem à representação explicita da conexão entre componentes (filtros) que, por sua vez, representam os elementos de processamento da implementação de um serviço. Aos pipes serão associadas as especificações de QoS, passando então a ser denominados *MediaPipes*.

MediaPipes também fornecerão a abstração em relação aos mecanismos de implementação dos serviços utilizados para honrar os compromissos de QoS, que irão depender do tipo de ambiente em que a comunicação estará acontecendo e das próprias características de qualidade requisitadas.

Este capítulo foi dedicado à apresentação das motivações, dos objetivos da tese e da estratégia geral adotada. A estrutura geral da tese foi também apresentada, fornecendo uma visão geral dos passos que serão seguidos nos próximos capítulos.

Capítulo 2

Serviços de Comunicação: Conceitos e Ciclo de Vida

A ENGENHARIA de Serviços de Telecomunicações [139, 138] é uma nova disciplina que tem como principal objetivo tratar dos aspectos de especificação, projeto, implementação, gerenciamento e validação de serviços de telecomunicações e sua implantação sobre arquiteturas de rede existentes e futuras. Um dos seus maiores desafios tem sido o de lidar com os problemas, citados no Capítulo 1, de integração dos diferentes serviços e da necessidade de sua rápida criação e contínua adaptação. Antes de passar diretamente aos aspectos do modelo que constituem a parte central desta tese, o presente capítulo será destinado a estabelecer uma visão mais precisa sobre o conceito de "serviço de comunicação". A importância e o significado do tratamento de informações em arquiteturas de comunicação e o impacto causado nos modelos de serviço serão discutidos e, ao longo dessa discussão, alguns trabalhos relacionados à modelagem e à adaptabilidade desses serviços serão comentados. As diversas fases que, em geral, estão presentes no ciclo de vida de um serviço serão apresentadas e o impacto da adaptabilidade sobre esse ciclo será estudado. Dessa análise, será possível obter subsídios para, posteriormente, classificar os diversos mecanismos de adaptação e programação presentes em vários trabalhos encontrados na literatura. O relacionamento da disciplina de engenharia de serviços de telecomunicações com a engenharia de software será apontado, enfatizando-se o aspecto de como a primeira pode fazer uso de técnicas de orientação a objetos, frameworks e estilos arquiteturais.

2.1 Serviços de Comunicação

O termo "serviço de comunicação" tem sido utilizado para representar vários conceitos em diferentes contextos. Em modelos hierárquicos como o OSI-RM [67], por exemplo, serviços são vistos como as "facilidades" providas por uma camada à sua camada superior adjacente, em geral descritos por primitivas e seus relacionamentos. As variações de serviços que podem ser obtidas restringem-se a alguns tipos predefinidos (como serviços orientados ou não à conexão) e à utilização de alguns parâmetros. A tônica desses modelos não é a de fornecer serviços sofisticados ou formas de acesso elaboradas. A interoperabilidade entre sistemas era a maior preocupação quando tais modelos foram inicialmente concebidos, e seu maior propósito foi o de alavancar a produção de protocolos padronizados.

Mas o conceito de um serviço de comunicação tem, gradativamente, assumido novas di-

mensões. Ao contrário de uma visão puramente tecnológica, operadoras têm passado a encarar serviços sob um ponto de vista também comercial, relacionado-o ao fornecimento de "pacotes de facilidades" na fronteira em que os consumidores "enxergam" o serviço de comunicação. Nesse sentido, um serviço de comunicação passa a ser entendido como uma forma de empacotar e comercializar um conjunto de facilidades ou recursos de comunicação que pode ser separadamente contratado por consumidores.[1]

A integração de serviços sugerida pelos modelos das Redes Digitais de Serviços Integrados de Faixa Estreita e de Faixa Larga (RDSI-FE [72] e RDSI-FL [70]) foi também uma das maiores forças de mudança da visão que se tinha de serviços de comunicação. Além de listar e classificar um grande número de novos serviços, esses modelos destacaram a necessidade da separação da infra-estrutura de comunicação fornecida pelos níveis inferiores da arquitetura (que são parte da Rede Digital Integrada — RDI) dos serviços providos diretamente aos consumidores. Esses serviços de nível mais alto passariam então a ser os alvos de novos modelos de serviço, podendo ser fornecidos, competitivamente, por várias empresas. Tais modelos de serviço seriam desenvolvidos tomando por base a existência de uma arquitetura na qual a infra-estrutura básica de comunicação (a RDI) é vista como uma plataforma distribuída sobre a qual serviços de mais alto nível podem ser instalados e operados [6].

Com as pesquisas nas áreas de redes programáveis e redes ativas, a visão da separação entre serviços de nível inferior e superior passa a sofrer novas modificações. Em primeiro lugar, novas formas mais dinâmicas com que os serviços são criados começaram a ser propostas. Além disso, a própria infra-estrutura de comunicação passou a ser vista como alvo de possíveis modificações ou adições de novos serviços. Em outras palavras, nessas novas propostas, a infra-estrutura básica de comunicação tem se tornado uma plataforma distribuída de uso mais geral, sobre a qual serviços de qualquer nível podem ser adicionados.

2.1.1 Uma Perspectiva Histórica do Tratamento de Informações em Arquiteturas de Comunicação

A evolução da infra-estrutura de redes de comunicação pode ser analisada sob o ponto de vista do tipo de informação que é processada pelos seus equipamentos e da facilidade com que essas informações podem ser inseridas, removidas ou alteradas. Nessa perspectiva, pode-se entender o funcionamento de um sistema de comunicação sob o prisma de uma *base de informações distribuída* que regula e auxilia no funcionamento de todo o sistema.

O primeiro tipo de informação amplamente utilizado e manipulado nessa base de informações foi o de *conectividade*. O sistema telefônico, que foi certamente o primeiro a experimentar esse tipo de modificação, passou da era do chaveamento manual para a era das chaves eletro-mecânicas, seguidas dos comutadores eletrônicos até, por fim, atingir o estágio das centrais computadorizadas. Do ponto de vista da informação, essas mudanças podem ser encaradas como a transição de um tipo de informação "pesada" (na qual as alterações são feitas de uma forma manual ou "física") para um tipo de um informação mais "leve" (ou "lógica", na qual a conectividade é representada, alterada ou controlada de maneira fácil e automática).

A introdução de sistemas computacionais nos ambientes de redes de comunicação , além de proporcionar a modificação de algumas informações, tornando-as mais leves, também permitiu

[1] A distinção entre os termos "serviço" e "facilidade" é bastante tênue já que facilidades podem também ser outros serviços [132]. Como algumas arquiteturas (como a IN [71], p. ex.) não são muito enfáticas nessa distinção, no presente trabalho ambos os termos serão utilizados indistintamente, com o mesmo sentido.

a introdução de novos tipos de informação na base. O exemplo clássico advém da introdução das bases de informação de gerência. O crescimento da complexidade das redes de comunicação fez com que as operadoras procurassem formas mais eficazes de gerenciamento e ferramentas que auxiliassem nas suas tarefas corriqueiras de operação. Arquiteturas de gerenciamento como ITU-T TMN (Telecommunications Management Architecture) [73] surgiram propondo a criação de novos *modelos de informação* que deveriam fornecer uma representação dos recursos da rede para os propósitos de monitoramento, controle e gerenciamento. Essas informações passariam então a se tornar disponíveis na base de informações distribuída das redes de comunicação.

A base de informações distribuída passaria a conter então um conjunto de informações de conectividade, configuração, estado dos dispositivos e recursos, etc. É importante notar que a distinção entre informações pesadas e leves continuava a existir, a exemplo das informações sobre quais equipamentos estão ligados a uma determinada porta de um comutador ou da topologia de uma rede, que são informações que só são alteradas quando mudanças físicas são efetuadas, sendo, por assim dizer, informações pesadas. Outras informações de configuração que podem ser dinamicamente alteradas pelos operadores, como, por exemplo, o número máximo de conexões a serem aceitas em um determinado enlace, podem ser consideradas leves.

A introdução de poder computacional nas redes de comunicação viria ainda a proporcionar novas modificações. A integração de serviços proposta nas RDSI-FE e RDSI-FL destacava a necessidade da separação da infra-estrutura de comunicação fornecida pelos níveis inferiores da arquitetura dos serviços de nível mais alto, que passariam então a ser os alvos dos novos modelos de serviço. Paralelamente, operadoras de serviços em conjunto com os órgãos de padronização internacional definiriam as *redes inteligentes* (Intelligent Networks — INs) [71], que surgem criando um ambiente para a rápida introdução de novos serviços nas redes de comunicação (visando principalmente as redes de telefonia).

Assim como as RDSIs, as INs (e suas precursoras, como os serviços 800) também eram baseadas na divisão entre os serviços de nível inferior e serviços superiores. Serviços providos em níveis mais altos da arquitetura seriam introduzidos com o auxílio de *bases de dados* utilizadas para armazenar programas e informações, cujo acesso seria ativado sempre que números pré-definidos fossem discados pelos usuários. Novamente, a flexibilidade aparece atada ao conceito de se fornecer bases de informação capazes de tornar disponível certos tipos de informação durante a operação, além de, ao mesmo tempo, tornar essas informações mais leves no que concerne sua manipulação ou alteração.

A evolução dos serviços e, por conseguinte, o crescente volume e complexidade das informações contidas na base informação criaram a necessidade de novos modelos e ferramentas para criação e manutenção dos serviços. A tarefa de se construir e manter a base de informação atualizada tem se tornado uma espécie de atividade relacionada à programação ou construção de sistemas distribuídos. Novos esforços como a *TINA (Telecommunications Information Networking Architecture)* [6] surgiram com o propósito de construir um modelo de referência para a construção de serviços de telecomunicações baseado em avanços oriundos da área de programação distribuída e orientada a objetos — como os modelos ODP (Open Distributed Processing) [69] do ITU-T e o CORBA (Common Object Request Broker Architecture) [97] do OMG. Na mesma direção, o *Binding Model* [80] define a base conceitual para criação de serviços de comunicação utilizando o conceito de “binding”. O termo *binding* refere-se à associação de um conjunto de recursos ou dispositivos de rede e equipamentos de usuários a um ou mais protocolos de transporte e a serviços de gerência desses recursos. Para efetuar essa associação,

o modelo define a utilização de uma *base de informações distribuída* denominada *Binding Interface Base (BIB)* sobre a qual os mecanismos de associação deverão operar. Os mecanismos de manipulação das informações nessa base deverão permitir a programação dos serviços, e a plataforma utilizada deverá ser um ambiente distribuído e orientado a objetos como aquele definido pelo CORBA.

Todo esse conjunto de esforços para transformar e encarar sistemas de comunicação como palataformas de desenvolvimento e programação de serviços deu origem ao termo "redes programáveis" [79, 17]. A principal característica dos trabalhos que se enquadram nessa linha é o de propor o fornecimento de um modelo que, acompanhado de um conjunto mínimo de interfaces simples implementadas nos equipamentos que compõem a rede (ou a p*lataforma distribuída de serviços*), permitem a programação distribuída dos serviços. Essas interfaces oferecem, entre outras coisas, o acesso aos recursos internos de cada equipamento e a modelagem das bases de informação necessárias para a construções dos serviços.

Uma estratégia que pode ser considerada ainda mais agressiva foi adotada pelos grupos que propõem a construção de *redes ativas* [130]. A idéia básica pode ser considerada como a mesma das redes programáveis, porém, levada às últimas conseqüências. Nesses trabalhos, o próprio código da implementação dos protocolos internos da rede passa a se tornar uma informação disponível na base de informações, podendo ser alterada, inserida e manipulada tanto por usuários como pela operadora durante a fase de operação. A rede como um todo passa a ser vista como uma espécie de "máquina computacional de uiso geral". Dessa forma, redes ativas permitirão que os equipamentos da rede, vistos como uma plataforma distribuída, executem procedimentos injetados em seus nós ao longo da operação do serviço. No casos mais extremos, pacotes convencionais serão substituídos por fragmentos de código (e informações), que serão executados a cada nó de rede do caminho em que passarem. Consequentemente, do ponto de vista de informação, entidades de protocolo podem passar a ser informações extremamente leves, colocadas, retiradas ou alteradas na base de informações a cada fragmento que trafega pela rede.

Em resumo, a introdução de adaptabilidade em redes de comunicação pode ser associada aos avanços na capacidade de processamento de informações dos equipamentos utilizados nesses sistemas. Não apenas novos tipos de informação foram introduzidos, como informações tornaram-se mais leves. Até mesmo informações como a topologia da rede que sempre foram tradicionalmente pesadas, começaram a se tornar leves nas denominadas *redes virtuais* [16, 18]. Em outras arquiteturas a própria implementação das pilhas de protocolo e sua configuração tem sido transformada em um tipo de informação adaptável. É claro que, quanto mais informações pesadas são tornadas leves, maior é a gama de serviços que pode ser dinamicamente modificada ou adicionada.

Mas essa flexibilidade, conseqüência da transformação e introdução de informações leves e o dos respectivos mecanismos de alteração ou criação, não é obtida sem um preço. Quanto mais flexível é o modelo, mais complexa é a tarefa de manter a base de informações, principalmente quando se consideram aspectos como segurança. A diversidade de mecanismos também é um ponto de complexidade a se considerar, fazendo surgir perguntas sobre a possibilidade de coexistência dos diversos modelos e sua compatibilidade.

Um dos propósitos desta tese é tornar mais clara a forma de atuação dos mecanismos de adaptação de forma a determinar em que casos eles podem ser utilizados conjuntamente, isto é, em que casos eles são incompatíveis ou complementares. Com esse propósito, a Seção 2.2 será dedicada a uma análise do ciclo de vida dos serviços e de que forma a capacidade de adaptação, em suas diferentes formas, atuam.

2.2 Ciclo de Vida de Serviços de Comunicação

O *ciclo de vida de um serviço (Service Life Cycle — SLC)* é um modelo que inclui todas as fases de um serviço, desde de sua concepção inicial, passando por todo o processo de implantação e utilização até o momento em que ele é retirado do sistema. Um SLC define, para cada fase, o conjunto de suas atividades e de que forma elas devem ser realizadas [44]. A especificação de um ciclo de vida também pode ser utilizada como uma especificação de requisitos de alto nível para construção de ferramentas automatizadas ou de auxílio à criação de serviços, plataformas de execução, ferramentas de instalação, etc. Nesta seção, as diversas fases que, em geral, estão presentes no ciclo de vida de um serviço serão apresentadas e o impacto da adaptabilidade sobre esse ciclo será estudado. Dessa análise, será possível obter subsídios para, posteriormente, classificar os diversos mecanismos de adaptação e programação de serviços.

2.2.1 Ciclo de Vida Tradicional

Qualquer SLC conterá, invariavelmente, aspectos técnicos (relacionados ao projeto, instalação, operação, manutenção, acesso de usuários, segurança dos serviços, etc.) e aspectos administrativos (relacionados ao estabelecimento e possível rompimento de um contrato de utilização dos serviços entre o consumidor e a operadora do serviço). Essa diferenciação se refletirá diretamente na definição dos "personagens" encontrados em um ambiente de oferecimento de serviços e suas responsabilidades em relação às fases do ciclo. Um cenário genérico de oferecimento de serviços, baseado em uma simplificação do modelo apresentado em [44], encontra-se ilustrado na Figura 2.1.

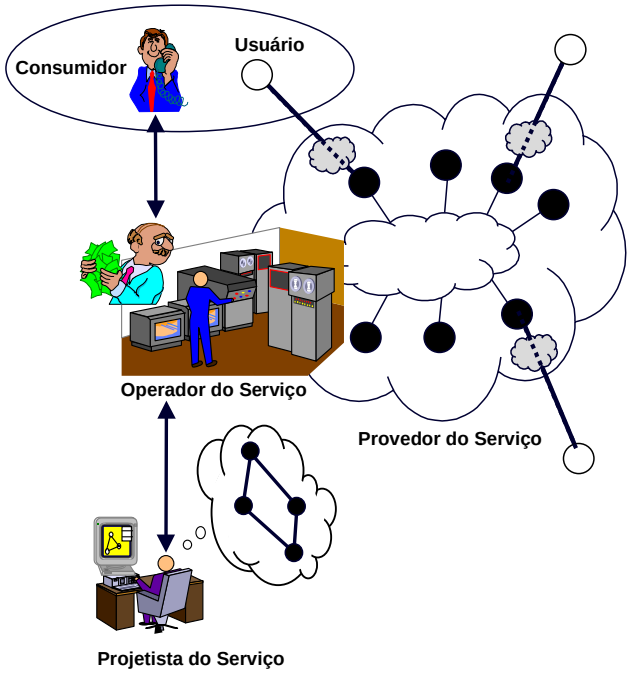


Figura 2.1: Cenário de oferecimento de serviços.

Usuários e *provedor* de serviço são entidades relacionadas aos aspectos técnicos do serviço, correspondendo aos conceitos usuais utilizados em arquiteturas como o OSI-RM [67]. Mais

voltados a aspectos administrativos, *consumidores* representam as entidades ou organizações que são responsáveis pela *contratação* (ou *assinatura*) do serviço junto às *operadoras (*também conhecidas como *prestadoras)* de serviços.[2] Essas, por sua vez, são as responsáveis por toda a administração do serviço, incluindo sua instalação, operação e manutenção, sendo, portanto, as representantes do serviço perante consumidores. *Projetista* é uma designação genérica atribuída à entidade ou organização responsável pela construção do serviço (que pode incluir a análise, o projeto, a implementação e o teste de componentes desse serviço). As fases de um SLC tradicional e as responsabilidades de projetistas, operadores, consumidores, usuários e provedores podem ser visualizadas na Figura 2.2.

Fases		Responsáveis
Construção	Análise	Projetista
	Projeto	
	Implementação	
	Testes	
Implantação	Instalação	Operador
	Ativação	
Operação	Assinatura	Consumidor e Operador
	Acesso	Usuário e Provedor
	Interação	
	Saída	
	Cancelamento	Consumidor e Operador
Retirada	Desativação	Operador
	Remoção	

Figura 2.2: Exemplo de ciclo de vida de serviços.

É fácil distinguir os responsáveis em cada uma dessa fases em um ambiente tradicional já que as funções são facilmente visualizadas. Projetistas limitam-se às fases de construção; operadores trabalham na implantação e retirada de serviços, além de atenderem a consumidores (quando o serviço já está disponível para operação) de forma a estabelecer ou cancelar contratos de fornecimento de serviços. A utilização do serviço propriamente dita é feita por meio da interação entre usuários e provedor. As fases de acesso, interação e saída podem corresponder, por exemplo, à sinalização para estabelecimento de uma conexão, troca de informação e sinalização para rompimento da conexão, respectivamente.

2.2.2 O Impacto da Adaptabilidade

Em serviços de comunicação tradicionais, consumidores permaneciam completamente alheios às tarefas associadas a projetistas e operadoras. Mas, com a introdução de redes programáveis [80] e redes ativas [130], as responsabilidades começam a se alterar. A separação das fases do SLC torna-se menos óbvia dado que a construção e alteração dos serviços pode ser feita de

[2] Em alguns trabalhos (como [44] p. ex.), o termo “provedor” é utilizado tanto para se referir ao que se convencionou aqui chamar de “operador” (ou “prestador”), como para o provedor em si, sendo a distinção feita pelo contexto em que o termo é uitlizado. Para evitar confusões, preferiu-se utilizar termos distintos. O mesmo comentário é válido para os termos “usuário” e “consumidor”.

forma dinâmica. Uma das características desses ambientes é justamente o fato de que consumidores assumem, ainda que parcialmente, as atividades de operadores e projetistas. É claro que não serão os consumidores finais das aplicações que deverão assumir essas responsabilidades, mas sim os desenvolvedores, que serão os responsáveis por construir os aplicações usuárias do sistema de comunicação. Nesse sentido, esses desenvolvedores são também parte da entidade a que convencionou-se chamar de consumidor.

Os consumidores dos serviços passam então, de certa forma, a participantes do processo de evolução da infra-estrutura dos serviços de comunicação. A complexidade desse processo poderá se tornar um empecilho se ele não estiver apoiado em um modelo adequado, que integre as duas visões (programação de aplicações e implementação de serviços) [24]. Vários trabalhos têm enfocado a forma com que serviços de comunicação são desenvolvidos, operados e mantidos. A estratégia adotada no presente trabalho inclui, como ponto de partida, a utilização de *frameworks* para a modelagem de serviços de comunicação [57, 112], como anteriormente mencionado. O planejamento das adaptações através da construção de frameworks modifica o ciclo de vida dos serviços tanto nas fases de construção como de operação, permitindo formas de adaptação estáticas e dinâmicas.

2.3 Frameworks: Planejando Adaptações

Não é de se surpreender que a tecnologia de orientação a objetos (OO) tenha sido uma das mais utilizadas nos trabalhos que tratam da modelagem de serviços de comunicação adaptáveis. Os problemas de reuso de software e adaptabilidade de serviços são de fato problemas similares. Em se tratando especificamente de adaptabilidade, o estudo de *frameworks* [104, 105] tem merecido a atenção de um número cada vez maior de pesquisadores.

Um *framework*, como já mencionado no Capítulo 1, difere de uma aplicação pelo fato de que algumas de suas partes (denominadas *pontos de flexibilização* ou *hot-spots*) são propositadamente deixadas incompletas ou marcadas como "sujeitas a variações". Através do respectivo preenchimento dessas "lacunas" ou de modificações nos pontos marcados, pode-se obter a adaptabilidade do serviço. O momento dentro do ciclo de vida de um serviço em que hot-spots são completados determina o seu grau de dinamismo. Os hot-spots complementados nas fases de construção dão origem a serviços específicos que, por sua vez, ainda podem apresentar hot-spots a serem complementados, modificados ou estendidos em tempo de operação.

Uma definição mais concreta de um framework, apresentada por Fontoura et al. [49, 48], afirma que um framework é uma aplicação que contém uma ou mais funções variáveis ou funções de extensão. Uma *função variável* é uma função cujo código (implementação) não é fornecido ou, alternativamente, quando fornecido, pode ser trocado no momento da instanciação ou execução de uma aplicação específica (derivada do framework). Por outro lado, ao contrário das funções variáveis, que são imprescindíveis dentro da lógica da aplicação, uma *função de extensão* é uma porção de código que pode ser opcionalmente adicionada a uma aplicação.

Na visão de Fontoura, ao associar-se hot-spots a funções, cobre-se boa parte das situações de adaptabilidade. Nessa visão, hot-spots estão estritamente associados a operações ou funções, ou ainda, no caso de ambientes OO, a métodos.

O usuário de um framework é um projetista de uma aplicação, que completará os hot-spots de forma a gerar essa aplicação segundo requisitos específicos. Assim, um framework pode ser considerado como um modelo de uma família de aplicações (no caso específico deste trabalho,

de uma família de serviços). O ato de produzir uma aplicação específica a partir de um framework durante as fases de construção é denominado *instanciação do framework.* Hot-spots podem, como já mencionado, também ser modificados ou estendidos em tempo de operação, através de procedimentos aqui denominados de *configuração* ou *adaptação*. A pura instanciação de um framework pode ser relacionada às formas mais tradicionais de reuso de código e projeto, enquanto que adaptações podem ser associadas às formas mais dinâmicas de evolução de serviços.

2.3.1 Questões de Projeto de Frameworks

Um dos obstáculos à utilização de frameworks tem sido a falta de uma notação adequada para a representação explícita de hot-spots. Durante a construção de um framework, projetistas deverão estar atentos à especificação de uma série de aspectos tais como: (i) de que forma um projetista de uma aplicação específica irá reconhecer os pontos que devem ser completados ou estendidos; (ii) qual deverá ser o processo de instanciação ou configuração desses pontos e suas restrições; e (iii) em que ponto do ciclo de vida esses hot-spots deverão ser completados.

Em geral, as atuais notações de projeto não oferecem suporte à representação explícita desses aspectos. Além disso, várias notações amplamente utilizadas, como UML [109] (em sua forma básica), não propõem diferenças de notação que refletem o nível de abstração em que um projetista esteja trabalhando [51]. Em outras palavras, a mesma notação deve ser supostamente utilizada na fase inicial do projeto (na qual as questões conceituais estão em primeiro plano), nas fases intermediárias (nas quais o projeto é refinado de forma a apresentar decisões de cunho mais específico) e nas fases finais (nas quais a implementação já se torna o foco principal e o projetista já está, em geral, envolvido com uma série de questões tecnológicas incluindo a linguagem de programação a ser utilizada).

Esses dois fatores (a falta de uma representação explícita e de uma separação dos níveis de abstração) fazem com que a modelagem e representação de frameworks se tornem desnecessariamente mais complexas do que o necessário. A forma que tem sido frequentemente utilizada para a modelagem de frameworks, e que evidencia ambos os problemas, é a utilização de design patterns específicos para a modelagem dos pontos de flexibilização. Os patterns *decorator* e *strategy* [53], por exemplo, são dois patterns especificamente projetados para a solução de determinadas situações de adaptabilidade. A utilização desse tipo de modelagem leva, porém, a diagramas complexos, onde os pontos de flexibilização estão, em geral, encobertos. Apesar disso, supõe-se de um usuário do framework a capacidade reconhecer os patterns inseridos nesse tipo de diagrama e, consequentemente, de distinguir os hot-spots e a forma como eles devem ser instanciados ou configurados.

É fácil observar que a utilização desses patterns está muito mais voltada à forma com que os hot-spots estão sendo implementados do que com a questão mais conceitual de apontar hot-spots e sua função dentro da modelagem da aplicação. Seria, provavelmente, mais interessante que, em um nível de abstração mais alto, o framework trouxesse simplesmente a informação conceitual, o que permitiria a um usuário reconhecer, acompanhar e determinar, de forma mais direta, como esses hot-spots foram implementados nos níveis mais baixos e, consequentemente, como eles devem ser completados.

Felizmente, a UML fornece mecanismos para a introdução de novos elementos na notação a partir dos elementos já existentes, tratando-se, na realidade, de uma linguagem extensível [110]. Essa capacidade motivou estratégias como as apresentadas em [48] e [40], nas quais extensões

são definidas de forma a melhorar a representação dos conceitos ligados a frameworks. Em particular, em [48], extensões são feitas de forma a representar hot-spots associados a funções (e suas características) em um nível conceitual.

2.3.2 Instanciação e Adaptação de Frameworks: Criação de Novos Serviços

Após as fases de modelagem e projeto de um framework, o projetista estará, supostamente, de posse de uma descrição de alto nível, na qual hot-spots estarão diretamente representados, e descrições de nível mais baixo, nas quais algumas decisões de projeto sobre como esses hot-spots serão realizados estarão documentadas. É chegado o momento da implementação, cujo resultado será o código e a documentação final sobre a forma de instanciação e configuração desse framework.

Fontoura defende a utilização de métodos assistidos por ferramentas automatizadas nesse processo, propondo e descrevendo um conjunto de utensílios que deverão auxiliar, de forma conjugada, na geração de código e documentação (pelos projetistas do frameworks) e na instanciação de aplicações pelos seus usuários [48]. O projetista do framework fará uso, em primeiro lugar, de uma *ferramenta de geração de código* que é responsável pelo mapeamento das definições dos hot-spots em construções de uma linguagem de programação. Esse mapeamento é baseado em algumas informações sobre um modelo de implementação fornecido pelo projetista. Adicionalmente, uma *ferramenta de documentação* é utilizada para gerar a descrição de um processo para a instanciação do framework, que representa um diagrama ou "mapa de passos válidos" a ser seguido durante a instanciação. Durante a fase de instanciação, um usuário do framework fará uso de uma *ferramenta de instanciação*,[3] que "executará" o diagrama gerado pela ferramenta de documentação e interagirá com esse usuário guiando-o e solicitando as complementações ou adaptações que se façam necessárias.

Essa descrição fornece um exemplo de um possível processo para as fases de construção de um SLC, que pode ser tomado como um ponto de partida para a revisão do ciclo de vida de serviços adaptáveis. A introdução da fase de instanciação fornece um primeiro ponto de adaptabilidade. Porém, trata-se de uma forma ainda estática, ligada ao reuso de código e projeto do serviço. Apesar de definir mecanismos para a especificação de hot-spots dinâmicos em sua linguagem de projeto, Fontoura deixa para trabalhos futuros a forma com que tais construções e as respectivas ferramentas deverão ser implementadas.

Além disso, a maior parte dos trabalhos sobre frameworks orientados a objetos está completamente baseado em um modelo conceitual de "objeto" como sendo definido por sua interface que, por sua vez, é definida por um conjunto de operações. Essas operações (ou métodos) são ativadas através de mensagens enviadas de um objeto a outro, representando solicitações de execução, parâmetros e respostas. A implementação dos métodos é, em geral, tomada como o alvo principal da definição de hot-spots. Porém, segundo o modelo definido no presente trabalho, o tipo de comunicação envolvido na chamada de operações entre objetos corresponde à apenas um dos tipos de serviços que serão considerados, denominado de *serviço de comunicação implícita*, discutido com maiores detalhes no Capítulo 3.

Na realidade, a forma de cooperação entre objetos pode dar origem a outras opções de comunicação que, dependendo da natureza do problema sendo considerado, podem ser mais convenientes em termos de modelagem. Da mesma forma com que operações podem ser alvos

[3] Fontoura também descreve uma forma alternativa de instanciação baseada na criação de linguagens específicas de domínio, cuja própria sintaxe serve de guia para a instanciação do framework [48].

da definição de hot-spots, os relacionamentos entre componentes também oferecem potencial para a especificação de pontos de flexibilidade. Esse será o assunto da próxima seção.

2.4 Estilos de Arquitetura e de Cooperação

A especificação da cooperação entre componentes dá origem, em geral, a um vocabulário rico, que pode ser aproveitado pelos projetistas de sistemas, e que, apesar de poder ser tão simples quanto chamadas de procedimentos ou variáveis compartilhadas, frequentemente está associado a formas complexas de comunicação e coordenação [54].

Em aplicações baseadas em *pipes e filtros* [13], por exemplo, um sistema é visto como: (i) um conjunto de componentes (filtros) que processam informações e (ii) um conjunto de conexões (pipes) entre esses componentes. Exemplos de aplicações que podem seguir esse tipo de arquitetura incluem compiladores, sistemas de processamento de sinais e, de forma particularmente interessante, arquiteturas de protocolos de comunicação. A característica comum dessas aplicações é a naturalidade com que suas funções são estruturadas através de um conjunto de subatividades independentes, processadas seqüencialmente ou em paralelo de acordo com algum grafo de precedências. Cada filtro é responsável por uma subatividade específica (por exemplo, o analisador léxico de um compilador ou uma entidade de protocolo de uma camada) da aplicação. Filtros não interagem entre si nem tem conhecimento uns dos outros; suas interfaces permitem apenas o recebimento e envio explícito de dados nos pipes de entrada e saída, respectivamente. Pipes podem definir ou restringir os tipos de dados das mensagens que transportam. Sistemas baseados em pipes e filtros podem diminuir o grau de acoplamento entre os componentes e facilitar mecanismos de configuração [89].

Assim como os hot-spot ligados a operações ou métodos, os aspectos de cooperação ou relacionamento entre componentes também devem ser representados de forma conceitual nos níveis mais abstratos da modelagem de um framework. Eles são, em geral, representados por meio das diferentes formas de associação definidas entre classes e objetos — geralmente representadas pelas linhas que interconectam as entidades dos diagramas nas notações mais comuns. Mas o mapeamento entre essas associações em abstrações de nível mais baixo parece não ser sempre muito simples. Grande parte da elegância e simplicidade, que é obtida da simples especificação de linhas, setas e anotações, pode ser perdida quando torna-se necessário especificar esses relacionamentos por intermédio de patterns, por exemplo. Em um projeto no qual relacionamentos devam ser refinados em cooperações do tipo pipes e filtros, diagramas desnecessariamente complicados serão gerados se uma notação mais concisa não for adotada. Monroe et al. [90] discute essa dificuldade defendendo a utilização de um conceito denominado de *estilo.*

O conceito de estilo pode ser atribuído a uma generalização e aprofundamento da idéia de *arquitetura de software,* originados, principalmente, nos trabalhos realizados na Escola de Ciência da Computação da Universidade de Carnegie Mellon [90, 2, 54, 55]. A noção de arquitetura está baseada na idéia de fornecer uma estrutura geral dos elementos que compõem um sistema e estabelecer os relacionamentos entre esses elementos. A idéia de *estilo* surge como uma generalização desse conceito, no sentido de que estruturações comuns, que se repetem em vários sistemas, podem ser desenvolvidas e utilizadas como base para definir todo um vocabulário, conjunto de regras, propriedades, etc. que são inerentes a essa estrutura e poderão facilitar o desenvolvimento de sistemas que dela se utilizem. Um exemplo de estilo, que pode ser chamado de *estilo arquitetural,* é o das arquiteturas em camadas. Ao descrever um sistema como sendo em camadas (ou hierárquico), uma série de considerações sobre estrutura, serviços e interfaces

já são automáticamente inferidas pelos projetistas, que passam a se beneficiar de um vocabulário conhecido sem a necessidade de especificar uma série de detalhes que, provavelmente, seriam imprescindíveis em uma arquitetura arbitrária.

Estilos são, em muitos aspectos, similares a patterns. A principal diferença parece vir do fato de que estilos sugerem o encapsulamento total de sua estrutura interna. Uma vez definido, um estilo provê uma série de recursos, notações (de fato uma linguagem) que pode ser utilizada de forma transparente. Além disso, estilos definem as propriedades da arquitetura ao invés da forma com que ela deve ser implementada. No caso de pipes e filtros, por exemplo, o estilo determina apenas que filtros são interconectados por pipes pelas suas *portas*, que podem ser de entrada ou de saída. Em [90], cinco patterns diferentes são propostos para a realização desse estilo. Nesse sentido, estilos podem ser considerados como uma forma de descrição de mais alto nível e que, no que diz respeito ao aspecto da adaptabilidade, podem sofrer adaptações em sua implementação.

Monroe et al. [90] cita vários exemplos de estilos como os de sistemas cliente-servidor, pipelines, pipes e filtros e RTP/C. Nenhuma forma de classificação é dada em relação esses estilos, sendo todos considerados estilos arquiteturais. No entanto, é fácil observar que alguns desses estilos determinam aspectos completamente diferentes de outros. O exemplo mais óbvio surge ao se comparar os estilos de camadas com o de pipes e filtros. Torna-se claro que eles definem aspectos completamente ortogonais quando se percebe que é perfeitamente viável (e útil) a idéia de se ter uma arquitetura na qual as entidades de cada camada correspondem a filtros que, por sua vez, se comunicam com as entidades das camadas adjacentes através de pipes.

O que parece estar encoberto é o fato de que alguns estilos privilegiam o aspecto da maneira com que a divisão em partes ou subsistemas é efetuada; já outros estilos enfocam a forma de cooperação entre as partes. No presente trabalho, essas duas categoriais de estilos serão denominadas, respectivamente, de *estilos arquiteturais* e *estilos de cooperação*. Estilos de cooperação fornecerão a base para a definição de aspectos específicos relacionados à comunicação entre componentes de uma arquitetura.

2.5 Conclusões

A importância e o significado do tratamento de informações em arquiteturas de comunicação froram apresentados e a evolução da infra-estrutura de redes de comunicação pôde ser analisada sob o ponto de vista do tipo de informação que é processada pelos seus equipamentos e da facilidade com que essas informações podem ser inseridas, removidas ou alteradas. Nessa perspectiva, observou-se o funcionamento de um sistema de comunicação sob o prisma de uma *base de informações distribuída* que regula e auxilia no funcionamento de todo o sistema. A tarefa de se construir e manter a base de informação atualizada tem se tornado uma espécie de atividade relacionada à programação ou construção de sistemas distribuídos. O próprio código da implementação dos protocolos tem se tornado disponível na base de informações, podendo ser alterado, inserido e manipulado tanto por usuários como pela operadora durante a fase de operação.

Mas essa flexibilidade, conseqüência da transformação e introdução de informações leves e o dos respectivos mecanismos de alteração ou criação, não é obtida sem um preço. Quanto mais flexível é o modelo, mais complexa é a tarefa de manter a base de informações. A diversidade de mecanismos também é um ponto de complexidade a se considerar, fazendo surgir perguntas sobre a possibilidade de coexistência dos diversos modelos e sua compatibilidade.

O primeiro ponto, levantado no presente capítulo, que poderá ser utilizado como base para a comparação entre os mecanismos de adaptabilidade de serviços, surge da observação sobre a forma com que ele pode se inserir ou modificar o ciclo de vida de um serviço. A separação das fases desse ciclo torna-se menos óbvia dado que a construção e alteração dos serviços pode ser feita de forma dinâmica. Uma das características desses ambientes é justamente o fato de que consumidores assumem, ainda que parcialmente, as atividades de operadores e projetistas. O exato efeito da adaptabilidade sobre essas fases poderá ser compreendido a partir da modelagem da estrutura de um serviço, que será apresentada nos próximos capítulos. Já tornou-se claro, porém que o planejamento das adaptações através da construção de frameworks modifica o ciclo de vida dos serviços tanto nas fases de construção como de operação, permitindo formas de adaptação estáticas e dinâmicas.

Capítulo 3

Meta Modelo para Serviços e Aplicações

A VISÃO do funcionamento de um sistema de comunicação sob o prisma de uma base de informações distribuída sugere que os elementos de um serviço podem ser encarados como peças de um "quebra-cabeças", a ser montado conforme as restrições básicas estabelecidas pelos seus "encaixes". As peças correspondem a unidades de informação, que podem estar disponíveis na base de informações para permitir a adaptabilidade do serviço. Os encaixes correspondem às restrições que permitem a construção de serviços válidos. A partir da definição dessas unidades, as formas de adaptação propostas na literatura poderão ser comparadas com relação aos tipos de elementos manipulados, o momento ou fase em que tais manipulações podem ocorrer, e os responsáveis por tais manipulações. Elementos básicos e as mencionadas restrições formarão uma espécie de "linguagem básica" para construção de serviços, a partir da qual estruturas pré-montadas poderão ser definidas para modelar necessidades comuns em serviços de comunicação (como a provisão da QoS e a comunicação de grupo). Tais definiões corresponderão exatamente às definições de frameworks, dois dos quais (já mencionados) serão assunto dos capítulos subseqüentes.

3.1 Notação

Descrições matemáticas serão por vezes utilizadas na definição de alguns conceitos. A notação é bastante simples, podendo ser resumida pela lista a seguir:

$\subset, \subseteq, \supset, \supseteq, \cup, \cap, \in, \notin, +, -, \emptyset$	Símbolos usuais de teoria de conjuntos
$A \times B$	Produto cartesiano
$\mathbb{P} A$	O conjunto de todos os subconjuntos de A (power set de A, também representado por 2^A)
$\{x \mid \textit{expressão}\}$	O conjunto de elementos x tais que a *expressão* é verdadeira
$\#A$	Número de elementos do conjunto A
$\wedge, \vee, \neg$	Operadores lógicos "e", "ou" e "negação"
$=, \neq, <, >, \leq, \geq$	Conectores usuais em predicados
$\forall, \exists, \exists_1, \neg\exists$	Quantificadores "para todo", "existe pelo menos um", "existe exatamente um" e "não existe"

Além disso, utiliza-se, para a meta linguagem de definição, os seguintes símbolos:

$\Rightarrow, \Leftrightarrow, \equiv$ implicação lógica, equivalência lógica e equivalência sintática.

Utilizar-se-á o conceito usual de *relação* entre dois conjuntos como sendo um subconjunto do conjunto de possíveis pares ordenados tomados sobre eles. Ou seja: $\varphi \in \mathbb{P}(A \times B)$ é uma relação entre A e B. O domínio de φ, representado por $\text{dom}(\varphi)$, é o conjunto de elementos de A que estão relacionados a pelo menos um elemento de B em φ. De maneira análoga, define-se a imagem $\text{Im}(\varphi)$ como o conjunto de elementos de B que estão relacionados a pelo menos um elemento de A em φ. Por último, uma *função* também é definida da maneira usual, ou seja, uma relação em que cada elemento do domínio é relacionado a apenas um elemento na imagem. Uma função f de A em B é representada por $f : A \rightarrow B$.

O propósito único e exclusivo da utilização dessa notação foi o de tornar a apresentação mais precisa e concisa. A interpretação de tais descrições como forma de formalização da semântica do modelo ou sua aplicação direta na construção de quaisquer ferramentas fogem ao escopo desta tese.

3.2 Elementos Básicos do Modelo

Os elementos de modelagem básicos utilizados para compor serviços são: (i) os *usuários*, que correspondem a entidades que utilizam diretamente os serviços, e (ii) os *provedores*, que são os responsáveis pelo funcionamento dos serviços durante sua operação. Usuários e provedores são, em geral, equipamentos e produtos de hardware ou software. Denomina-se de um *ambiente de oferecimento de serviços* a um conjunto de usuários e os respectivos provedores. A Figura 3.1 ilustra um ambiente de oferecimento de serviços com um provedor e quatro usuários.

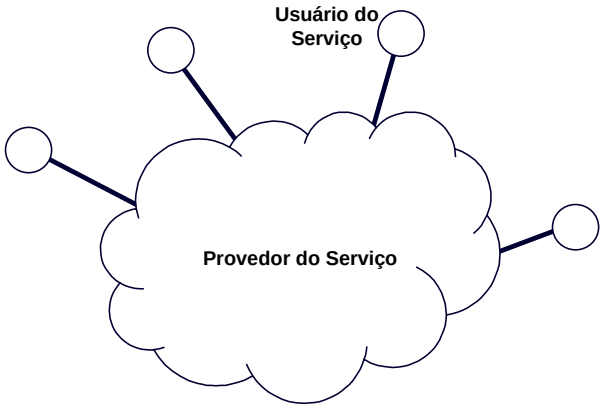


Figura 3.1: Exemplo de ambiente de oferecimento de serviços.

A definição de serviços de comunicação é abstrata no que concerne a dispersão dos usuários. Usuários podem, por exemplo, estar tão próximos quanto objetos implementados em uma mesma linguagem de programação e ativados em um mesmo processo de um sistema operacional de uma máquina isolada, ou tão distantes quanto aplicações localizadas em máquinas de sub-redes distintas.

3.2.1 Provedor de Serviços

A implementação de um provedor é realizada com a utilização de elementos denominados *componentes de implementação do serviço* (ou, simplesmente, *componentes do serviço*). Para

desempenhar suas funções, componentes de serviço comunicam-se utilizando-se dos serviços fornecidos por outros provedores mais primitivos, cada qual denominado *provedor de infra-estrutura* (ou, simplesmente, *infra-estrutura) do serviço*, conforme ilustrado na Figura 3.2.[1] Além disso, todo provedor de serviços deverá fornecer uma forma de acesso para que usuários se comuniquem com componentes de implementação do serviço. Um tal serviço de comunicação é denominado *serviço de acesso*, conforme ilustrado nessa mesma figura.

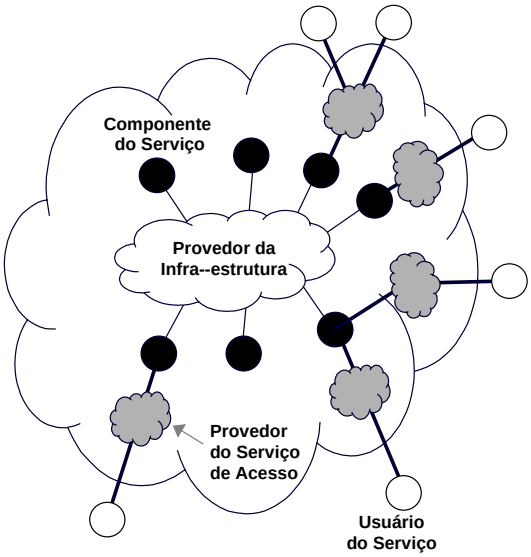


Figura 3.2: Estrutura de um provedor de serviços.

3.2.1.1 Infra-estrutura do Serviço

O serviço oferecido por uma infra-estrutura fornece o suporte necessário aos componentes do serviço para que esses se comuniquem e, coletivamente, implementem as funções que realizarão o comportamento esperado do provedor.

Em sendo um provedor, a infra-estrutura também pode ser recursivamente estruturada, fazendo com que os componentes de serviço que aparecem na Figura 3.2 passem a ser vistos como usuários e, internamente, novos componentes de implementação de serviço, novos provedores de infra-estrutura e de acesso apareçam. Assim, componentes de serviço e a infra-estrutura formam, recursivamente, novos ambientes de oferecimento de serviços.

Esse aninhamento de provedores pode ser, em certos casos, utilizado como um conceito semelhante ao princípio de divisão em camadas definido por arquiteturas hierárquicas como o OSI-RM. Nesse caso, componentes de serviço poderiam ser associados a entidades de protocolo de uma camada N, com a infra-estrutura correspondendo à camada (N-1), e usuários a entidades de protocolo da camada N+1. É importante salientar que tais correspondências são apenas aproximações, haja visto que o provedor de serviço ainda apresentará outras características ausentes nos modelos tradicionais. Além disso, a granularidade dos componentes no presente modelo não é necessariamente igual a de uma entidade de protocolo, podendo ser tanto mais fina como mais grossa. Essas características torna-se-ão mais claras nas próximas seções. Apesar

[1] Por simplicidade, escolheu-se iniciar a apresentação dos elementos do modelo utilizando apenas uma única infra-estrutura. Posteriormente, provedores com mais de uma infra-estrutura serão estudados.

das diferenças, a analogia com camadas de protocolo será algumas vezes utilizada, ao longo deste trabalho, para exemplificar como conceitos tradicionais podem ser abordados.

3.2.1.2 Serviço de Acesso

O provedor do serviço de acesso é um provedor utilizado para fornecer a comunicação entre os diferentes níveis, isto é, entre usuário e componente de serviço. Como qualquer outro provedor, esse também pode ser eventualmente estruturado recursivamente de forma análoga à da Figura 3.2.

Componentes de serviço poderão ser usuários de mais de um dos provedores internos (incluindo infra-estruturas e acessos). Além disso, do ponto de vista do provedor de acesso, alguns de seus usuários serão exatamente os mesmos usuários do serviço externo, enquanto outros serão os componentes de implementação do serviço.

3.2.1.2.1 Relação entre os Serviços de Acesso e a Provisão da QoS

Voltando à associação feita com o modelo OSI na Seção 3.2.1.1, observa-se que o serviço de acesso é o correspondente a um mecanismo de comunicação local entre entidades de duas camadas adjacentes. Porém, esse tipo de comunicação é considerada fora do escopo do OSI-RM, que delega tal tratamento aos sistemas específicos e seus respectivos ambientes locais (Local System Environments — LSEs). Mas, ao se considerar sistemas onde garantias de QoS são necessárias, todas as partes integrantes da comunicação devem ser cuidadosamente projetadas para a construção de um sistema onde os requisitos fim-a-fim são realmente satisfeitos. Por essa razão, vários grupos de trabalho (como os de Illinois [94] e Lancaster [15]) têm proposto abordagens que tratam da arquitetura do sistema como um todo, na solução do problema da provisão de QoS fim-a-fim, sendo coletivamente denominadas de *arquiteturas de QoS* [14]. Essas arquiteturas procuram considerar os vários aspectos e partes do sistema envolvidos na comunicação fim-a-fim, incluindo vários outros ítens, além da própria rede e seus protocolos, como ilustrado na Figura 3.3 [14].

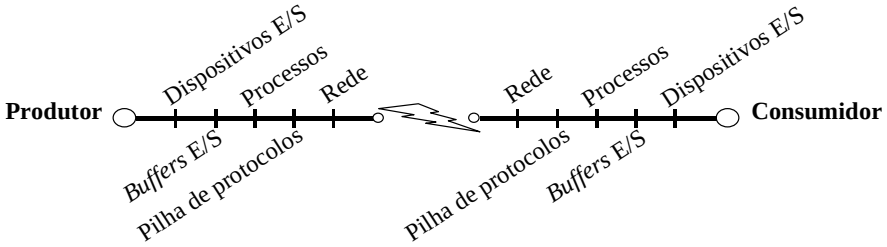


Figura 3.3: As várias partes envolvidas na comunicação fim-a-fim.

Independentemente da abordagem adotada pelas diferentes arquiteturas de QoS, o que elas têm essencialmente em comum é a definição de conjuntos de interfaces que formalizam o conceito de QoS em diferentes partes do sistema, em geral implementadas sob a forma de bibliotecas desenvolvidas sobre esses sistemas. Utilizando conjuntamente as abstrações fornecidas em cada uma dessas partes, pode-se obter a *orquestração* da QoS ao longo de todo o caminho [57, 56].

Uma das grandes dificuldades para a obtenção da orquestração de sistemas utilizando esse tipo de abstração vem da diversidade com que os aspectos de QoS devem ser especificados e, posteriormente, a forma com que eles são tratados nos diferentes sistemas que compõem o ambiente de comunicação e processamento como um todo.

A modelagem de serviços de acesso por meio da abstração de um provedor, abstração idêntica à utilizada na modelagem da comunicação entre quaisquer usuários independente de sua dispersão, busca exatamente propor uma forma de tornar mais homogêneos o tratamento da QoS em ambientes de processamento e de comunicação. O framework de QoS, a ser apresentado no Capítulo 4, confirmará essa homogeneidade através de uma modelagem única, na qual parte dos hot-spots serão responsáveis justamente pelo tratamento das questões específicas de cada ambiente.

3.2.2 Usuários e Componentes de Serviço

A natureza recursiva do modelo faz com que componentes de implementação de um serviço tornem-se os usuários de outros provedores (os de infra-estrutura e de acesso) que, por sua vez, se tornam novamente alvos do mesmo tipo de estruturação. Analogamente, usuários podem ser componentes de implementação de um serviço de mais alto nível. Assim, usuários e componentes de serviço são ambos “componentes” em um sentido mais genérico, que assumem papéis diferentes segundo o foco da observação. O termo “componente” será, de agora em diante, utilizado para denotar o conceito de um elemento que se comunica através de algum provedor, independentemente do papel por ele assumido.

Em serviços de comunicação, a granularidade mais comum associada a um componente é igual a exatamente uma entidade de uma camada protocolo. Porém, duas situações podem ser apontadas como motivadoras para a modelagem de componentes de granularidade diferente. Em primeiro lugar, encontram-se na literatura algumas arquiteturas que relaxam as fronteiras entre camadas de protocolos introduzindo a idéia de arquiteturas baseadas em componentes orientados a tarefas de granularidade mais fina [118, 66]. Tais tarefas podem ser as mais variadas, incluindo o estabelecimento de conexões, tratamento de erros, controle de fluxo, etc. O principal foco desses trabalhos é a possível obtenção de ganho de desempenho através da introdução de paralelismo no processamento dessas tarefas, além da reutilização do projeto e da implementação de componentes comuns em mais de um protocolo. Por outro lado, a possibilidade de agrupar componentes em *componentes compostos* (vide Figura 3.4) abre a possibilidade para a definição de componentes de granularidade mais grossa do que uma única camada. Esse tipo

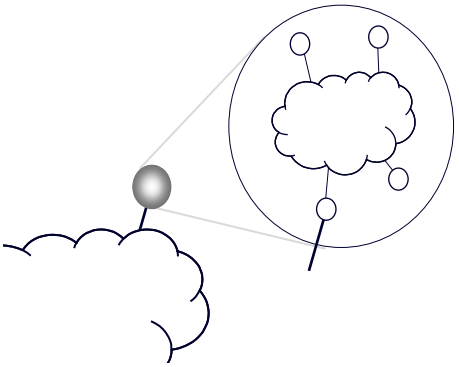


Figura 3.4: Componentes compostos.

de composição pode ser útil tanto para agrupar componentes que representam camadas diferentes (simplificando a configuração de protocolos que usualmente trabalham em conjunto) como também para reunir componentes de granularidade mais fina de forma a compor componentes que representem uma camada inteira, caso isso seja necessário.

Da Figura 3.4 também é possível perceber que a estrutura de um componente composto é, novamente, a de um ambiente de oferecimento de serviços.

3.3 Estruturação dos Elementos Básicos

A partir dos conceitos de modelagem apresentados na seção anterior, a estrutura dos elementos básicos pode ser detalhada. Uma série de regras que definem as formas válidas de estruturação de um ambiente de oferecimento de serviços serão fornecidas. Essas regras dizem respeito, principalmente, à forma de composição ou estruturação interna de provedores e componentes. Ambas serão baseadas na definição genérica do próprio ambiente de oferecimento de serviços.

DEFINIÇÃO 3.1 (AMBIENTE DE OFERECIMENTO DE SERVIÇOS). *Um* ambiente de oferecimento de serviço (AOS) *é definido por uma tupla* $\Gamma = (C, P, \varphi)$ *onde:*

- C *e* P *são conjuntos disjuntos de* componentes *e* provedores*, respectivamente, tais que :*

$$(\#C) + (\#P) \neq 0$$

- $\varphi \in \mathbb{P}(C \times P)$ *é uma relação representando as ligações existentes entre componentes e provedores.*

Um AOS é dito *simples* se ele é formado por apenas um provedor. AOSs são ditos *unidos* se eles tem componentes em comum. A "soma" de AOSs (simples ou não, ligados ou não) dá origem a um novo AOS. O conceito de *soma* de AOSs, juntamente com algumas outras definições úteis, encontra-se na Definição 3.2.

DEFINIÇÃO 3.2.

1. *Um AOS* $\Gamma = (C, P, \varphi)$ *é* simples *se e somente se* $\#P = 1$.
2. *Define-se que dois AOSs* $\Gamma_1 = (C_1, P_1, \varphi_1)$ *e* $\Gamma_2 = (C_2, P_2, \varphi_2)$ *estão unidos (representados por* $\Gamma_1 \sim \Gamma_2$ *) da seguinte forma:*

$$\Gamma_1 \sim \Gamma_2 \Leftrightarrow C_1 \cap C_2 \neq \emptyset$$

3. *se* Γ_1 *e* Γ_2 *são simples e* $\Gamma_1 \sim \Gamma_2$*, então os respectivos provedores* $p_1 \in P_1$ *e* $p_2 \in P_2$ *são ditos* adjacentes *(*$p_1 \; adj \; p_2$*)*
4. *Define-se a* soma *de AOSs como um AOS* Γ *com as seguintes características:*

$$\Gamma = \Gamma_1 + \Gamma_2 + ... + \Gamma_n \Leftrightarrow \begin{cases} C = C_1 \cup C_2 \cup ... \cup C_3 & \wedge \\ P = P_1 \cup P_2 \cup ... \cup P_n & \wedge \\ \varphi = \varphi_1 \cup \varphi_2 \cup ... \cup \varphi_n \end{cases}$$

Dessa forma, é possível decompor um AOS qualquer em vários AOSs simples, cada um definido à partir de um dos provedores existentes no AOS original (*Princípio da Decomposição*).

3.3.1 Composições

Tanto provedores quanto componentes podem ser estruturados. A estrutura de cada um deles também corresponde a um ambiente de oferecimento de serviços, cada qual com características próprias. No caso do provedor, sua estrutura é composta de tal forma que as fronteiras com o ambiente externo é sempre feita através de provedores (os provedores de acesso).

DEFINIÇÃO 3.3 (ESTRUTURA DE UM PROVEDOR). *Dado um AOS simples $\Gamma = (C, P, \varphi)$ a estrutura interna do provedor $p \in P$ é também um AOS $\Gamma_p = (C_p, P_p, \varphi_p)$ com as seguintes características:*

1. *o conjunto dos provedores internos P_p é a união dos conjuntos de provedores de acesso e de infra-estrutura, isto é: $P_p = A_p \cup I_p$;*
2. *pelo princípio da decomposição, cada elemento de cada um dos conjuntos $A_p = \{a_1, ..., a_m\}$ e $I_p = \{i_1, ..., i_n\}$ dá origem a um dos AOSs simples $\Gamma_{a_1}, ..., \Gamma_{a_m}, \Gamma_{i_1}, ..., \Gamma_{i_n}$. A_p e I_p têm as seguintes características:*

$$\begin{aligned} \forall\, c \in C,\, \exists\, a_j \in A_p \quad &| \quad (c, a_j) \in \Gamma_{a_j} \\ \forall\, c \in C,\, \neg\exists\, i_k \in I_p \quad &| \quad (c, i_k) \in \Gamma_{i_k} \end{aligned}$$

   (isso representa a restrição de que todo usuário do serviço externo é usuário de algum serviço de acesso e que, ao mesmo tempo, não é usuário de um serviço de infra-estrutura);

3. *seja o AOS $\Gamma_x = (C_x, P_x, \varphi_x)$ o resultado da soma dos AOSs:*

$$\Gamma_x = \sum_{j=1}^{m} \Gamma_{a_j} + \sum_{k=1}^{n} \Gamma_{i_k}$$

   então define-se C_p e φ_p como:

$$\begin{aligned} C_p &= C_x - C \\ \varphi_p &= \varphi_x - \{(c, p) \,|\, c \in C\} \end{aligned}$$

Um exemplo da aplicação dessas definições pode ser visualizado na Figura 3.5. Nessa figura, representou-se um AOS com três usuários e um único provedor (isto é, um AOS simples). A estrutura interna desse provedor é formada por quatro outros provedores e cinco componentes de implementação do serviço. Dos quatro provedores, dois são de acesso e dois são de infra-estrutura, dando origem aos AOSs identificados na figura por: Γ_{a_1}, Γ_{a_2} (derivados dos provedores de acesso), Γ_{i_1} e Γ_{i_2} (derivados dos provedores de infra-estrutura). A estrutura interna do provedor original é definida pelo AOS Γ_p, que é o resultado da soma desses quatro AOSs retirando-se os componentes externos e as respectivas ligações (vide ítem 3 da Definição 3.3.

A estrutura de um componente composto é também um AOS, com as características apontadas pela Definição 3.4.

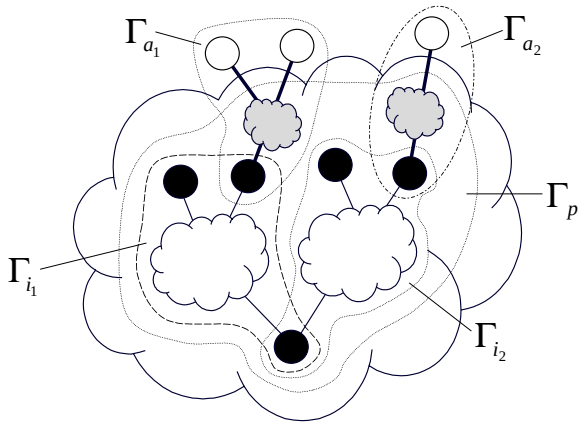


Figura 3.5: Exemplo de um ambiente de oferecimento de serviços e sua decomposição.

DEFINIÇÃO 3.4 (COMPONENTES COMPOSTOS). *Dado um AOS $\Gamma = (C, P, \varphi)$ e um componente composto qualquer $\theta \in C$, a estrutura interna desse θ é um AOS $\Gamma_c = (C_c, P_c, \varphi_c)$ tal que, ao se substituir θ pelos seus elementos internos, obtém-se um novo AOS $\Gamma_l = (C_l, P_l, \varphi_l)$ com as seguintes características:*

$$\begin{aligned} C_l &= (C - \{\theta\}) \cup C_c \\ P_l &= P \cup P_c \end{aligned}$$

e, em relação ao φ_l, observa-se a seguinte propriedade:

$$\forall(\theta, p_i) \in \varphi,\ \exists_1 c_j \in C_l \mid (c_j, p_i) \in \varphi_l$$

isto é, para cada ligação do componente composto a um provedor, existe um e somente um componente interno que estará ligado a esse mesmo provedor ao se substituir a composição por seus elementos internos.

Essa definição corresponde a descrição de estruturas da forma apresentada pela Figura 3.4 (pág. 27). Nesse caso, as fronteiras com o ambiente externo são sempre realizadas através de componentes.

3.3.2 Espaços e Subespaços

O conceito de um componente está associado à idéia de um *espaço* a ele reservado. Nesse espaço, todo o código de sua implementação e informações sobre seu estado interno serão armazenados. O *espaço de memória virtual (Virtual Memory Space — VMS)* de um AOS é definido como todo o espaço que pode ser utilizado por seus componentes.

O espaço de um componente pode, a qualquer instante da operação de um serviço ou aplicação, estar preenchido ou não. Em qualquer ambiente de oferecimento de serviços, deverá existir alguma forma de preencher e esvaziar os espaços dos componentes; esses processos serão conhecidos, respectivamente, por *encarnação* e *desencarnação.*

DEFINIÇÃO 3.5. *1. Uma VMS é definida como um conjunto de* células*:* $VMS = \{w_1, w_2, ..., w_n\}$.

2. Um espaço Ψ *é definido como um subconjunto qualquer de VMS, isto é:* $\Psi \subset VMS$.

3. Um subespaço Φ *de um espaço* Ψ *é um espaço tal que* $\Phi \subset \Psi$. *(Qualquer espaço é subespaço da VMS).*

A idéia de um espaço corresponde a um conjunto qualquer de células da VMS. Espaços podem ser divididos em *subespaços*, que correspondem a subconjuntos do conjunto de células daquele espaço. Os componentes internos de um componente composto ocupam subespaços desse componente. O AOS definido pela estrutura interna de um componente composto corresponde ao espaço desse componente no AOS externo.

3.3.2.1 Representações

Todo o componente tem uma *representação.* Cada representação está associada a dois conjuntos: um conjunto representando o domínio de valores válidos para encarnação em componentes com aquela representação, e outro definindo a forma de espaço utilizado para armazenar tais valores.

DEFINIÇÃO 3.6. *Uma* representação *é definida por um par* (V, π) *onde* V*, denominado* domínio *dessa representação, é um conjunto de* valores *e* $\pi \subset \mathbb{P}VMS$ *é um conjunto que define todas as possibilidades de armazenamento válidas para elementos de* V.

Uma representação define, além de todos os possíveis valores, todas as formas com que os valores do domínio podem ser armazenado em uma VMS. Essas possíveis formas de armazenamento são representadas através de todos os subconjuntos da VMS que pertencem a π.

Em qualquer momento da operação de um serviço ou aplicação, uma *função de ocupação* dos componentes presentes no AOS define como os componentes estão distribuídos pelo espaço da VMS. Essa distribuição deverá obedecer as restrições impostas pela representação de cada componente, que determina como cada valor pode estar armazenado.

DEFINIÇÃO 3.7 (FUNÇÃO DE OCUPAÇÃO). *Sejam* $\Gamma = (C, P, \varphi)$ *um AOS qualquer e* $\Omega = \{R_1, ..., R_n\}$ *o universo de todas as representação válidas, onde* $R_i = (V_i, \pi_i)$ *e* $V_1 \cap ... \cap V_n = \emptyset$ $(1 \leq i \leq n)$*. Seja* $V = V_1 \cup ... \cup V_n$ *o universo de todos os valores válidos de todos as representações. A função de ocupação é definida da seguinte forma:*

$ocup: C \rightarrow V \times \mathbb{P}VMS$ *tal que:*

$$\begin{aligned} dom(ocup) &= C \\ ocup(c_k) = (v_j, \Psi_n) &\Rightarrow \exists_1 R_m \in \Omega \mid v_j \in V_m \wedge \Psi_n \in \pi_m \\ ocup(c_p) = (v_r, \Psi_s) \wedge ocup(c_q) = (v_t, \Psi_u) &\Rightarrow \Psi_s \cap \Psi_u = \emptyset \end{aligned}$$

A Definição 3.7 supõe que os domínios das representações não tem valores em comum ($V_1 \cap ... \cap V_n = \emptyset$). Embora essa restrição possa ser relaxada, isso traria uma série de complicações de

notação que pouco acrescentariam em termos conceituais.[2] A função de ocupação define que todos os componentes presentes em um AOS em um determinado instante ocupam um espaço que é compatível com sua representação e que esses espaços não tem células em comum.

Componentes compostos, que seguem a estruturação da Definição 3.4, têm representação dita composta, enquanto componentes simples têm representações simples. O espaço utilizado por um componente composto é dividido em subespaços utilizados por outros componentes internos. Segundo a Definição 3.4, componentes compostos definem um AOS interno no qual as fronteiras com o ambiente externo são sempre realizadas através de componentes.

O espaço utilizado por componentes simples é dividido em dois subespaços: o primeiro, denominado de *área de código*, é onde fragmentos de código de implementação propriamente ditos estão contidos; o segundo, denominado *área de dados*, contém outros subespaços reservados para as *variáveis* que são internamente manipuladas pelo código.

Componentes de representação simples incluem: (i) números, caracteres, pixels de imagens, etc., todos denominados de *propriedades;* (ii) *operações*, cujo domínio corresponde a fragmentos de código e dados da implementação fornecidos por algum projetista e que podem ser utilizados para produzir resultados.

Componentes denominados *objetos*, podem ser tanto componentes de representação simples como de representação composta. Um *objeto de representação simples* é um componente no qual a área de código é subdivida em espaços, cada um reservado a uma operação (dita *método*) fornecida por um projetista, e a área de dados contém os subespaços reservados às variáveis compartilhadas por todas as operações desse objeto. *Objetos de representação composta* são formados por componentes internos (estruturados de forma a respeitar as regras da Definição 3.4) cujas representações podem ser de operação, de propriedade, ou de outros componentes compostos.

É importante notar a diferença entre a estruturação de componentes e a de componentes compostos: apesar de ambos serem internamente estruturados em subespaços, componentes simples não definem um AOS e, portanto, comunicações não são definidas no seu interior. Da mesma forma, variáveis, apesar de estarem contidas em subespaços, não são consideradas componentes. Já um componente composto define um AOS onde comunicações ocorrem entre seus componentes internos com o auxílio de provedores. O maior impacto dessa diferença é observado no fato de que componentes compostos, por serem modelados pela utilização de outros elementos mais simples do modelo, permitirão a configuração dinâmica nos ambientes em que isso for necessário, enquanto que componentes simples modelam elementos considerados básicos e sobre os quais o modelo não tem ascensão. Em especial, a existência de *objetos* de representação simples e de representação composta permite que o modelo seja versátil o suficiente para capturar tanto o comportamento de ambientes OO, nos quais objetos são entidades básicas, como de ambientes mais simples sobre os quais se deseja exatamente construir essa abstração a partir de outras mais primitivas.

Propriedades são componentes de representação simples cuja área de dados conterá um único subespaço destinado ao armazenamento de apenas uma variável. Supõe-se que a área de código em propriedades é automaticamente gerada e preenchida de forma a conter *seus métodos de acesso*, responsáveis por permitir a atribuição e a consulta ao valor armazenado na área de

[2] Na verdade, bastaria considerar que cada valor é, na verdade, um par $(valor, repres)$, de forma que não há mais como confundir valores de representações diferentes.

dados.[3] Como nesse caso o código não é explicitamente fornecido, será comum considerar que o domínio de representação de propriedades são os conjuntos de valores dos fragmentos que podem ser armazenados na área de dados (como os conjuntos de números ou caracteres propriamente ditos, por exemplo).

DEFINIÇÃO 3.8. *1. A representação (V, π) de um componente simples é definida de forma que o seu domínio é um par $V = (c, d)$ de conjuntos de fragmentos de* código *e* dados*, e $\pi \subset \mathbb{P}$VMS é um conjunto que define todas as possibilidades de armazenamento válidas para esses elementos.*

2. A representação (V, π) de um componente composto é definida para refletir sua estrutura interna como sendo a de uma nova VMS, subconjunto da original. Assim, o domínio de sua representação será simplesmente o próprio subconjunto de células da VMS original que será destinado ao seu armazenamento, isto é $V = \pi$.

3.3.2.2 Relações de Ancestralidade

A decomposição de espaços em subespaços gera relacionamentos denominados de *relações de ancestralidade*, conforme descrito pela Definição 3.9.

DEFINIÇÃO 3.9 (RELAÇÕES DE ANCESTRALIDADE). *Dado um AOS qualquer onde V é o universo de todos os valores válidos de todas as representações e $ocup : C \rightarrow V \times \mathbb{P}$VMS é a sua função de ocupação. Define-se que:*

1. *Um espaço A é dito* pai *de um espaço B (representado como $A \downarrow B$) se e somente se:*

$$\{A, B\} \subseteq \mathit{dom}(ocup) \ \wedge \ (B \subset A) \ \wedge \ \neg\exists\, C \in \mathit{dom}(ocup) \,|\, B \subset C \subset A$$

2. *Subespaços com um mesmo pai são ditos* irmãos.

3. *Um espaço A é denominado um* ancestral *(de nível n, $n > 1$) de um espaço C (representado como $A \downarrow^n C$) se e somente se: $\exists B \mid (A \downarrow B) \wedge (B \downarrow^{n-1} C)$.*

4. *Um pai é um* ancestral de nível 1*, isto é, $B \downarrow^1 C \equiv B \downarrow C$.*

5. *Convenciona-se que $A \downarrow^0 B$ significa que $A = B$.*

6. *Um espaço A é dito* filho *de um espaço B (representado como $A \downarrow^{-1} B$) se A é pai de B, ou seja: $A \downarrow^{-1} B \equiv B \downarrow A$.*

7. *Um espaço A é denominado um* descendente *(de nível n, $n > 1$) de um espaço C (representado como $A \downarrow^{-n} C$) se e somente se: $\exists B \mid (A \downarrow^{-1} B) \wedge (B \downarrow^{1-n} C)$.*

As definições acima são feitas de forma a ter-se:

$$(X \downarrow^n Y) \wedge (Y \downarrow^m Z) \ \Rightarrow \ X \downarrow^{n+m} Z$$

[3] Uma propriedade é equivalente a um objeto com um único valor na área de dados e dois métodos: um para atribuição e outro para consulta.

onde $n, m \in \mathbb{Z}$. Conseqüentemente, torna-se trivial mostrar que $X \downarrow^n Y \Leftrightarrow Y \downarrow^{-n} X$ (vide Teorema 3.2 e corolário no Apêndice 3A, pág. 56).

3.3.2.3 Visibilidade

A partir da definição de subespaços e ancestralidade, define-se o conceito de visibilidade, que será utilizado como forma de restringir e encapsular subespaços, impedindo que acesso sejam feitos, indiscriminadamente, a qualquer componente. Adota-se a notação $A \rightharpoonup B$ para representar o fato de que "A é visível a partir de B". Utiliza-se a notação $A \rightleftharpoons B$, equivalente sintático de $(A \rightharpoonup B) \wedge (B \rightharpoonup A)$, para representar o fato de que A e B são *visíveis entre si*. A definição de visibilidade encontra-se a seguir.

DEFINIÇÃO 3.10 (VISIBILIDADE). *B é visível a partir de C quando ele é um ancestral desse C, ou quando existe um ancestral A (de nível $n \geq 0$) de C tal que B é filho desse A, isto é:*

$$B \rightharpoonup C \equiv \begin{cases} B \downarrow^n C & \vee \\ \exists A \mid (A \downarrow^n C) \wedge (A \downarrow B) & \end{cases}$$

Um teorema bastante útil (denominado *teorema de visibilidade*) é decorrência direta dessa definição.

TEOREMA 3.1 (VISIBILIDADE). *Qualquer ancestral A de C e os filhos desse A são visíveis a partir C, isto é:*

$$(A \downarrow^n C) \wedge (A \downarrow^m B) \Rightarrow B \rightharpoonup C \quad \forall n \geq 0,\ m \in \{0, 1\}$$

Prova. Basta notar que se $m = 0$ então $A = B$, caso em que $B \downarrow^n C \Rightarrow B \rightharpoonup C$ é verdadeiro por definição; se, por outro lado, $m = 1$ então $(A \downarrow^n C) \wedge (A \downarrow B) \Rightarrow B \rightharpoonup C$ é também verdadeiro por definição.

□

Partindo desse teorema, é possível provar que um pai é sempre visível a partir de seus filhos e vice-versa, e que subespaços irmãos são visíveis entre si (vide Apêndice 3B, Teoremas 3.3 e 3.4, na página 57).

Um exemplo das relações de visibilidade pode ser desenvolvido a partir da configuração exemplificada na Figura 3.6.

De acordo com a Figura 3.6, as seguintes afirmações são verdadeiras:[4]

[4] Utilizou-se a seguinte notação:
$a \rightharpoonup \{b, c\} \equiv (a \rightharpoonup b) \wedge (a \rightharpoonup c)$
$\{d, e\} \rightharpoonup f \equiv (d \rightharpoonup f) \wedge (e \rightharpoonup f)$.

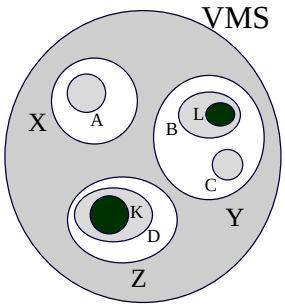


Figura 3.6: Exemplo de composição de componentes.

A partir de quem cada componente é visível	Quem é visível a partir de cada componente
$X \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, A\} \rightharpoonup X$
$Y \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, B, C\} \rightharpoonup Y$
$Z \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, D\} \rightharpoonup Z$
$A \rightharpoonup \{X, A\}$	$\{X, Y, Z, A\} \rightharpoonup A$
$B \rightharpoonup \{Y, B, C, L\}$	$\{X, Y, Z, B, C, L\} \rightharpoonup B$
$C \rightharpoonup \{Y, B, C, L\}$	$\{X, Y, Z, B, C\} \rightharpoonup C$
$D \rightharpoonup \{Z, D, K\}$	$\{X, Y, Z, D, K\} \rightharpoonup D$
$K \rightharpoonup \{D, K\}$	$\{X, Y, Z, D, K\} \rightharpoonup K$
$L \rightharpoonup \{B, L\}$	$\{X, Y, Z, B, C, L\} \rightharpoonup L$

Outra propriedade interessante da visibilidade é que o conjunto de espaços visíveis a partir de um determinado espaço é também visível a partir de todos os seus descendentes de qualquer nível (vide Teorema 3.5, Apêndice 3B).

Por default, relações de visibilidade são válidas para todas as composições, restringindo como espaços são visíveis uns aos outros. É possível porém, relaxar algumas dessas restrições tornando subespaços *visíveis através* de seus pais.

DEFINIÇÃO 3.11 (VISÍVEL ATRAVÉS). *1. Dados dois espaços A e B tais que $A \downarrow B$, tornar $B \triangleright A$ (leia-se B* visível através de A*) tem o seguinte significado:*

$$B \triangleright A \;\Rightarrow\; B \rightharpoonup X, \qquad \forall X \mid A \rightharpoonup X$$

2. A relação de "visibilidade através" se estende por qualquer cadeia de ancestrais onde cada pai torna o filho visível, isto é: sejam $A_1, A_2, ..., A_n$ espaços tais que $A_i \downarrow A_{i+1}$ $(1 \leq i \leq n-1)$, então:

$$A_n \triangleright^n A_1 \;\Rightarrow\; A_n \rightharpoonup X, \qquad \forall X \mid A_1 \rightharpoonup X$$

onde $A_n \triangleright^n A_1 \;\equiv\; A_n \triangleright A_{n-1} \triangleright A_{n-2} \triangleright ... \triangleright A_2 \triangleright A_1$

Com essa definição, o efeito tornar um espaço visível através de seu pai é o de promovê-lo ao mesmo nível de visibilidade de algum de seus ancestrais. Mais uma vez, essas definições são feitas de forma a ter-se $X \triangleright^m Y \triangleright^n Z \;\Rightarrow\; X \triangleright^{m+n} Z$.

Na Figura 3.7, apresenta-se um exemplo no qual alguns dos espaços da Figura 3.6 foram explicitamente definidos como visíveis através de seus respectivos pais. A convenção gráfica utilizada foi a de tornar parte dos contornos coincidentes para representar que o subespaço é

visível através de seu pai, isto é, no caso da Figura: $L \triangleright B, C \triangleright Y, D \triangleright Z$ e $K \triangleright D$.

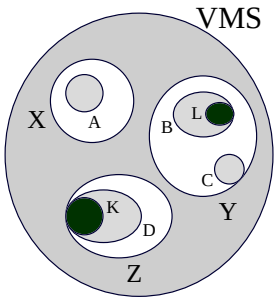


Figura 3.7: Tornando alguns espaços da Figura 3.6 visíveis através de seus pais.

Convém notar que, como $K \triangleright D$ e $D \triangleright Z$ (ou seja, $K \triangleright^2 Z$), K foi promovido ao mesmo nível de visibilidade de Z. Esse ponto também é facilmente observável na figura.

De acordo com a configuração da Figura 3.7, as seguintes afirmações sobre visibilidade pode ser feitas:

A partir de quem cada componente é visível	Quem é visível a partir de cada componente
$X \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, A, C, D, K\} \rightharpoonup X$
$Y \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, B, C, D, K, L\} \rightharpoonup Y$
$Z \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, C, D, K\} \rightharpoonup Z$
$A \rightharpoonup \{X, A\}$	$\{X, Y, Z, A, C, D, K\} \rightharpoonup A$
$B \rightharpoonup \{Y, B, C, L\}$	$\{X, Y, Z, B, C, D, K, L\} \rightharpoonup B$
$C \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, B, C, D, K, L\} \rightharpoonup C$
$D \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, C, D, K\} \rightharpoonup D$
$K \rightharpoonup \{X, Y, Z, A, B, C, D, K, L\}$	$\{X, Y, Z, C, D, K\} \rightharpoonup K$
$L \rightharpoonup \{Y, B, C, L\}$	$\{X, Y, Z, B, C, D, K, L\} \rightharpoonup L$

A visibilidade restringe a possibilidade de uma comunicação se realizar. Na realidade, para que serviços de comunicação possam ser efetivamente utilizados por um usuário, será necessário identificar outros componentes visíveis, como será detalhado na Seção 3.4.1.1.2.

3.4 Arquitetura dos Serviços

Definida a estrutura de componentes e provedores, pode-se passar à definição de serviços propriamente ditos. Serviços serão construídos fazendo uso dessas estruturas, conferindo-lhes uma semântica por intermédio da definição dos aspectos dinâmicos do modelo. A menção mais próxima que se fez até o momento sobre essa dinâmica está relacionada à definição dos espaços reservados ao armazenado de código de operações. Nada, porém, foi detalhado a respeito da forma de ativação ou execução dos mesmos.

3.4.1 Estilos de Cooperação e Suas Interfaces

Cada componente, seja ele simples ou composto tem uma *interface*. Interfaces definem como e quais informações são trocadas com um componente, estabelecendo o seu comportamento observável perante o provedor de serviços e os outros componentes. Esse comportamento reflete o propósito do componente dentro de um serviço ou aplicação.

Interfaces podem ser definidas para: (i) a interação entre componentes ou (ii) a interação entre componente e provedor. A definição desses dois tipos de interface está ligada à distinção entre os dois tipos de *serviços de comunicação básicos* (também denominados de *estilos de cooperação* — vide Capítulo 2) oferecidos por um provedor: (i) *serviços de comunicação explícita,* nos quais componentes interagem diretamente com o provedor através de uma interface dedicada ao envio e recebimento de informações, e (ii) *serviços de comunicação implícita,* nos quais a comunicação ocorre diretamente entre os componentes, sendo realizada com a "assistência" do provedor, mas sem uma interação direta com ele (ou mesmo percepção da sua existência). Usuários de um provedor e os dois tipos de serviço encontram-se representados simbolicamente na Figura 3.8.

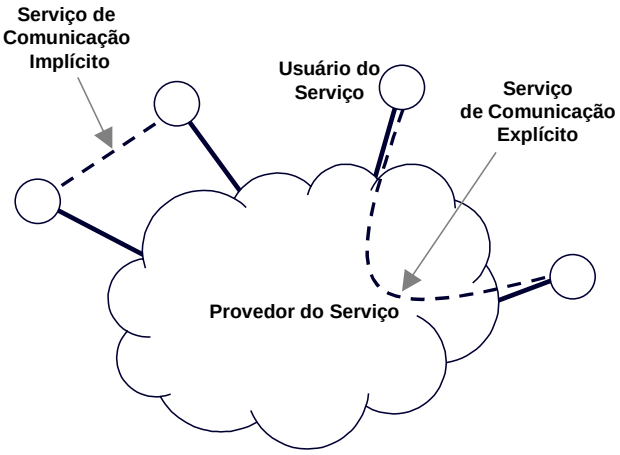


Figura 3.8: Representação dos elementos básicos do modelo de serviços e os dois estilos de comunicação.

Serviços oferecidos por camadas de protocolo através de interfaces como Sockets [126] são exemplos de serviços de comunicação explícita. Exemplos de serviços de comunicação implícita são serviços que oferecem suporte a chamadas de procedimento comum ou chamadas de procedimento remoto (Remote Procedure Calls — RPC) e similares, como chamadas de método remoto (Remote Method Invocation — RMI [29]) ou facilidades para chamadas de operações em ambientes baseados no CORBA [97].

3.4.1.1 Interfaces

Uma interface define um conjunto de *primitivas* de comunicação, que por sua vez, definem o conjunto de *fluxos* de informação trocadas a cada vez que uma primitiva é utilizada. Cada componente simples do tipo operação ou propriedade define uma primitiva para a solicitação da execução de sua área de código e, opcionalmente, uma primitiva para que essa operação possa retornar resultados. Em componentes compostos, a interface será definida pelo conjunto de interfaces dos seus componentes visíveis ao exterior, o que significa que essa interface pode estar organizada, recursivamente, em conjuntos. Objetos de representação simples tem sua interface idêntica a de componentes compostos, tornando-os, sob esse aspecto, indiferenciáveis.

Primitivas corresponderão a solicitações que um componente pode fazer para se utilizar do serviço de comunicação. Cada fluxo de informação trocado quando uma primitiva é solicitada pode ser: um *sinal,* uma *mensagem,* um *stream* ou *combinação* de um sinal com um outro fluxo qualquer. Sinais são utilizados para ativar a execução do código de alguma operação. Mensa-

gens correspondem a informações delimitadas em blocos de tamanho determinado, enquanto streams correspondem a informações de tamanho indeterminado e que exigem tratamento temporal e contínuo.

Cada fluxo definido em uma interface de um componente define também a existência de uma *porta* naquele componente. Portas concentram todas as abstrações necessárias ao consumo do fluxo correspondente, incluindo o armazenamento temporário das informações nos intervalos enquanto o componente ainda não as consumiu. Note que portas são definidas apenas como *entradas* do componente. Componentes simples do tipo operação e propriedade tem apenas uma porta. Componentes com várias portas podem ser obtidos através da utilização de componentes compostos. Assim a identificação de portas corresponde sempre à identificação de um subespaço.

A passagem de um sinal sozinho corresponde a simples ativação do código de um componente, que poderá então consumir as informações de suas portas (a forma com que isso é feito depende do próprio código do componente). A passagem de um sinal acompanhado de uma mensagem ou stream, além de ativar a execução do código, torna a informação em questão disponível. A passagem de uma mensagem ou fluxo sem sinal não ativa o código da operação no destino o que faz com que a informação não seja consumida (a menos que um sinal tenha sido enviado de outra fonte).

Primitivas são solicitadas dentro do código de uma operação. Tanto nos serviços de comunicação implícita como nos de explícita, primitivas dependerão da capacidade de identificação de subespaços. No caso de serviços de comunicação explícita, a identificação será entregue ao provedor; no caso de comunicação implícita, a identificação será utilizada diretamente como alvo da solicitação. A Seção 3.4.1.1.2 tratará de alguns aspectos relacionados à identificação.

3.4.1.1.1 Associações

Tanto o estilo de cooperação explícito quanto o implícito assumem que alguma forma de associação existe entre os componentes que devem se comunicar. Assim como os estilos de comunicação, associações também podem ser *explícitas* ou *implícitas*.

Em associações implícitas, componentes usuários detém alguma identificação dos componentes com quem se comunicam. A forma com que essa identificação é fornecida pode variar desde a codificação de endereços estabelecidos até a obtenção dinâmica de identificações através da comunicação com outros componentes. A partir dessa identificação, em uma comunicação com associação implícita, o provedor é responsável por encaminhar os fluxos de informação aos destinos.

Associações explícitas são criadas de forma a conectar pontos terminais de fluxos de informação. Uma associação explícita tem um ponto terminal fonte e um ou mais pontos terminais de destino, correspondendo, portanto, a uma abstração ponto-a-multiponto unidirecional, denominada de *pipe*. Pontos terminais são componentes simples do tipo operação. Ao serem criados, pipes passam a ser identificados nessas operações como um componente qualquer. No caso de utilizar-se de comunicação implícita, a identificação do pipe tomará o lugar da identificação do componente destino, de forma que a operação continua a achar que se comunica diretamente com o seu parceiro final. No caso de comunicação explícita, o pipe é quem oferece as primitivas explícitas de envio ou entrega das mensagens oferecidas pelo provedor. A Seção 3.4.1.2 trará maiores detalhes sobre as possibilidades de comunicação implícita e explícita, tanto através de associações explícitas como implícitas.

3.4.1.1.2 Notação para Identificação de Componentes

Duas formas de identificação são definidas: uma *relativa* e outra *absoluta*, que serão utilizadas, respectivamente, nas primitivas de serviços implícitos e explícitos.

Na forma absoluta, uma VMS será vista como uma memória associativa na qual todos os espaços e, recursivamente, subespaços nela contidos serão identificados por um *valor* de um tipo básico especial denominado de *endereço*. O formato utilizado por endereços não é relevante; qualquer alfabeto de símbolos pode servir como domínio desse tipo. Assim, para identificar o espaço cujo o endereço é ξ em uma VMS X, pode-se utilizar a notação $X[\xi]$. Porém, endereços só podem ser (corretamente) utilizados no código de componentes para os quais o referenciado é visível. A forma com que endereços de componentes se tornam conhecidos aos clientes pode variar: endereços fixos e conhecidos podem existir e ser utilizados diretamente durante a codificação dos serviços ou aplicações, ou eles podem ser comunicados ao longo da operação dos serviços ou aplicações.

Na forma relativa, componentes clientes são capazes de efetuar solicitações utilizando uma referência contida em uma *variável* do tipo endereço. Variáveis desse tipo só poderão conter endereços de espaços a ela visíveis. Além disso, a identificação relativa é baseada em uma identificação hierárquica à partir de componentes compostos. Nessa forma de identificação, cada espaço composto é visto como uma memória associativa, na qual qualquer de seus subespaços pode ser indexado por uma variável ou valor de um tipo básico qualquer. Nos exemplos, adotou-se as seguintes convenções sintáticas: letras minúsculas serão utilizadas para representar variáveis de um tipo básico qualquer, seja ele comum ou endereço; cadeias de caracteres (ou strings) serão representadas entre aspas, como nesse “exemplo”.

Exemplo 1 Um atributo *vetor de caracteres* pode ser modelado como um espaço composto A (de endereço ρ) particionado em subespaços $B_1, B_2, ..., B_n$, cada um do tipo *caracter*. A indexação desses subespaços em A é definida de forma a utilizar valores do tipo básico *inteiro não negativo*. Assumindo que uma operação cliente ocupa um espaço O de tal forma que $\{B_1, B_2, ..., B_n\} \rightharpoonup O$, o código contido em O poderá se utilizar de uma variável v, do tipo endereço, que poderá ser preenchida com o valor igual a ρ. Assim, o acesso a cada caracter do vetor poderá ser feito, nesse cliente, por meio da identificação $v[1], v[2],$ etc.

Exemplo 2 Um *objeto*, instância de uma classe qualquer que define uma interface com duas operações, pode ser modelado como um espaço composto de endereço é η, particionado em dois subespaços B_1 e B_2, cada qual do tipo *operação*. A indexação dessas operações será feita por valores do tipo básico *string, com valores* “opA” e “opB” respectivamente. Assumindo que uma operação cliente ocupa um espaço O de tal forma que $\{B_1, B_2\} \rightharpoonup O$, o código contido em O poderá utilizar uma variável x, com conteúdo igual η, que permite a identificação dessas operações no código do cliente por x[“opA”] e x[“opB”]. Convenciona-se que x[“$nome$”] pode ser escrito como $x.nome$, onde fica subentendido que *nome* é um valor do tipo string.

Subespaços de um mesmo espaço não precisam ser necessariamente indexados por valores do mesmo tipo; nada impede que um mesmo espaço x tenha subespaços identificados por $x.nome$ e $x[1]$.

3.4.1.2 Estilos de Cooperação

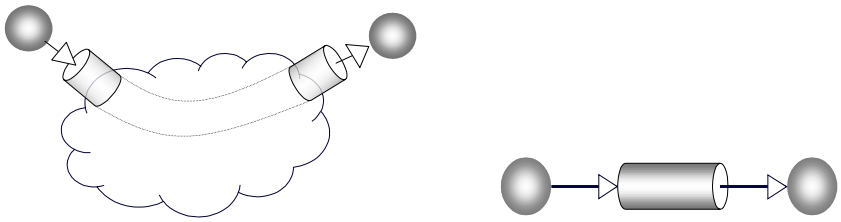
Os estilos de cooperação definidos no modelo correspondem a utilização de serviços de comunicação implícita ou explícita. Cada um desses estilos pode ser realizado através de associações implícitas ou explícitas. Nas seções subseqüentes, apresenta-se esses dois estilos com as respectivas opções de associação, e os possíveis relacionamentos entre as combinações resultantes.

3.4.1.2.1 Comunicação Explícita

Componentes que se utilizam de comunicação explícita são componentes do tipo operação denominados *filtros*. A interface desses componentes não é dependente dos usuários, sendo determinada apenas pela própria definição do provedor de serviços. Assim, filtros são componentes que interagem diretamente com o provedor, solicitando dele a capacidade de envio de informações a outros filtros.

Filtros não interagem entre si e podem nem mesmo ter conhecimento uns dos outros. Suas interfaces permitem apenas o recebimento e envio explícito de informações através do provedor. Duas formas de comunicação explícita são possíveis. Na primeira, cria-se uma associação explícita entre o filtro transmissor e o filtro (ou filtros) receptor(es) denominada de *pipe*. Essa criação é feita com base em valores do tipo endereço, fornecidos pelo componente que toma a iniciativa da criação (que pode ser feita tanto por um componentes usuários do serviço de comunicação para o qual essa associação está sendo criada, como um outro componente qualquer). Na segunda, filtros comunicam-se com o provedor indicando o destino do fluxo a ser enviando e o provedor, com base em informações internas, é capaz de encaminhar as informações

Pipes correspondem à representação explícita de uma associação unidirecional entre filtros. que pode ser ponto-a-ponto ou ponto-a-multiponto. A representação de pipes ponto-a-ponto encontra-se ilustrada na Figura 3.9.



(a) Representação da abstração de Pipe fornecida em um provedor.

(b) Representação simplificada.

Figura 3.9: Representação gráfica de associações explícitas entre os filtros levando a um estilo arquitetural da forma pipes e filtros.

Pipes podem definir ou restringir os tipos de dados das mensagens que transportam. Aos pipes podem ser associadas especificações de QoS utilizadas para definir e regular a forma de comunicação entre os filtros que os utilizam. Pipes que são associados a especificações de QoS passam a ser denominados *MediaPipes* [25, 24, 23].

MediaPipes fornecerão a abstração em relação aos mecanismos de implementação dos serviços utilizados para honrar os compromissos de QoS, que irão depender do tipo de ambiente

em que a comunicação estará acontecendo e das próprias características de qualidade requisitadas, conforme será apresentado em detalhes no Capítulo 4. A estruturação em camadas será utilizada como base para os mecanismos de abstração e reificação dos serviços, que permitirão, respectivamente, a programação em alto nível e as adaptações do serviço. A Figura 3.10 ilustra a abstração de um MediaPipe (em um nível *N*) com sua estrutura (no nível *N-1*) revelada.

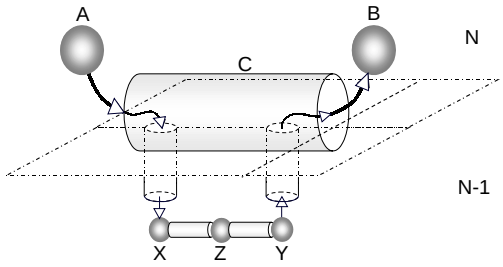


Figura 3.10: Estruturação em camadas.

As esferas *A* e *B* correspondem aos componentes (filtros) usuários (de nível *N*) de um serviço de comunicação provido através do MediaPipe *C*. Ao revelar a estrutura de *C*, obtém-se a visão do nível inferior (*N-1*) que, nesse exemplo, contém dois MediaPipes verticais e dois horizontais, além dos componentes *X*, *Y* e *Z*. Todos esses elementos (filtros e MediaPipes) são conjuntamente responsáveis pelo comportamento da abstração do MediaPipe do nível *N*. Cada um dos MediaPipes do nível *N-1* pode ser então, recursivamente, estruturado de maneira similar ao MediaPipe *C*. Os mecanismos de adaptação dos serviços corresponderão aos mecanismos de criação, destruição e modificação (através da reificação e modificação dos elementos internos) de pipes e componentes.

As primitivas utilizadas para comunicação explícita deverão ser da forma de solicitações para envio e entrega de informações. A interface *Sockets* corresponde a um exemplo de definição padronizada para esse tipo de interface [126].

3.4.1.2.2 Comunicação Implícita

Na comunicação implícita, a interface dos componentes de representação simples é definida como sendo composta de duas primitivas: uma *chamada* e uma possível *resposta*. Cada uma delas é pode ser uma combinação de um sinal com uma mensagem. Já a função da interface definida em tipos compostos é permitir ou impedir comunicação com os componentes internos fazendo uso da noção de visibilidade. Dessa forma, a interface de um tipo composto simplesmente expõe ou "se transforma" em um conjunto de outras interfaces: aquelas dos seus componentes internos que foram tornados visíveis ao exterior. Conseqüentemente, o mecanismo de composição pode ser utilizado como forma de organizar interfaces em grupos relacionados.[5]

Na comunicação implícita, interfaces definem primitivas que possibilitam as seguintes solicitações:

- *atribuições* ou *consultas* dos valores contidos no espaço de propriedades e suas *respostas*;

[5] A habilidade de um componente em apresentar interfaces organizadas em conjuntos é importante e está alinhada com alguns modelos de objetos como o ODP [69], ANSA [82], e o modelo proposto em [11] (que é, na realidade, uma extensão ao modelo do ODP). Essa habilidade permitirá que comportamento externo de um componente seja logicamente particionado de forma a refletir propósitos diferentes.

- *chamadas*, feitas a componentes do tipo operação e as respectivas *respostas*.

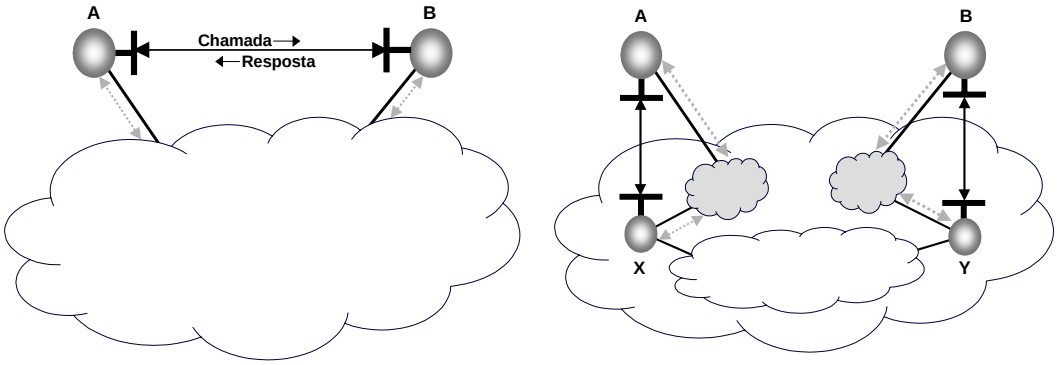
Para cada uma dessa duas formas, primitivas são fornecidas pelo provedor de forma a oferecer um serviço no qual os usuários identificam-se uns aos outros e trocam informações com base apenas na interface definida pelo tipo do componente alvo da solicitação, e não da interface do provedor em si.

Em uma atribuição de um caracter a uma posição de um vetor de caracteres, por exemplo, assumindo que o vetor é um componente composto identificado em um cliente de forma relativa por v, uma primitiva pode ser exemplificada nesse cliente por $v[1] := 'a'$. Analogamente, a primitiva para consulta pode ter a forma $x := v[2]$, onde x é uma variável do tipo caracter definida nesse mesmo cliente.

Para um exemplo de chamada e resposta, assume-se que um objeto, identificado em uma operação cliente através da variável o, é modelado como um componente composto que tem um componente visível do tipo *operação*. Se essa operação for identificada por uma *string* "op" e a assinatura for $(inteiro, caracter) \rightarrow real$, então a primitiva de chamada pode ser representada no cliente por $x := o.op(y, z)$, onde x é uma variável *real*, e y e z são valores ou variáveis dos tipos *inteiro* e *caracter*, respectivamente. A operação solicitada poderá, a qualquer momento de sua execução, utilizar uma primitiva $return(r)$ para devolver um resultado ao solicitante, onde r é uma variável ou valor do tipo *real*.

A comunicação implícita também pode ser feita através de associações implícitas ou explícitas. Em ambos os casos, as primitivas de comunicação escondem dos componentes a distribuição ou dispersão dos componentes envolvidos.

A Figura 3.11 ilustra a forma com que o provedor implementa, internamente, chamadas e respostas entre dois componentes utilizando-se de associação implícita.



(a) (b)

Figura 3.11: Comunicação implícita através de associação implícita.

A Figura 3.11(a) representa a visão dos usuários da existência de uma interface entre eles. A linhas pontilhadas claras indicam as primitivas utilizadas. A Figura 3.11(b) ilustra que a mesma interface que era antes oferecida pelo componente B é internamente oferecida por X. Assim a referência contida numa variável x no código de A conterá, nesse caso, uma referência ao componente X. Para A, porém, essa referência representa B. Esse processo pode se repetir, agora sobre os provedores acinzentados da figura. A comunicação entre X e Y pode ser tanto implícita como explícita, conforme comentado na Seção 3.4.1.2.3.

A Figura 3.12 representa a comunicação explícita através de associações explícitas.

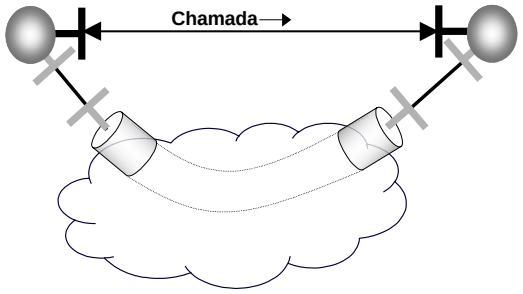
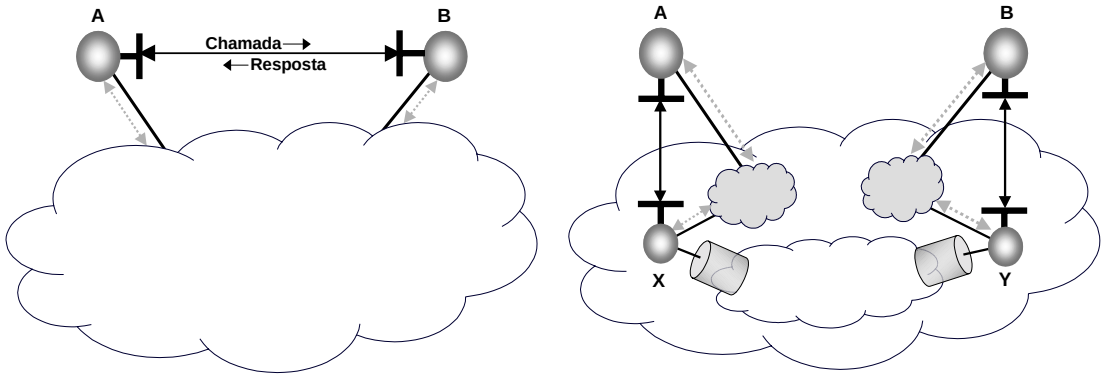


Figura 3.12: Comunicação implícita através de associação explícita.

Ao serem criados, pipes passam a ser identificados nessas operações como se fossem os componente finais. A identificação do pipe tomará o lugar da identificação do componente destino, de forma que a operação continua a achar que se comunica diretamente com o seu parceiro final. Para atingir este nível de transparência, a própria implementação do pipe oferecerá uma interface idêntica a do componente original de forma que as primitivas utilizadas são as mesmas do que nos casos anteriores. A Figura 3.12 ilustra a associação explícita apenas para a chamada; outra associação poderia também ser criada para a resposta.

3.4.1.2.3 Relação entre os Dois Estilos

A Figura 3.13 mostra como uma chamada na comunicação implícita pode se transformar, internamente, em uma comunicação explícita. Considere, por exemplo, que a operação cliente



(a) (b)

Figura 3.13: Um exemplo de relacionamento entre os dois estilos de cooperação.

A tem uma referência a um componente composto B contida em uma variável b. Considere ainda que B tem uma operação visível ao exterior identificada por "op". Assim, no código de A poderá se encontrar uma solicitação da forma $b.op(...)$. Porém, a implementação do serviço faz com que a referência contida em b seja, na realidade, uma referência a X de maneira que, essa solicitação será encaminhada ao componente interno ao provedor. Como X tem a mesma interface que B, tudo se passa sem o conhecimento de A. A implementação da operação

identificada por op no componente X, no exemplo da figura, será responsável por traduzir a solicitação em uma mensagem (mais, possivelmente um sinal) e transmití-la pelo pipe representado nessa mesma figura. Admitiu-se que esse pipe foi anteriormente criado de forma a associar explicitamente o componente X ao componente Y adequado, que irá transformar essa mensagem em uma solicitação idêntica à original, agora entregue diretamente ao B.

Note que, no caso acima, mesmo que o pipe tenha uma QoS estabelecida (i.e., ele seja um MediaPipe), não se tem comunicação com QoS fim-a-fim pois não existe garantia nos ambientes finais, apenas no intermediário. Essa é a situação comumente encontrada em ORBs convencionais, que se utilizam de conexões em redes como ATM para o suporte às suas atividades.

O exemplo anterior ilustrou a transformação da comunicação implícita sem associação explícita, em uma comunicação interna explícita com associação também explícita. Porém inúmeras outras transformações também são possíveis.[6] Em particular, tanto a comunicação implícita pode, internamente, ser transformada em explícita como a explícita pode ser transformada em implícita. Esse segundo caso é mais comumente observado no acesso do que na infra-estrutura; em outras palavras, as primitivas utilizadas para a comunicação explícitas passam a ser vistas pelo provedor de acesso como solicitações de chamadas (um operação $send(...)$, por exemplo).

3.4.2 Topologias Virtuais

Uma *topologia virtual* é uma abstração, definida sobre um provedor de serviço, que oferece a ilusão da existência de uma topologia, isto é, um conjunto de ligações entre componentes, através da qual eles podem se comunicar. A topologia virtual é uma abstração construída sobre um conjunto de informações de conectividade e estado do provedor, que pode incluir informações de roteamento, conexões estabelecidas, etc. Baseando-se nesse conceito, define-se a principal noção sobre a adaptação e criação de serviços:

> *A adaptação ou criação de serviços é, na realidade, uma questão de adaptação ou criação de topologias virtuais, que envolve a criação (ou modificação ou destruição) de componentes e de associações entre eles.*

A idéia de adaptação baseada no conceito de topologia virtual pode ser vista como uma reedição da noção de manipulação de uma *base de informações*, apresentada no Capítulo 2, onde as informações contidas na base vão desde a própria configuração de componentes ou protocolos, até o estado atual de carga na rede, e todo o conjunto de informações que, de algum modo, influenciam na forma com que informações são trocadas entre os usuários. A diferença básica consiste no fato de que a topologia virtual fornece uma abstração única sobre esse conjunto de informações.

Mecanismos convencionais como gerência e sinalização podem ser vistos como mecanismos de adaptação. A interface integrada de gerenciamento local (Integrated Local Management Interface — ILMI) definida pelo ATM Forum [4], por exemplo, define uma MIB para interfaces ATM que permitirá a recuperação e, eventualmente, a modificação de várias informações,

[6] Existem quatro opções de serviços: comunicação implícita com associação implícita, comunicação implícita com associação explícita, comunicação explícita com associação implícita e comunicação explícita com associação explícita. Assim tem-se, a princípio, dezesseis transformações como as descritas. Porém, cada provedor poderá internamente ter vários provedores e as tranformações utilizadas em um influenciam nos seus adjacentes. O importante será sempre manter a interface e as referencias corretas que fornecem a transparência desejada.

incluindo o estado corrente das conexões (VCCs e VPCs) em qualquer interface ATM. A disponibilidade desse tipo de informação em uma base durante a operação de um sistema pode permitir, por exemplo, que operadores configurem previamente um conjunto de conexões permanentes, criando serviços específicos e pré-definidos utilizando-se de ferramentas de gerência. Com mecanismos de sinalização, uma funcionalidade similar pode ser obtida de uma forma mais dinâmica, na qual conexões são estabelecidas e rompidas por ação dos próprios usuários e de acordo com a sua demanda. Ambos os mecanismos, porém, podem se utilizar da mesma base de informações do grafo de conexões, representando a topologia virtual para um ou mais serviços.

Porém, tanto a sinalização quanto a gerência são mecanismos de adaptação limitados. No caso da sinalização, por exemplo, pode-se apontar limitações em, pelo menos, dois aspectos: (i) a criação e destruição de associações entre componentes só é possível desde que a destruição se restrinja às associações que foram previamente criadas pelo própria sinalização, e (ii) novos componentes de implementação de serviço só podem ser criados a partir de uma "fôrma" ou "gabarito" preexistente, isto é: novas instâncias de componentes podem ser criadas a partir de tipos existentes, mas novos tipos não podem ser criados dinamicamente. Com relação a esse segundo ítem, uma das características diferenciadas em ambientes como os de redes programáveis e de redes ativas vem da capacidade de criação de novos tipos de componentes.

Topologias virtuais podem ser criadas "do nada" ou a partir de outras já existentes. Dependendo do grau de adaptabilidade do ambiente, a criação de topologias sem a utilização de outra como base pode ser limitada às fases de construção e implantação, que acontecem antes da operação dos serviços propriamente dita. Nesses casos, os mais comuns nos dias atuais, a prestadora de serviços é em geral a responsável por essa criação, empregando procedimentos que podem variar da instalação manual de componentes de hardware e software até a utilização de ferramentas de gerenciamento e configuração.

A criação de novas topologias a partir de outras existentes pode envolver a criação, modificação ou remoção de componentes de serviço no provedor, e a criação, modificação ou remoção de associações entre os componentes na infra-estrutura. A natureza do serviço e dos mecanismos de adaptação e criação pode limitar essas operações, como no caso da sinalização citado anteriormente.

Vários mecanismos de criação de topologias podem existir simultaneamente para um mesmo provedor. A forma com que eles influenciam uns nos outros pode ser a mais variada, desde a total dependência, quando um mecanismo age diretamente sobre a topologia resultante do outro, até a total independência, quando topologias completamente separadas e não relacionadas são geradas. Como exemplo, a Figura 3.14 poderia simbolizar a construção da topologia relacionada ao serviço de uma camada de comunicação do nível de rede, representada pelo provedor de serviços, utilizando-se da infra-estrutura do serviço do nível de enlace, representado pela infra-estrutura do serviço.

No plano mais abaixo da figura, encontram-se representados os componentes de serviço, que no exemplo correspondem a entidades de protocolo do nível de rede, configurados previamente pelo operador em cada nó de rede e ligados pela topologia dada pela camada de enlace. O segundo plano, logo acima, pode representar, por exemplo, o resultado da utilização de um mecanismo qualquer de provisão de informações sobre as rotas utilizáveis (que podem ser fornecidas através de qualquer mecanismo, desde configuração manual até a utilização de protocolos de roteamento). Dessa forma, protocolos de roteamento também podem ser vistos como mecanismos de adaptação de serviços. Esses dois planos, que constituem as duas primei-

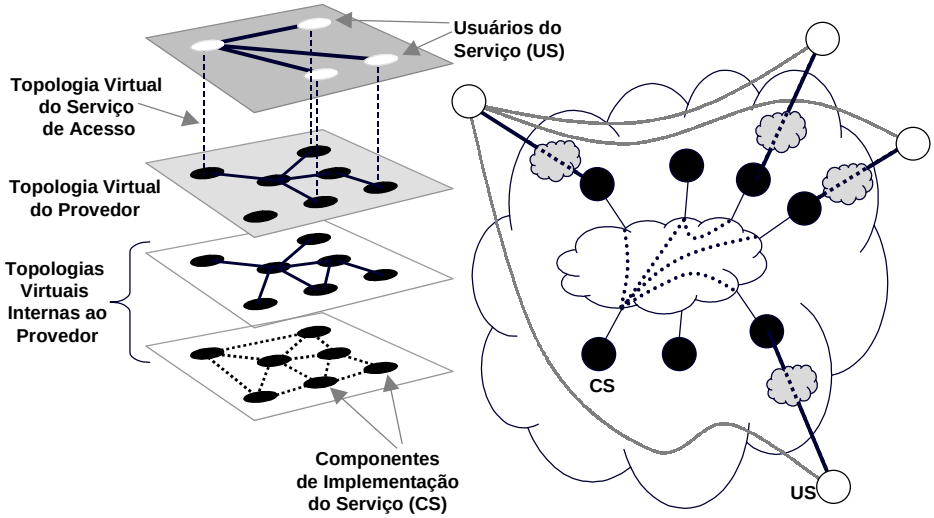


Figura 3.14: Exemplo da criação de topologias virtuais.

ras topologias virtuais, não são visíveis aos usuários. Já o terceiro plano, corresponde à uma topologia virtual diretamente utilizável pelos usuários, podendo ser, por exemplo, o resultado do mecanismo de sinalização que estabelece conexões utilizáveis. O último plano, localizado no alto da figura, representa o primeiro nível da topologia do serviço do nível superior.

O exemplo anterior ilustra uma situação onde as diversas topologias são dependentes umas das outras. A Figura 3.15 contém um exemplo simplificado de uma situação onde algumas topologias são independentes.

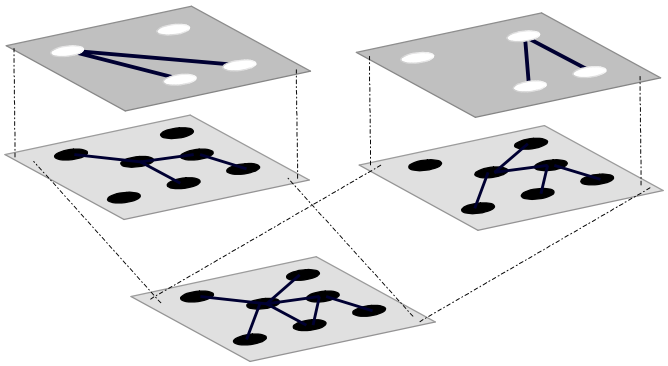


Figura 3.15: Exemplo de topologias virtuais onde algumas são independentes.

3.5 Arquitetura do Nível Meta

Um ambiente no qual adaptações a serviços estão disponíveis pode ser conceitualmente dividido em: (i) um ambiente de oferecimento do serviço, e (ii) um sistema (ou sistemas) que permite adaptações a esse ambiente — uma espécie de “meta sistema”.

Um *meta sistema* é um sistema que age ou dá significado a um outro sistema. No caso de serviços de comunicação, meta sistemas agem sobre serviços sendo por essa razão denominados

meta serviços. A arquitetura na qual os elementos de um serviço estão representados e disponíveis para a adaptação por meio de um meta serviço será denominada *arquitetura do nível meta* [127]. Um exemplo de arquitetura do nível meta encontra-se ilustrado na Figura 3.16.

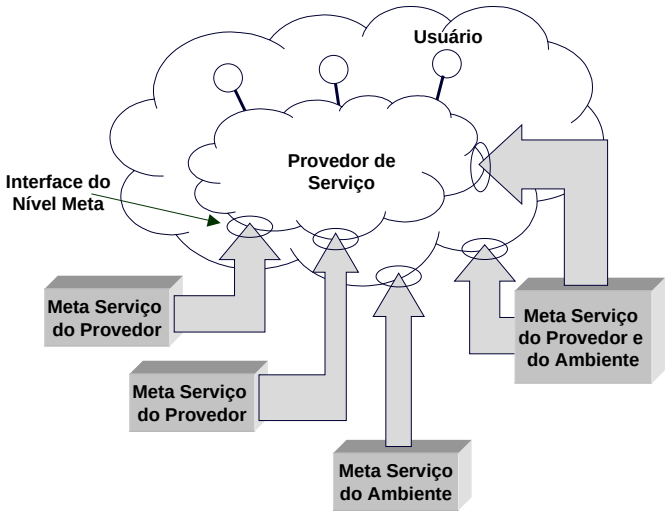


Figura 3.16: Arquitetura do nível meta.

A forma com que meta serviços são implementados pode variar desde procedimentos manuais ou ferramentas simples de instalação até ambientes complexos de processamento distribuído. Alguns exemplos de meta serviços comuns são os mecanismos de sinalização e gerência, e os protocolos de roteamento. Meta serviços podem atuar sobre serviços sem qualquer restrição de nível. Além disso, o mesmo provedor pode ser o alvo de vários meta serviços e um mesmo meta serviço pode agir sobre mais de um provedor, inclusive de níveis diferentes. O serviço sobre o qual um meta serviço atua é dito o *serviço alvo*.

3.5.1 Interface de Serviço do Nível Meta

Para que um meta serviço possa atuar sobre um serviço alvo, esse alvo deve apresentar uma *implementação aberta*. Implementações abertas expõem detalhes internos do provedor de uma maneira organizada e controlada. O conceito de serviço com implementação aberta é uma adaptação dos conceitos apresentados em [8, 108]. Um serviço com uma implementação aberta fornece (pelo menos) duas interfaces a seus usuários: (i) uma interface do *nível base (Base Level Interface — BL Interface)*, que permite o acesso às facilidades que, em última instância, constituem o próprio objetivo final do serviço alvo, e (ii) uma interface de *nível meta (Meta Level Interface — ML Interface)*, que revela aspectos da implementação do serviço alvo e permite a adaptação do comportamento do serviço obtido na interface do nível base por meio de modificações nesses aspectos. Em sistemas OO, a interface de nível meta é frequentemente denominada de *protocolo de meta objeto (Meta-Object Protocol — MOP)* [75].

3.5.2 Meta Serviços

Um meta serviço pode assumir diferentes formas. A maioria dos equipamentos como roteadores e comutadores, por exemplo, possuem alguma espécie de “ambiente de operação local” (ou

algo que poderia ser comparado a um sistema operacional em um computador comum) que possibilita um certo grau de configuração de suas funções, seja por meio de instalação de módulos de software ou atualizações no firmware.

Outros tipos de meta serviços permitem adaptações de forma mais dinâmica e distribuída, como no caso dos mecanismos de sinalização, redes programáveis e redes ativas. Em todos esses casos, o meta serviço é estruturado como um ambiente de oferecimento de serviços, no qual componentes se comunicam com o intuito de realizar modificações em pontos determinados do serviço alvo. Esse tipo de meta serviço e sua relação com serviço alvo encontra-se ilustrado na Figura 3.17.

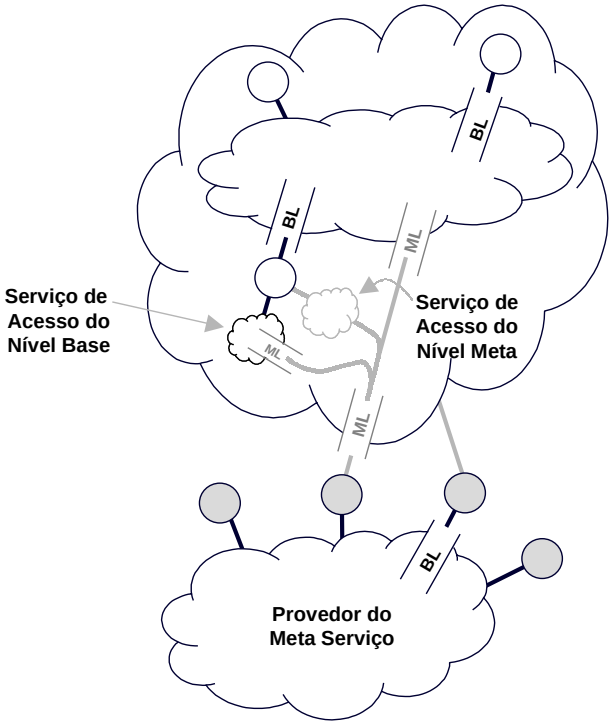


Figura 3.17: Meta serviço.

Utilizando-se da interface ML, usuários do meta serviço podem ter acesso aos elementos internos da implementação do serviço alvo, a saber: os provedores do serviço de acesso (do nível base), os componentes de implementação do serviço e os provedores de infra-estrutura. O acesso aos componentes de implementação é realizado por intermédio de um provedor de acesso especial denominado *provedor de serviço de acesso do nível meta (Meta-Level Access Service provider — MLAS provider).*

Ao se comunicarem através do provedor do meta serviço, usuários se utilizam da interface do nível base desse provedor, de forma que o meta serviço e seus usuários formam um ambiente de oferecimento de serviços como outro qualquer. Dessa forma, meta serviços podem ser o serviço alvo de outros meta serviços, constituindo o que se poderia chamar de “meta serviços de meta serviços” ou *torres de serviços e meta serviços*. Em arquiteturas com essa característica, o “topo da torre” será denominado o *serviço principal*. Como exemplo de uma torre de meta serviços, considere o roteamento e a sinalização em uma rede ATM. O serviço principal é plano do usuário, responsável pela comunicação das informações dos usuários; a sinalização é

um meta serviço e o protocolo de estabelecimento das informações de roteamento é um meta serviço desse meta serviço.

3.5.2.1 Introduzindo Reflexividade

Na Figura 3.17, o meta serviço representado atuava sobre um outro serviço que, de certa maneira, pode ser considerado independente do primeiro. No caso de se estar representando um mecanismo de sinalização, por exemplo, essa poderia ser a modelagem adequada de um sistema de sinalização *out-of-band*.

Um outro caso possível de modelagem surge quando se quer representar o fato de que o provedor do serviço e o provedor do meta serviço são, na realidade, um mesmo único provedor, como ilustrado na Figura 3.18.

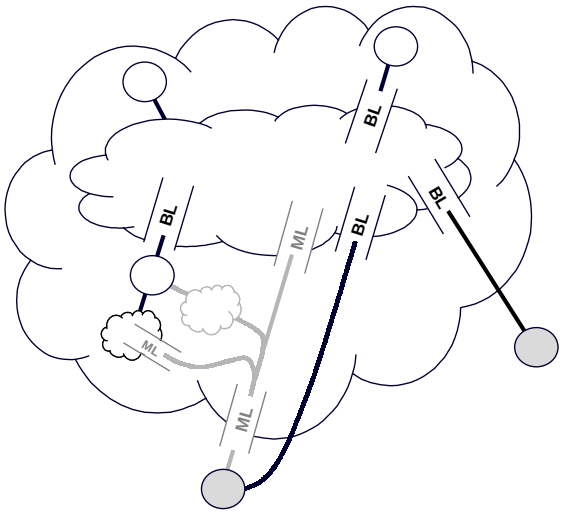


Figura 3.18: Arquitetura de nível meta representando o caso no qual existe um único provedor compartilhado para serviço e meta serviço.

Usuários do meta serviço utilizam a interface de nível base para se comunicar, da mesma forma que no caso anterior. Porém, nesse caso, essa interface é a mesma que a utilizada pelos usuários do serviço principal (já que o provedor é o mesmo). Esse tipo de arquitetura poderia ser a escolha para a modelagem de mecanismos de sinalização que se utilizam de um mesmo sistema de transporte para informações do usuário e de controle (sinalização). É importante notar, porém, que dependendo da forma de atuação do mecanismo de sinalização, tal modelagem levará a uma situação equivalente à anterior (com os provedores separados), encarada sob um ponto de vista diferente. No caso, por exemplo, em que a topologia virtual utilizada para sinalização for completamente independente daquela utilizada pelos usuários do serviço principal, pode-se considerar que, logicamente, existem dois provedores separados: um para sinalização e outro para o serviço principal.[7] Outro exemplo pode ser encontrado na especificação da re-

[7] Para que realmente as duas topologias funcionem como se existissem em provedores diferentes, os mecanismos de alocação de recursos devem garantir que o tráfego de uma não influencie na outra. Mecanismos de alocação "pessimistas" (baseados nas taxas de pico, p. ex.) podem ser utilizados com esse intuito. O importante aqui, porém, é que, em determinadas condições e sob pontos de vista específicos, as duas modelagens *podem ser* equivalentes, sendo a escolha de qual deve ser utilizada uma decisão do modelador.

de TMN do ITU-T, que pode ser definida como uma rede completamente separada (lógica ou fisicamente) e utilizada apenas para o propósito de gerência.

A configuração apresentada na Figura 3.18 ainda permite uma outra forma de utilização, quando se considera o fato de que, ao unificar os provedores do meta serviço e do serviço alvo, a comunicação entre os usuários desses dois níveis se torna possível (desde que, obviamente, não se esteja no caso em que as respectivas topologias virtuais são completamente separadas). Se essa comunicação se efetivar, a arquitetura exibirá uma capacidade denominada de *reflexão* ou *reflexividade*. Esse tipo de capacidade é interessante para modelar, por exemplo, a criação de novas instâncias de componentes. Nesse exemplo, os usuários que eram originalmente usuários do meta serviço[8] podem ser utilizados para modelar o que é usualmente denominado de *meta classe* (assumindo ambientes OO baseados em classes). Meta classes são componentes responsáveis, entre outras coisas, por atender a solicitações de criação de componentes de uma classe a qual eles representam. Essas solicitações podem ser originadas por usuários do serviço principal, correspondendo a utilização de serviços de comunicação implícita entre eles e as meta classes.

3.5.2.2 Arquitetura Reflexiva

A capacidade de reflexão originada pelo compartilhamento dos provedores do serviço alvo e do meta serviço, aliada a um outro tipo de unificação, dará origem a uma arquitetura denominada arquitetura *reflexiva*. Em uma arquitetura reflexiva, além dos provedores do serviço alvo e do meta serviço serem um único provedor comum, os próprios usuários desses serviços também podem se fundir, conforme indicado na Figura 3.19.

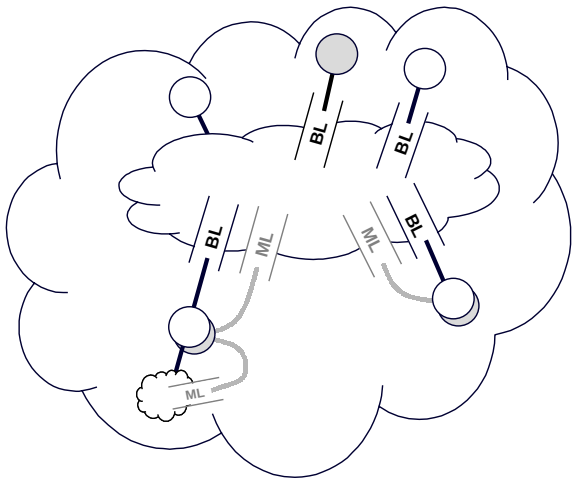


Figura 3.19: Arquitetura Reflexiva.

Como exemplo, no caso da sinalização, os mesmos componentes seriam os responsáveis por tratar da comunicação de informações de usuário e de controle. Além disso, alguns desses componentes também teriam acesso a interface do meta nível de forma a introduzir as informações necessárias relativas às conexões estabelecidas ou rompidas, efetuar reserva ou alocação de

[8] Como os todos usuários se utilizam do mesmo provedor, essa distinção é conceitual e diz respeito ao propósito a que serve cada componente.

recursos, etc. Esse tipo de sistema de sinalização é frequentemente denominado de *in-band*. Protocolos como X.25 [19], por exemplo, operam dessa forma.

3.6 Trabalhos Relacionados

O modelo proposto no presente capítulo encontra-se relacionado a trabalhos que pertencem, basicamente, a três linhas distintas: (i) propostas de modelos de objetos que incluem o tratamento de questões como o processamento de tipos contínuos e qualidade de serviço, (ii) propostas de modelos para novas arquiteturas de sistemas de comunicação e plataformas distribuídas e (iii) propostas de modelos para mecanismos que permitem adaptações de serviços e aplicações.

Modelos de objetos e arquiteturas tem sido, cada vez mais, tratados de forma integrada nos trabalhos relacionados a engenharia de serviços de telecomunicações. Os principais trabalhos nessa que seguem essa estratégia baseiam-se no suporte oferecido por um ambientes baseados em plataformas definidas pelo CORBA [97] ou pelo ODP [69]. O ODP, em particular, já prevê a existência de interfaces do tipo *fluxo* e de chamadas de operações síncronas e assíncronas. Também no ODP encontram-se conceitos que permitem o tratamento de diferentes abstrações no que concerne a construção de sistemas distribuídos. Ele divide a especificação de uma sistema em cinco pontos de vista (*viewpoints*), juntamente com um conjunto de conceitos genéricos que expressam cada um deles. Para cada ponto vista, uma linguagem própria pode ser definida para a utilização em outros padrões ou em implementações que especifiquem sistemas particulares, com seus requisitos próprios. A especificação de cada ponto de vista corresponde a uma abstração do sistema como um todo, porém tomada de apenas um determinado ângulo. Os cinco pontos de vista são:

- o *enterprise viewpoint*, que lida com os aspectos relacionados aos objetivos e funções do sistema no seu ambiente e as relações com o mundo externo. Trata do sistema e do meio em que ele está inserido, focalizando o papel desse sistema nesse meio;
- o *information viewpoint*, que se concentra na definição das informações utilizadas, processadas ou trocadas pelo sistema; não trata, porém, da forma de concatenação ou articulação do sistema no tratamento dessa informação;
- o *computational viewpoint*, que descreve o sistema como um conjunto de objetos atuantes na formação do sistema, sem entrar em detalhes de localização ou distribuição de objetos;
- o *engineering viewpoint*, cuja principal foco é o mecanismo de distribuição dos objetos;
- o *technology viewpoint*, que cuida dos detalhes relativos a definição do hardware e software que compõem os nós de processamento do ambiente distribuído.

O modelo proposto neste capítulo pode ser comparado às visões de informação, computacional e de engenharia. Porém, a divisão dessas visões não é feita de forma estática; a própria recursividade do modelo faz com que a visão computacional se transforme na visão de engenharia de um outro nível. A definição de meta serviços também pode ser encarada como uma forma de mudança no tipo de visão que se tem de um sistema. Os mecanismos que permitirão a abstração e reificação refletem esse tipo de mudança de visão, flexibilidade essa que não é prevista no ODP.

Alguns trabalhos no sentido de incorporar maior flexibilidade no tratamento de tipos, qualidade de serviço e sincronização no modelo de objetos do ODP têm sido realizados por um grupo da universidade de Lancaster, como os trabalhos descritos em [33, 31, 32, 34]. Nesses trabalhos, a definição de interfaces e tipos passa a poder contar com a especificação de tipos especiais denominados de *tipos contínuos*.

A questão da reflexividade em plataformas de componentes baseadas no ODP também é abordada nos trabalhos descritos em [9, 11, 8, 10, 30]. No presente trabalho, porém, três características podem ser ressaltadas como diferenças básicas em relação a essas propostas: (i) a existência de uma organização em camadas, que regula a definição dos mecanismos de adaptabilidade e orquestração, (ii) a definição de frameworks, que correspondem a estruturas ou modelos estabelecidos de forma a regular as adaptações a serem feitas no ambiente e (iii) a definição de uma modelagem que captura, especificamente, as abstrações comuns presentes em serviços de comunicação. Em relação a essa terceira característica, já existem modelos baseados no ODP especificamente voltados aos aspectos de serviços de comunicação como o TINA [6]. O TINA trata-se de um conjunto de esforços para transformar e encarar sistemas de comunicação como plataformas de desenvolvimento e programação de serviços baseados nos conceitos de orientação a objetos.

Esforços semelhantes aos mencionados acima são encontrados também para plataformas baseadas no padrão CORBA, como os descritos em [35, 134, 120, 119]. A maioria dessas propostas não menciona explicitamente os mecanismos de adaptabilidade de serviços, estando mais voltadas ao aspecto das abstrações para programação com qualidade de serviço. O foco nos mecanismos de adaptabilidade é encontrado com maior freqüência nos trabalhos relacionados a redes programáveis e redes ativas. Em particular, os trabalhos descritos em [17, 18] procuram fornecer um modelo que compare as estratégias utilizadas na adaptação de serviços nas diferentes propostas de arquiteturas para redes ativas e redes programáveis. O foco das comparações, porém, foi feito com base na definição de uma arquitetura genérica para os nós da rede que compõem os respectivos sistemas de comunicação. A classificação a ser apresentada na Seção 3.6.1 pode ser considerada mais genérica no sentido de que ela está baseada somente nos elementos do modelo conceitual definido para os serviços, que é, na realidade, um meta modelo a ser utilizado para a construção de arquiteturas.

3.6.1 Classificação das Formas de Adaptabilidade Segundo o Modelo

Na Figura 3.20 escolheu-se oito estratégias para serem classificadas segundo três parâmetros: o momento em que adaptações são realizadas, os atores dessas adaptações e os seus alvos.

Essas estratégias correspondem a mecanismos presentes nos mais diferentes tipos de redes e serviços. Nos parágrafos que se seguem, encontram-se resumidas algumas características e interpretações de cada estratégia.

Estratégia 1 A estratégia 1 corresponde aos mecanismos normalmente utilizados na construção tradicional de serviços. Projetistas estabelecem equipamentos, enlaces físicos, protocolos básicos, etc., tudo feito em tempo de construção do serviço.

Estratégia 2 Essa estratégia pode ser interpretada como o tipo de mecanismo utilizado pelos operadores para configurar a rede em termos dos protocolos dos níveis mais altos do que na estratégia 1, além de outras informações como conexões permanentes (associações).

Figura 3.20: Estratégias de adaptação de serviços.

Estratégia 3 A estratégia 3 corresponde as ações de adaptação efetuadas pelos operadores em concordância com o acordado ou solicitado por consumidores. A criação de enlaces e serviços específicos, por exemplo, são criados por este tipo estratégia.

Estratégia 4 A estratégia 4 é simplesmente a representação de um mecanismo normal de sinalização, no qual os próprio usuários adaptam os serviços (no caso de arquiteturas reflexivas), ou isto é feito através de um meta serviço separado.

Estratégia 5 Na estratégia 5, outros usuários (terceiros) podem adaptar no estado e na configuração de associações e entidades de protocolos, sendo essas adaptações possíveis tanto durante os períodos de comunicação como fora deles. Mecanismos de Gerência agem, basicamente dessa forma.

Estratégia 6 Mecanismos utilizados por protocolos para estabelecimento de rotas podem ser vistos como instâncias da estratégia 6. Um meta serviço pode ser o responsável por tais adaptações.

Estratégia 7 A estratégia 7 representa a capacidade de projetistas, operadores, usuários ou meta serviços podem ter de adaptar os serviços, incluindo a capacidade de adicionar novos protocolos à rede (representada pela última coluna da tabela). A possibilidade de que essas adaptações possam ser feitas durante a operação do serviço (apesar de ser antes da comunicação propriamente dita) revela tratar-se de mecanismos relacionados a redes programáveis.

Estratégia 8 Por fim, a estratégia 8 é representativa dos mecanismos presentes em redes ativas, nos quais as adaptações podem ser feitas (de forma reflexiva) pelos usuários, durante a própria comunicação e incluem a adição de novos componentes de protocolo.

Essa lista de estratégias não abrange todas as possibilidades e é bastante genérica em relação as especificidades de cada mecanismo. Ela serve, porém como ponto de partida para a avaliação da função que cada mecanismo pode ter em uma rede e auxiliar no levantamento de que mecanismos podem ser necessários em cada caso ou necessidade.

3.7 Sumário e Conclusões

Este capítulo apresentou um modelo genérico para a construção de serviços e arquiteturas de comunicação. Seus principais aspectos estão relacionados à modelagem dos elementos básicos que podem ser encarados como unidades de informação e configurados para a construção dinâmica de serviços. O fornecimento de abstrações de modelagem que incorporam conceitos de QoS deverá fornecer uma base conceitual para a criação de modelos específicos para implementação de plataformas, linguagens ou ambientes de desenvolvimento.

Os conceitos estabelecidos permitiram analisar alguns mecanismos comuns, presentes em redes tradicionais, bem como diferenciar de que forma as novas arquiteturas que tratam do problema da adaptabilidade se relacionam com esses mecanismos.

O modelo definiu dois tipos de serviços básicos (ou estilos de cooperação): serviços de comunicação *implícita* ou *explícita*. Cada um desses estilos pode ser realizado através de associações também *implícitas* ou *explícitas*.

O modelo para comunicação explícita com associações explícitas é o ponto onde se concentram as abstrações que tratam da provisão de QoS. Ele está baseado em um estilo arquitetural da forma *pipes e filtros*. Pipes correspondem à representação explícita da associação entre componentes (filtros) que, por sua vez, representam os elementos de processamento da implementação do serviço. Pipes podem ser ponto-a-ponto ou ponto-a-multiponto. Aos pipes serão associadas as especificações de QoS que serão utilizadas para definir e regular a forma de comunicação entre os filtros que os utilizam. Pipes que são associados a especificações de QoS passam a ser denominados *MediaPipes*.

Pipes também fornecerão a abstração em relação aos mecanismos de implementação dos serviços utilizados para honrar os compromissos de QoS, que irão depender do tipo de ambiente em que a comunicação estará acontecendo e das próprias características de qualidade requisitadas, como será abordado no Capítulo 4.

As primitivas utilizadas para comunicação explícita deverão ser da forma de solicitações para envio e entrega de informações. A interface *Sockets* corresponde a um exemplo de definição padronizada para esse tipo de interface. O mecanismo de composição pode ser utilizado como forma de organizar interfaces em grupos relacionados.

Na comunicação implícita, interfaces definem primitivas que possibilitam as seguintes solicitações: (i) *atribuições* ou *consultas* dos valores contidos no espaço de propriedades e suas *respostas* e (ii) *chamadas*, feitas a componentes do tipo operação e as respectivas *respostas*. Para cada uma dessa duas formas, primitivas são fornecidas pelo provedor de forma a oferecer um serviço no qual os usuários identificam-se uns aos outros e trocam informações com base apenas na interface definida pelo tipo do componente alvo da solicitação, e não da interface do provedor em si. A relação entre os dois estilos também foi estabelecida.

Com base nas formas de associação entre componentes, a noção de criação e adaptação de serviços pôde ser desenvolvida utilizando-se do conceito de *topologias virtuais*. A ação dos mecanismos de adaptação sobre essas topologias motivou a descrição do conceito de *meta serviços* e das *torres de serviços e meta serviços*. Com base nas formas em que as arquiteturas de nível meta se organizam, também definiu-se também os conceitos de reflexividade nesses ambientes.

Algumas das características diferenciadas do presente modelo também foram ressaltadas na Seção 3.6 como a existência de uma organização em camadas, que regula a definição dos mecanismos de adaptabilidade e orquestração e a utilização de frameworks como base da definição

de estruturas ou modelos que regulam as adaptações a serem feitas no ambiente.

Apêndice 3A: Ancestralidade

TEOREMA 3.2. *Se X, Y e Z são espaços, então:*

$$(X \downarrow^n Y) \wedge (Y \downarrow^m Z) \;\Rightarrow\; X \downarrow^{n+m} Z$$

Prova. (Simplificada).
Se $n = 0$ e $m = 0$ então os três são iguais e a afirmação é verdadeira.
Se $n = 1$ e $m = 0$ então $Y = Z$ e $(X \downarrow Y) \wedge (Y \downarrow^0 Z) \;\Rightarrow\; X \downarrow^{1+0} Z \equiv X \downarrow^1 Y$
Se $n = 1$ e $m = 1$ então $(X \downarrow Y) \wedge (Y \downarrow Z) \;\equiv\; (X \downarrow Y) \wedge (Y \downarrow^1 Z) \;\equiv_{def}\; X \downarrow^2 Z$
Aplicar indução.

□

COROLÁRIO 3.1. *Se X e Y são espaços, então:*

$$X \downarrow^n Y \;\Leftrightarrow\; Y \downarrow^{-n} X$$

Prova. Tomando $m = -n$ no teorema, tem-se que $(X \downarrow^n Y) \wedge (Y \downarrow^{-n} Z) \;\Rightarrow\; X \downarrow^0 Z$. Isto significa que se $X = Z$ então o lado esquerdo da sentença é sempre verdadeiro. Logo, substituindo Z por X:

$$X \downarrow^n Y \;\Leftrightarrow\; Y \downarrow^{-n} X$$

□

Apêndice 3B: Visibilidade

TEOREMA 3.3. *O pai é sempre visível a seus filhos e vice-versa, isto é:*

$$X \downarrow Y \;\Rightarrow\; X \rightleftharpoons Y$$

Prova. A definição de visibilidade (definição 3.10, pág. 34) já garante que o pai é visível a seus filhos, isto é:

$$(X \downarrow Y) \;\equiv\; (X \rightharpoonup Y) \tag{3.1}$$

Para mostrar que os filhos são visíveis ao pai, utiliza-se o teorema de visibilidade,

$$(A \downarrow^{n} C) \wedge (A \downarrow^{m} B) \;\Rightarrow\; B \rightharpoonup C \quad \forall n \geq 0,\, m \in \{0, 1\}$$

com $n = 0$ e $m = 1$, obtendo $(A \downarrow^{0} C) \wedge (A \downarrow B) \;\Rightarrow\; B \rightharpoonup C$. Mas essa situação corresponde a uma onde $A = C$, ou seja $C \downarrow B \;\Rightarrow\; B \rightharpoonup C$. Efetuando a substituição de variáveis:

$$(X \downarrow Y) \;\Rightarrow\; Y \rightharpoonup X \tag{3.2}$$

De 3.1 e 3.2, obtém-se o resultado desejado:

$$X \downarrow Y \;\Rightarrow\; X \rightleftharpoons Y$$

□

TEOREMA 3.4. *Espaços irmãos são sempre visíveis entre si, isto é:*

$$(A \downarrow C) \wedge (A \downarrow B) \;\Rightarrow\; B \rightleftharpoons C$$

Prova. É suficiente tomar $n = 1$ e $m = 1$ no Teorema de visibilidade

$$(A \downarrow^{n} C) \wedge (A \downarrow^{m} B) \;\Rightarrow\; B \rightharpoonup C \quad \forall n \geq 0,\, m \in \{0, 1\}$$

para mostrar que $(A \downarrow C) \wedge (A \downarrow B) \;\Rightarrow\; B \rightharpoonup C$.
Como os termos de uma expressão com operador $\wedge$ podem ser invertidos, basta inverter os termos no lado direito nesse mesmo teorema e tomar novamente $n = 1$ e $m = 1$ para chegar a $(A \downarrow B) \wedge (A \downarrow C) \;\Rightarrow\; C \rightharpoonup B$. Desses dois resultados, obtém-se que $(A \downarrow C) \wedge (A \downarrow B) \;\Rightarrow\; B \rightleftharpoons C$.

□

TEOREMA 3.5. *O conjunto de espaços visíveis a um determinado espaço é também visível a todos os seus descendentes de qualquer nível, isto é:*

$$(X \rightharpoonup A) \wedge (A \downarrow^n B) \Rightarrow (X \rightharpoonup B), \quad \forall X, \forall n \geq 0$$

Prova. Basta considerar os dois casos da definição de visibilidade e substituí-los no lado esquerdo da expressão do teorema:

1. $X \rightharpoonup A \equiv X \downarrow^m A \quad (m \geq 0)$
   Nesse caso,
   $(X \rightharpoonup A) \wedge (A \downarrow^n B) \equiv (X \downarrow^m A) \wedge (A \downarrow^n B) \Rightarrow X \downarrow^{m+n} B \equiv X \rightharpoonup B$
2. $X \rightharpoonup A \equiv \exists Y \mid (Y \downarrow^m A) \wedge (Y \downarrow X)$
   Nesse caso
   $(Y \downarrow^m A) \wedge (Y \downarrow X) \wedge (A \downarrow^n B) \Rightarrow (Y \downarrow^{m+n} B) \wedge (Y \downarrow X) \equiv X \rightharpoonup B$

□

Capítulo 4

Framework para QoS

A IDÉIA de se definir o framework apresentado neste capítulo nasceu da percepção de que as estruturas e mecanismos relativos à provisão de QoS são recorrentes nas várias escalas de distribuição e níveis de abstração presentes em um AOS qualquer, e independentes da diversidade de serviços passíveis de serem oferecidos pelo seu provedor de serviços. O uso do framework para provisão de QoS tornará mais fácil a um projetista de sistemas adaptar esses mecanismos às suas necessidades específicas, haja visto que ela:

- oculta detalhes relativos à distribuição dos componentes de um sistema;
- mostra, de maneira clara, como tornar flexível um sistema, com relação ao fornecimento de serviços, e
- permite a construção de partes isoladas de um sistema associadas à provisão de QoS por intermédio da utilização de elementos específicos do framework.

O trabalho apresentado neste capítulo é o resultado da revisão e extensão dos conceitos apresentados em [57] e [56]. Os principais pontos que mereceram considerável atenção e que diferiram da abordagem anteriormente apresentada nesses trabalhos dizem respeito a dois aspectos:

1. A representação dos frameworks tornou-se consideravelmente mais simples com a adoção de uma notação que privilegia o nível conceitual e não apresenta mais os detalhes da implementação de patterns específicos dedicados a adaptabilidade dos hot-spots (vide Cap. 2). Essa estratégia segue as idéias apresentadas por Fontoura [49, 48], embora a notação utilizada não seja exatamente a mesma.
2. Os diversos mecanismos são analisados sob a luz do modelo definido no Capítulo 3, estabelecendo quais desses mecanismos podem ser considerados meta serviços, sobre quais serviços eles atuam e de que forma o fazem.

4.1 Modelo de Provisão de QoS

Um modelo genérico de operação relacionado à provisão de QoS pode ser dividido nas seguintes fases:

- iniciação
- requisição para o estabelecimento de contratos de serviço
- estabelecimento de contratos de serviço
- controle e gerência de contratos de serviço
- rompimento de contratos e liberaçao de recursos.

4.1.1 Iniciação do Sistema

Para tornar um provedor de serviços operacional, é necessário primeiramente que seja definida a infra-estrutura que dará suporte aos serviços oferecidos e que determinará a forma de descrição do estado interno do mesmo. A definição do estado interno de um provedor de serviços inclui informações sobre recursos disponíveis no ambiente.

Na iniciação do sistema, as informações necessárias para a criação das "raízes" das *árvores de recursos virtuais* serão estabelecidas. Conforme ilustrado na Figura 4.1, a cada recurso real estará associada uma árvore de recursos virtuais.

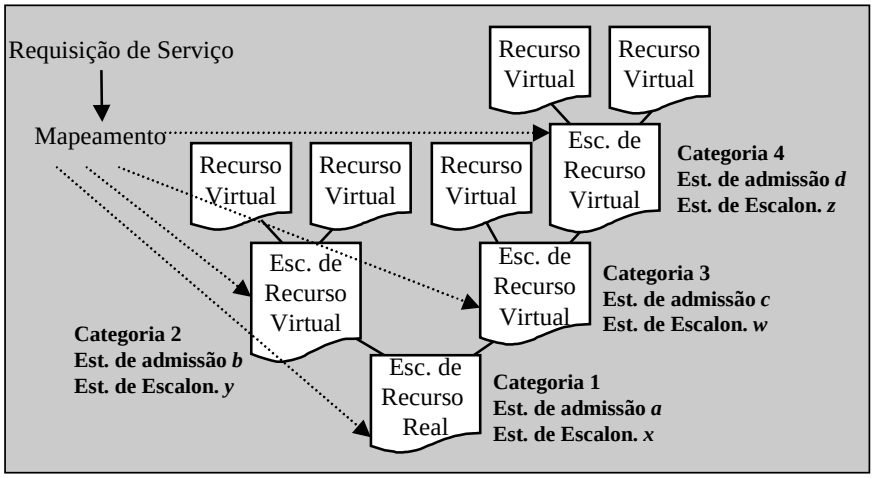


Figura 4.1: Árvore de recursos virtuais.

Cada escalonador mantém-se associado a uma determinada categoria de serviço e a políticas de provisão de QoS específicas (controle de admissão, escalonamento, etc.). Escalonadores de threads ou processos, e comutadores de circuitos virtuais são exemplos de mecanismos de escalonamento que podem ser implementados com base em árvores de recursos virtuais.

A partir das raízes das árvores de recursos virtuais, os nós que correspondem aos recursos virtuais (isto é, a parcelas da utilização de algum recurso) serão progressivamente criados durante a fase de estabelecimento de contratos de serviço.

4.1.2 Requisição para o Estabelecimento de Contratos de Serviço

Após a iniciação, usuários podem requisitar ao provedor, a qualquer momento, o estabelecimento de contratos de serviço. No que se refere às características de processamento e comunicação desejadas pelo usuário, uma requisição se dá através da:

- caracterização da carga de entrada a ser gerada, caso o usuário seja um produtor (fonte) de dados, ou de saída, caso ele seja um consumidor (destino), e da
- especificação da QoS que ele deseja do provedor.

O termo *carga* refere-se à dinâmica do fluxo de dados a ser gerado pelos eventos ocorridos na interface entre os usuários e o provedor, e que estão associados ao fornecimento do serviço requisitado.

A interface fornecida pelo provedor deve permitir requisições de serviço condizentes com as visões de QoS dos usuários. Isto significa que a caracterização da carga e a especificação da QoS devem ser facilmente compreendidas, ou seja, devem estar de acordo com o nível de abstração no qual os usuários estão inseridos.

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

4.1.3 Estabelecimento de Contratos de Serviço

O estabelecimento do contrato de serviço é o responsável, em última instância, pela criação de um recurso virtual: o MediaPipe entre entre os elementos participantes do serviço de comunicação principal. Os mecanismos de *mapeamento, roteamento, sintonização, controle de admissão* e a *alocação dos recursos* envolvidos estão ligados a essa fase.

A criação do MediaPipe (ou estabelecimento do contrato de serviço) é efetuada através da criação dos MediaPipes nos provedores internos adequados, incluindo os provedores de acesso e de infra-estrutura. A fase de estabelecimento começa quando o mecanismo de *controle de admissão* é informado sobre uma requisição de um novo contrato. O controle de admissão, em um provedor formado por vários provedores internos, aciona um conjunto de mecanismos que conjuntamente recebem o nome de *negociação,* cuja função é identificar todos os provedores internos que podem se envolver no fornecimento do serviço requisitado e atribuir a cada um deles uma parcela da responsabilidade pela provisão da QoS especificada na requisição. A negociação é, na realidade, o resultado da ação conjunta dos mecanismos de roteamento, mapeamento, sintonização e as novas requisições de estabelecimento aos provedores internos selecionados.

O *mecanismo de roteamento* é o responsável pela escolha dos subambientes adequados utilizando como base as informações de alcançabilidade dos componentes envolvidos. A informações de alcançabilidade serão fornecidas por mecanismos relacionados ao framework definido no Capítulo 5. Essas informações tornam-se disponíveis por intermédio de um mecanismo de divulgação de informações das rotas disponíveis e utilizáveis em um sistema de comunicação, mecanismo esse comumente incluido nos *protocolos de roteamento*.

O roteamento é seguido pela *sintonização*, que é responsável por definir, quantitativamente, as parcelas de responsabilidade de cada subambiente escolhido. O mecanismo de sintonização depende de alguma estratégia ou heurísitca de particionamento, que permita decidir como será feita essa divisão de responsabilidades entre os subambientes escolhidos pelo roteamento. A partir desse particionamento será possível gerar as novas requisições de estabelecimento de contratos de serviço aos procedores selecionados. Como exemplo, considere uma requisição na qual a QoS solicitada limita o retardo de transferência absoluto em x unidades de tempo. A

criação do MediaPipe com essa característica envolverá a criação de um número n de MediaPipes concatenados em provedores internos ao provedor original. O mecanismo de estabelecimento deverá atribuir a cada um dos n MediaPipes uma parcela $\frac{x}{p_1}, ..., \frac{x}{p_n}$ de unidades de tempo tais que $\sum_{i=1}^{n} \frac{1}{p_i} = 1$, utilizando esses valores no estabelecimento de cada um dos n contratos de serviços.

Como o estabelecimento envolve a participação de diferentes provedores, cada qual possivelmente associado a uma visão de QoS distinta (conforme mencionado na Seção 4.1.2), torna-se necessário o *mapeamento* das requisições para o estabelecimento de contratos nesses diferentes provedores.

Todo esse processo, envolvendo o roteamento, sintonização, mapeamento e as novas requisições aos provedores internos, se repete até chegar-se ao nível em que o mecanismo de controle de admissão pode efetuar testes diretamente sobre um recurso virtual sem repassá-los a nenhum provedor interno. Se o teste indicar a disponibilidade de recursos, o controle de admissão retorna uma resposta afirmativa. No nível do provedor externo, a negociação receberá de volta todas as respostas de todos os seus provedores internos e se todas forem positivas, o controle de admissão pode retornar ao seu provedor externo uma resposta positiva. Se, em algum dos níveis, um ou mais dos provedores internos não oferecer a disponibilidade desejada, a negociação tem ainda a possibilidade de tentar alterar o balanceamento gerado pela heurística daquele nível ou alterar a escolha dos provedores envolvidos (por meio de um novo roteamento, p. ex.), resubmetendo as novas parcelas aos respectivos controle de admissão.

Se, ao final de todo o processo, o ambiente tiver recursos suficientes, o provedor admitirá o fluxo do usuário e fará a alocação dos recursos necessários para servi-lo (levando possivelmente a uma mudança de estado interno), de tal forma que a QoS especificada seja satisfeita. Senão, ele poderá rejeitar o fluxo do usuário ou lhe sugerir uma outra especificação de QoS factível de ser satisfeita. Nesse último caso, se o usuário aceitar a nova proposta, seu fluxo é admitido, senão é rejeitado, podendo o usuário tentar requisitar o serviço novamente em outra oportunidade.

4.1.4 Controle e Gerência da QoS

Após o fluxo do usuário ser admitido, o provedor 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. Dentre os mecanismos responsáveis pelo controle e gerência da QoS, estão incluídos os de *monitorização de fluxos* e a *ressintonização da QoS*.

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

Os mecanismos de *ressintonização* da QoS são responsáveis pela manutenção da orquestração de recursos definida durante o estabelecimento do contrato sem que haja a necessidade de interrupção do fornecimento do serviço. Quando um provedor gerencia vários recursos, se um deles não puder suportar sua parcela de responsabilidade pela provisão da QoS fim-a-fim (devido a uma sobrecarga, por exemplo), caberá ao mecanismo de ressintonização efetuar operações de ajuste ou, em casos mais drásticos, escolher provedores alternativos, a fim de manter

a QoS previamente negociada. A ressintonização é similar à sintonização, operando para realizar as correções necessárias a uma sintonização prévia. A ressintonização envoverá, em geral, mecanismos que deverão atuar de forma mais ágil do que os da sintonização inicial (realizada durante o estabelecimento) já que ressintonização acontece ao longo da utilização do serviço pelo usuário.

Um bom exemplo da atuação dos mecanismos de ressintonização está na correção de variações no retardo de trânsito de dados em um sistema de comunicação. Caso o mecanismo de monitorização detecte, em um determinado fluxo, a presença de variações no retardo de trânsito que estejam fora das especificações do contrato estabelecido, o mecanismo de ressintonização da QoS pode ser acionado para, por exemplo, ajustar os mecanismos de escalonamento de pacotes tanto nas estações finais quanto nos nós intermediários da rede de distribuição, para melhor delimitar as variações no retardo de trânsito dos dados desse fluxo. Em casos mais drásticos, o mecanismo de roteamento pode ter que ser acionado de forma a escolher um ou mais novos provedores que possam voltar a estabelecer as características necessárias.

4.2 Serviços e Meta Serviços

Os serviços de estabelecimento de contrato e de controle e gerência da QoS podem ser considerados como meta serviços do serviço principal, estabelecendo e controlando o funcionamento de topologias virtuais, conforme ilustrado na Figura 4.2. O serviço de requisição e estabeleci-

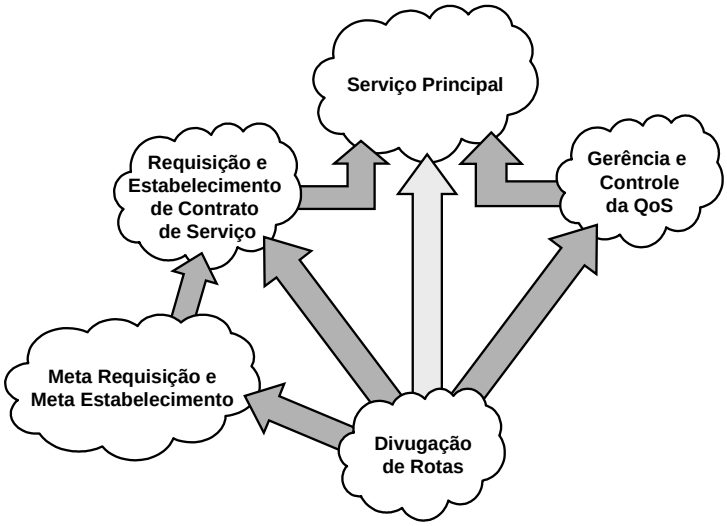


Figura 4.2: Meta serviços definidos pelo framework de QoS.

mento de contrato pode ser implementado através de um serviço de sinalização, por exemplo. Nesse caso, é possivel que outros meta serviços semelhantes sirvam de base para o estabelecimento da topologia virtual utilizada para a a sinalização, como é o caso da “meta sinalização” definida pelos padrões ATM,[1] serviços esses genericamente denominados de meta requisição e meta estabelecimento.

A divulgação de rotas é responsável por tornar disponíveis as informações que permitirão a escolha dos provedores que participarão na composição dos contratos requisitados. Esses

[1] Teoricamente, esse processo pode se repetir idefinidamente, criando uma “meta meta sinalização”, por exemplo.

protocolos servem de meta serviços para os demais serviços. Em particular, em redes onde não há estabelecimento ou requisição de QoS, esse meta serviço atuará diretamente sobre o serviço pincipal, seguindo a seta mais clara da Figura 4.2. Caso contrário, o meta serviço de distribuição de informação de roteamento atuará apenas sobre os demais serviços.

O escalonamento é um meta serviço especial (e por isso não é representado na figura) responsável pela própria abstração de recurso virtual e, por conseguinte, da própria noção de topologia virtual. O conceito de serviço, como definido na Seção 3.4.2, está de tal forma atrelado à existência do escalonamento que sua representação passa a ser desnecessária, subentendendo-se que ele se faz necessário como parte do próprio serviço principal.

4.2.1 Exemplo em uma Rede ATM

A Figura 4.3 exemplifica o cenário descrito para o caso da sinalização, meta sinalização e protocolo de roteamento em uma rede ATM. Os serviços de sinalização, meta sinalização e o

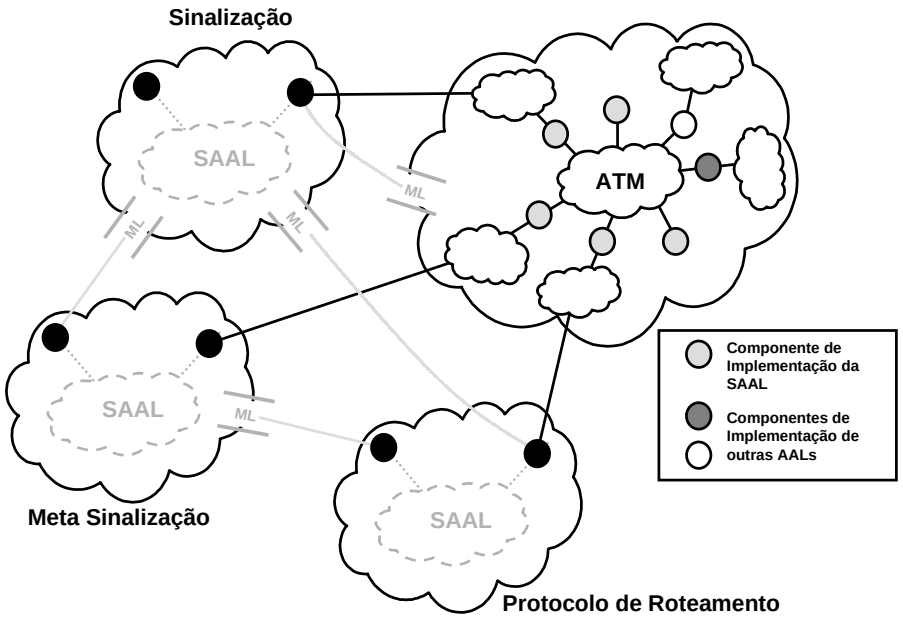


Figura 4.3: Exemplo de serviços e meta serviços para a provisão da QoS em redes ATM.

protocolo de roteamento são todos implementados sobre uma camada de adaptação específica de sinalização denominada de SAAL (Signaling ATM Adaptation Layer). A SAAL (assim como as demais AALs definidas pelo modelo de referência do ATM) é implementada diretamente sobre a camada ATM, que é comum ao serviço principal (implementado pelo plano do usuário) e aos demais serviços. Assim, introduz-se um nível de reflexividade no qual os elementos usuários de cada serviço atuam de forma independente. As informações estabelecidas pelo protocolo de roteamento servirão tanto à sinalização quanto à meta sinalização. A meta sinalização estabelece a topologia virtual utilizada pela sinalização que, por sua vez, cria a topologia virtual utilizada pelo serviço principal.

4.2.2 Orquestração

O conjunto de mecanismos que efetuam o particionamento da QoS entre os vários provedores internos e controlam seu funcionamento é coletivamente denominado de *orquestração*. Assim, a orquestração atua nas fases de estabelecimento (através dos mecanismos de negociação) e no controle e gerência da QoS (através das possíveis ressintonizações). Como exemplo, caso algum MediaPipe, em um determinado momento, viole a qualidade acordada, a orquestração pode reparticionar os parâmetros de qualidade entre os MediaPipes internos de forma a restabelecer a qualidade fim-a-fim. Todos os mecanismos ligados à orquestração podem ser implementados segundo um modelo *centralizado* ou um modelo *distribuído*.

Modelo Centralizado

No modelo centralizado, ilustrado na Figura 4.4, um *centralizador de orquestração* é designado para o provedor. Cada subambiente é responsável pela orquestração de seus próprios recursos, enquanto o centralizador trata da coordenação entre eles. A orquestração interna em cada um dos subambientes pode ser realizada de maneira centralizada ou distribuída.

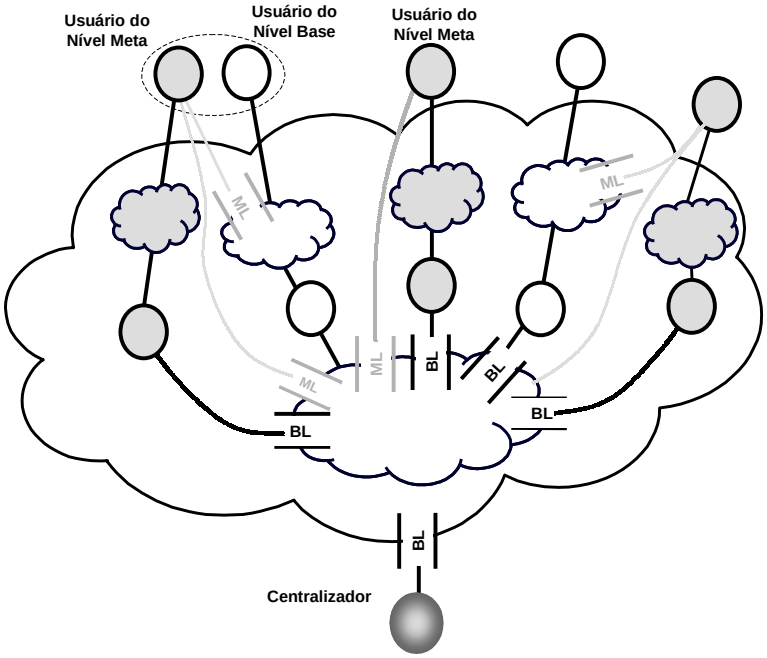


Figura 4.4: Modelo centralizado para a orquestração.

Os componentes acinzentados representam componentes do meta serviço, que pode ser a negociação ou a ressintonização. Eles se comunicam através do provedor de serviços alvo de forma a contactar o centralizador para obter sua parcela dentro do balanceamento geral que ele estipular.

Modelo Distribuído

Ao contrário do modelo centralizado, no modelo distribuído, ilustrado na Figura 4.5, a orquestração não é coordenada por um único elemento e sim feita de maneira cooperativa, por

meio da ação conjunta da orquestração dos subambientes. Isto significa que todos os subambientes são igualmente responsáveis não só pela orquestração dos seus próprios recursos, mas também pela integração com os demais subambientes. A forma exata com que essa ação conjunta será realizada depende de cada ambiente. De uma forma geral, ela dependerá da troca das informações globais entre os diversos elementos do meta serviço. Novamente, a orquestração de recursos interna de cada subambiente pode ser feita de maneira centralizada ou distribuída.

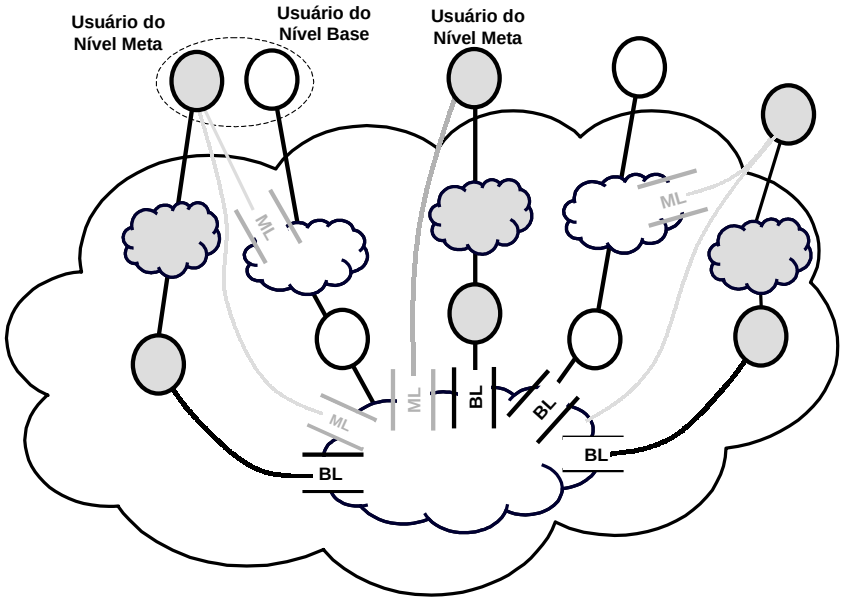


Figura 4.5: Modelo distribuído para a orquestração.

4.3 Elementos do Framework

Com base na definição dos mecanismos apresentada na seção anterior, o framework para provisão de QoS será obtido a partir da aplicação conjunta de um conjunto de frameworks menores:

- Framework para Parametrização de Serviços;
- Framework para Escalonamento de Recursos;
- Framework para Negociação de QoS e
- Framework para Sintonização de QoS.

4.3.1 Notação

Todos os elementos do framework para QoS são descritos por meio de uma notação textual informal, auxiliada de uma notação gráfica baseada na linguagem unificada de modelagem (Unified Modeling Language — UML [109]).

Durante a descrição textual, foi utilizada também uma grafia diferenciada ao nomear classes e métodos abstratos (em *itálico*) e classes e métodos concretos, além de atributos e relacionamentos. Já as dicas de implementação incluem pedaços de código escritos em uma pseudo-notação derivada da linguagem Java.

Os hot-spots são representados, nos diagramas de classe, através de métodos contidos em caixas acinzentadas e com a grafia em itálico. Este abordagem para a representação de hot-spots é decorrência direta das idéias apresentadas em [48], embora, nesse trabalho, uma linguagem bem mais completa tenha sido proposta.

4.4 Framework para Parametrização de Serviços

A adaptação dos mecanismos de provisão de QoS a serviços e escalas de distribuição específicas é essencialmente guiada segundo a forma de descrição do comportamento dos fluxos dos usuários e da infra-estrutura existente no ambiente. Todas essas informações podem ser estruturadas através do uso de parâmetros de caracterização do serviço. Um parâmetro é a imagem de uma propriedade (relacionada a uma grandeza qualquer, como tempo, vazão, etc.) existente ou desejável em um provedor, dotada de um descritor que a nomeia e associada a um valor de um tipo qualquer (inteiro, ponto flutuante, lista de outros valores, etc.).

A partir das formas possíveis de utilização de parâmetros, pode-se definir três tipos de parâmetros diferentes:

- *parâmetros de desempenho do provedor*, que descrevem o próprio estado interno atual do provedor, ou seja, a habilidade do provedor em fornecer os serviços requisitados pelos usuários. Vazão disponível (em um recurso) e retardo de trânsito (entre dois pontos) são exemplos de parâmetros de desempenho do provedor;

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

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

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

Para tornar a parametrização de serviços em um provedor qualquer suficientemente flexível, definiu-se uma estrutura, correspondente ao núcleo do framework para provisão de QoS, responsável por definir um esquema de parametrização que seja independente dos possíveis serviços a serem oferecidos pelo provedor. Este núcleo é modelado pelo *framework para parametrização*. O resultado de sua instanciação é, em última análise, a definição de uma base de informações de QoS sobre a qual os diversos mecanismos de provisão de QoS definidos pelos frameworks de escalonamento, negociação e sintonização podem atuar.

Utiliza-se a noção de grandeza como fundamento para a definição de parâmetros abstratos. Pode-se definir, como exemplificado mais à frente, o parâmetro base *retardo* a partir do qual podem ser derivados outros parâmetros mais específicos como um retardo de trânsito. Porém,

esse também é um parâmetro abstrato, logo várias outras especializações podem ser necessárias (criando uma hierarquia de derivação de parâmetros), de forma a se conseguir um parâmetro concreto.

O framework de parametrização de serviços também se vale de *qualificadores* para estruturar parâmetros. Em especial, parâmetros podem ser descritos como funções de outros parâmetros por meio da definição de *qualificadores de dependência funcional*. A partir do parâmetro vazão, por exemplo, pode-se definir vazão média, vazão máxima, variância da vazão, etc.

4.4.1 Categorias de Serviço

Os parâmetros de caracterização de serviços podem ser divididos em subconjuntos (não necessariamente disjuntos) que definem as *categorias de serviço* que um provedor pode oferecer. Numa visão geral, uma categoria de serviço relaciona um conjunto de parâmetros. Em [68], algumas categorias genéricas de serviço são sugeridas, entre elas:

- *serviços multimídia*, para aqueles usuários que requerem que os dados de diversas mídias sejam eficientemente processados e comunicados no ambiente;
- *serviços seguros*, para aqueles usuários que requerem do ambiente garantias de proteção contra o acesso ou manipulação, intencional ou não, de seus dados por usuários ou componentes não autorizados;
- *serviços confiáveis*, para aqueles usuários que requerem do ambiente componentes altamente confiáveis e tolerantes a falhas.

Toda manipulação de parâmetros feita pelos mecanismos de provisão de QoS deve ocorrer no contexto da categoria desejada, que funciona como uma espécie de interface para a base de informações de QoS do provedor.

Categorias de serviço podem ser refinadas considerando-se propriedades relacionadas ao nível de visão de QoS do usuário e tipos de dados dele provenientes. Mais ainda, tais propriedades devem ser descritas por intermédio de derivações apropriadas de parâmetros. Assim, define-se categorias de serviço, de forma mais precisa, como conjuntos de parâmetros que possuem determinadas propriedades em comum, relacionadas ao tipo do ambiente, ao nível de visão de QoS e aos tipos de dados do usuário.

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

Pode-se refinar a categoria de serviço de vídeo em uma subcategoria de serviço de vídeo sem compressão e compactação e outra de serviço de vídeo com compressão. Nesse exemplo, uma propriedade que difere as duas categorias relaciona-se à natureza da carga gerada pelos dados. Assim, derivações do parâmetro vazão podem ser usadas na descrição dessas subcategorias. Para vídeos sem compressão e compactação, o parâmetro vazão máxima é suficiente, enquanto que para vídeos com compressão e compactação, os parâmetros vazão média e máxima variação da vazão, entre outros, deverão ser os mais utilizados.

4.4.2 Elementos do Framework de Parametrização

A Figura 4.6 ilustra as classes e os respectivos relacionamentos que constituem o Framework para Parametrização de Serviços.

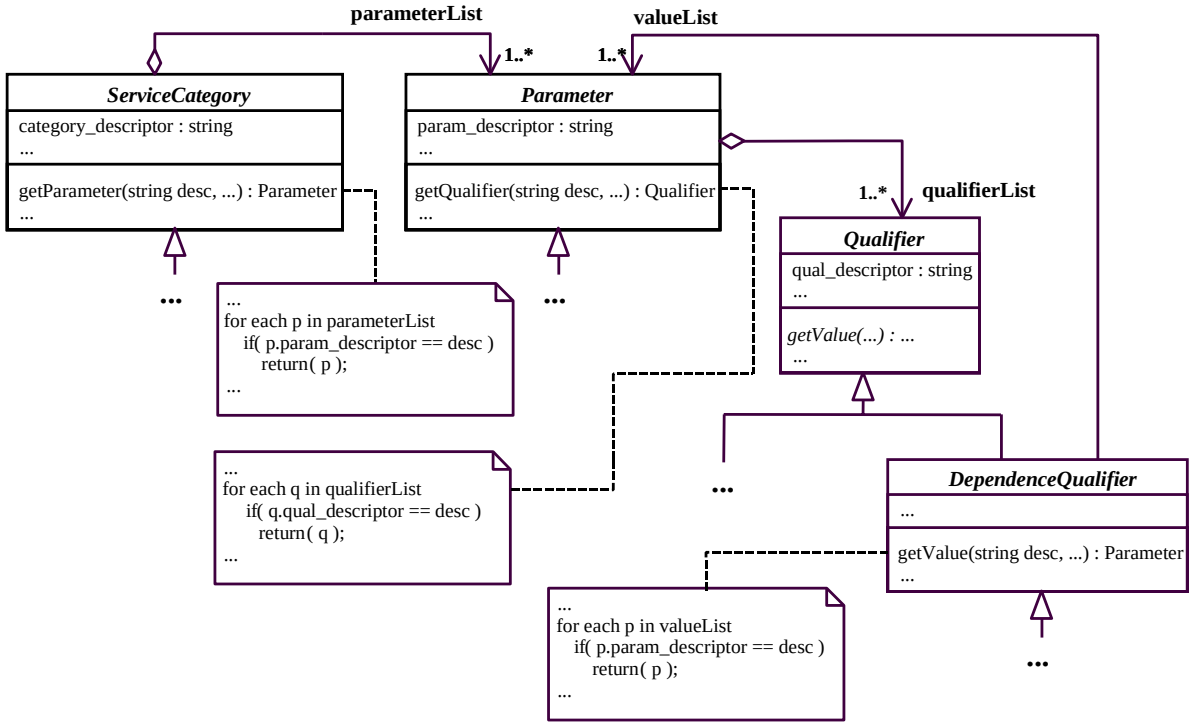


Figura 4.6: Framework para parametrização de serviços.

As classes *`ServiceCategory`* e *`Parameter`* simbolizam, abstratamente, as hierarquias de categorias de serviço e de derivação de parâmetros, respectivamente. O pattern estrutural Bridge [53] é utilizado para representar as categorias de serviço como conjuntos de parâmetros (através do relacionamento de agrupamento `parameterList`), possibilitando o desacoplamento entre as hierarquias de categorias de serviço e de derivação de parâmetros, tornando-as independentemente extensíveis. O desacoplamento é conseguido através do método `getParameter()` em *`ServiceCategory`*, que retorna uma instância de qualquer subclasse de *`Parameter`* com base no atributo `param_descriptor`.

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

4.4.3 Exemplo de Instanciação do Framework de Parametrização

A Figura apresenta um exemplo simplificado de instanciação do framework de parametrização.

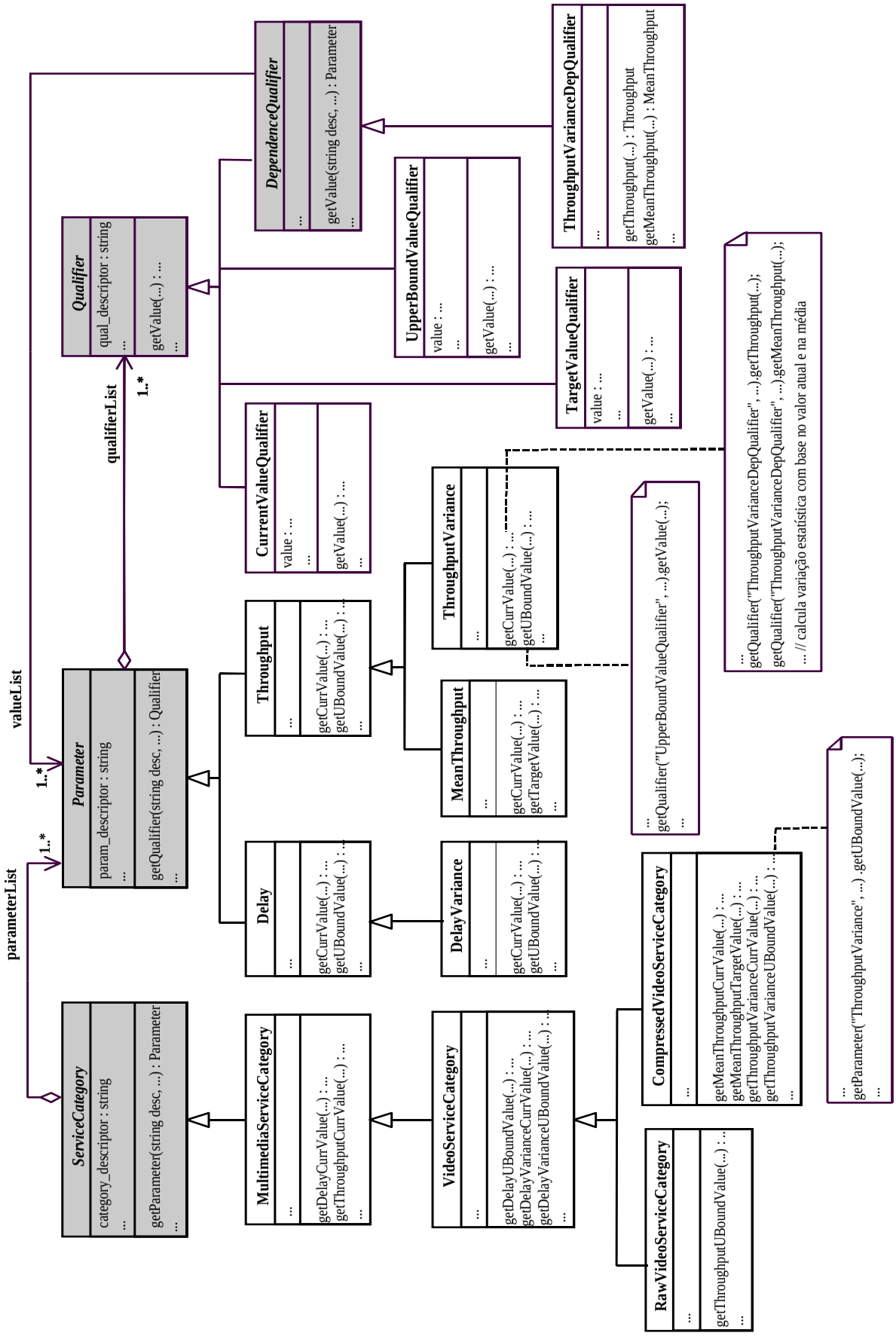


Figura 4.7: Exemplo de instanciação do framework de parametrização de serviços.

Subclasses de *Qualifier* são utilizadas para desempenhar o papel dos qualificadores que armazenam os valores aos quais as duplas de métodos de acesso *set()* e *get()* têm acesso. A classe *ThroughputVariance*, por exemplo, representa o parâmetro abstrato variação estatística da vazão. As duplas de métodos `setCurrValue()`, `getCurrValue()` e `setUboundValue()`, `getUboundValue()` nela presentes representam os parâmetros variação estatística (instantânea) da vazão e máxima variação da vazão, respectivamente. As classes `CurrentValueQualifier` e `UpperBoundValueQualifier` representam os qualificadores associados a essas duas duplas de métodos.

O desacoplamento entre *ServiceCategory* e *Parameter* permite, ainda no exemplo da Figura 4.7, que uma instância de `CompressedVideoServiceCategory`, por exemplo, associe-se dinamicamente a uma instância de uma classe que represente o parâmetro abstrato variação estatística da vazão e que tenha a mesma interface e semântica de `ThroughputVariance`, mas que esteja associada a uma estratégia de monitorização que calcule o valor da variação estatística da vazão de um modo diferente. A associação dinâmica entre uma categoria de serviço e representações distintas de um mesmo parâmetro é em geral muito útil, pois a definição de quais estratégias serão adotadas não só pelos mecanismos de monitorização, mas também pelos de mapeamento, influi diretamente na escolha de uma representação interna adequada para os parâmetros por eles manipulados. Neste sentido, os qualificadores de dependência funcional, modelados na pela classe *DependenceQualifier*, mostram-se muito importantes na tarefa de manter suficientemente flexíveis as estruturas de parâmetros para que as estratégias de monitorização e de mapeamento possam ser facilmente substituídas. Especificamente no exemplo da figura, o relacionamento de agrupamento `valueList` associado a *DependenceQualifier* permite que a estrutura de registro de medição da variação estatística da vazão, formada pelas classes `ThroughputVariance`, `ThroughputVarianceDepQualifier`, `Throughput` e `MeanThroughput`, possa ser dinamicamente substituída por outra que não envolva esta última classe. Esta mudança de estrutura pode ser necessária, por exemplo, se for adotada uma nova estratégia de monitorização que calcule a variação estatística da vazão adaptativamente, ao invés de utilizar valores previamente calculados para a vazão média.

4.5 Framework para Escalonamento de Recursos

O framework para escalonamento de recursos modela os mecanismos de escalonamento com base no conceito de árvores de recursos virtuais.

4.5.1 Elementos do Framework de Escalonamento

A Figura 4.8 apresenta as classes e os respectivos relacionamentos que o constituem o framework para escalonamento de recursos.

Cada *ResourceScheduler* tem uma lista de recursos virtuais a serem escalonados (representada pelo relacionamento `virtualResourceList`). Cada instância de *VirtualResource* pertencente a essa lista pode ser um novo escalonador (i.e., um `VirtualResourceScheduler,` compondo um novo nível da árvore de recursos) ou um nó terminal (um `ConcreteVR`). Cada instância de *ResourceScheduler* pode ser um `VirtualResourceScheduler` ou um `RealResourceScheduler` correspondendo aos casos em que o escalonador é responsável pelo compartilhamento da ca-

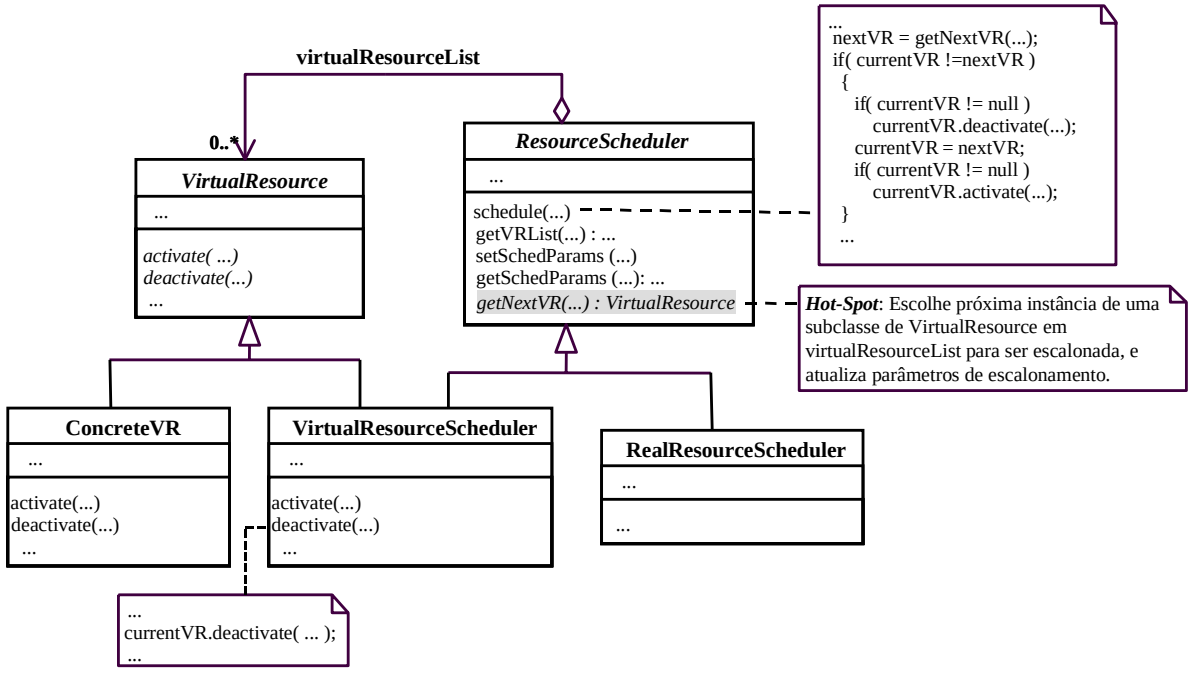


Figura 4.8: Framework para escalonamento de recursos.

pacidade de um recurso virtual (i.e., um recurso já escalonado de um outro) ou de um recurso real. Em outras palavras, a árvore de recursos virtuais tem na raiz uma instância da classe `RealResourceScheduler`, nos nós intermediários instâncias da classe `VirtualResourceScheduler` e, finalmente, nas folhas, instâncias da classe `ConcreteVR`.

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

Caso o recurso virtual filho sendo ativado ou desativado também seja um escalonador (de recurso virtual), os conceitos de ativação e desativação envolvem considerações adicionais. A desativação de um escalonador de recurso virtual provoca a desativação imediata de qualquer um de seus recursos virtuais filhos que por ventura esteja de posse da parcela de utilização do recurso real cedida pelo escalonador.

Quando `schedule()` é disparado, a escolha do próximo recurso virtual a utilizar um recurso real é delegada ao método *`getNextVR()`*, hot-spot da classe *`ResourceScheduler`*. Esse hot-spot representa, abstratamente, as estratégias de escalonamento, sendo suas instanciações responsáveis por implementá-las concretamente. Isto possibilita a flexibilidade dos mecanismos de escalonamento com relação às estratégias adotadas, ou até mesmo com relação às categorias de serviço a serem oferecidas.

Para que seja possível a um escalonador permitir que *`getNextVR()`* escolha adequadamente um de seus recursos virtuais filhos, a classe *`ResourceScheduler`* contém os seguin-

tes métodos:

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

Ambos os métodos deverão ser utilizados apenas pelas implementações de *`getNextVR()`*, sendo, portanto, métodos privativos da classe *`ResourceScheduler`*.

4.5.2 Exemplo de Instanciação do Framework de Escalonamento

A Figura 4.9 apresenta um caso de uso do framework para escalonamento de recursos, apresentado em [57], onde é sugerida uma arquitetura para o escalonamento de processadores em sistemas operacionais da família UNIX. Naquele trabalho, o pattern comportamental Strategy [53] foi utilizado para implementar o hot-spot *`getNextVR()`*.

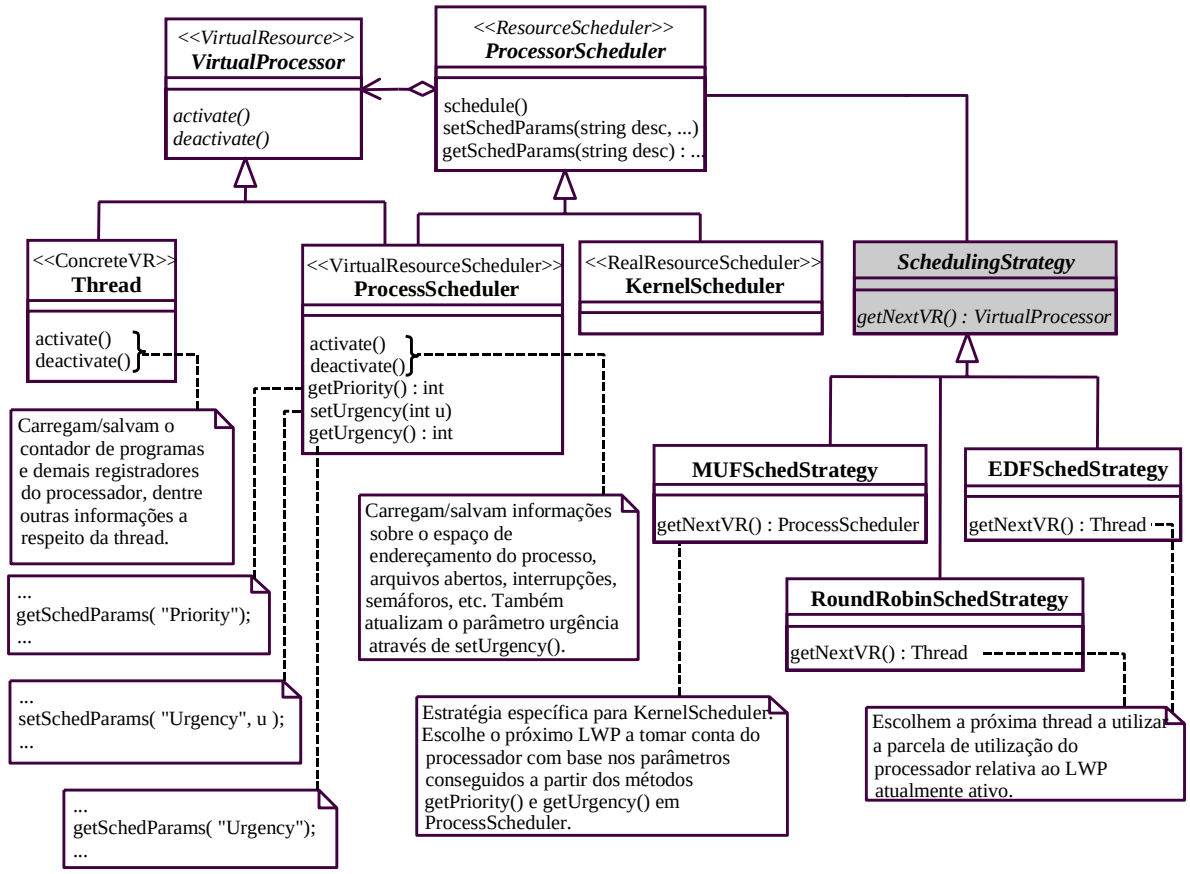


Figura 4.9: Exemplo de instanciação do framework de escalonamento.

Esta arquitetura, baseada no trabalho apresentado em [58], explora o conceito de processos leves (Lightweight Processes — LWPs) já existente nos sistemas operacionais Solaris 2.x[2], acrescentando-lhe algumas características desejáveis com relação ao suporte à provisão de QoS.

[2] O exemplo assume algumas simplificações do modelo de processos adotado por esses sistemas como, por exemplo, a existência de um único processador por estação.

Na raiz da árvore de recursos encontra-se o escalonador central (classe `KernelScheduler`), localizado no espaço de endereçamento do núcleo do sistema operacional, e que é responsável por escalonar o processador entre os LWPs dos vários processos. No nível superior da arquitetura, cada LWP age como um escalonador de processo (classe `ProcessScheduler`), responsabilizando-se por escalonar a sua parcela de utilização do processador entre threads (classe `Thread`) que se encontram no espaço de endereçamento do processo que o detém. Essa configuração pode também ser visualizada na Figura 4.10.

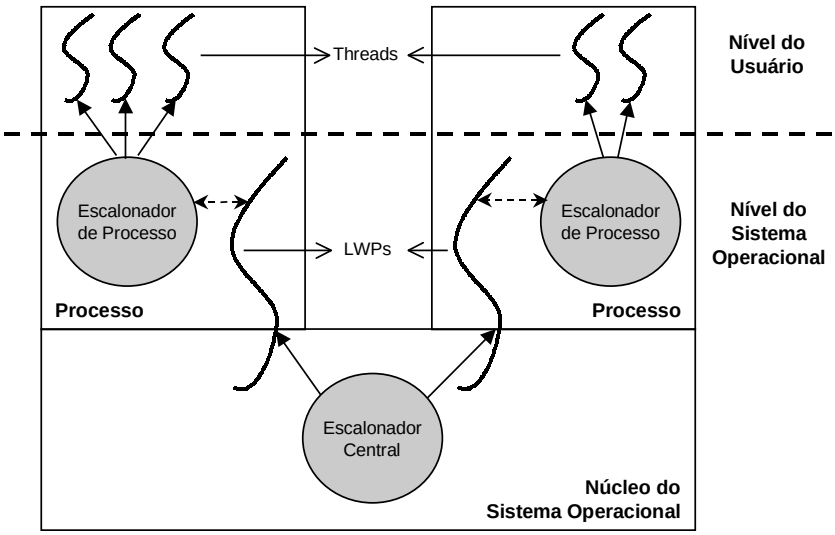


Figura 4.10: Arquitetura para escalonamento de processadores.

Nessa arquitetura, define-se, para o escalonador central, uma estratégia de escalonamento específica, denominada de *Most Urgent First (MUF)*, representada pela classe `MUFSchedStrategy`. A grosso modo, esta estratégia define que o LWP a possuir o processador será sempre aquele de maior prioridade, dentre todos os LWPs que possuem threads ativáveis no momento. Em caso de serem escolhidos dois ou mais LWPs, o que estiver oferecendo suporte à thread mais urgente, dentre todas as threads controladas por estes LWPs, será o escolhido. O conceito de urgência depende da estratégia de escalonamento adotada pelos escalonadores de processo. Por exemplo, na estratégia EDF, o que define a urgência das threads é o deadline das mesmas.

4.6 Framework para Estabelecimento de Contrato de Serviço

O estabelecimento do contrato de serviço é o responsável, em última instância, pela criação de um recurso virtual: o MediaPipe entre entre os elementos que serão os participantes do serviço de comunicação principal. O controle de admissão e a negociação estão envolvidos nesse processo. A criação desse MediaPipe (ou seja, o estabelecimento do contrato de serviço) é efetuada através da criação dos MediaPipes nos provedores internos adequados, incluindo os provedores de acesso e de infra-estrutura. Em alguns provedores, o estabelecimento de contrato é similar à sinalização para estabelecimento de conexões.

4.6.1 Elementos do Framework para Estabelecimento de Contrato de Serviço

A Figura 4.11 contém as classes e os respectivos relacionamentos que constituem o framework para estabelecimento do contrato de serviço. O estabelecimento começa quando requisita-

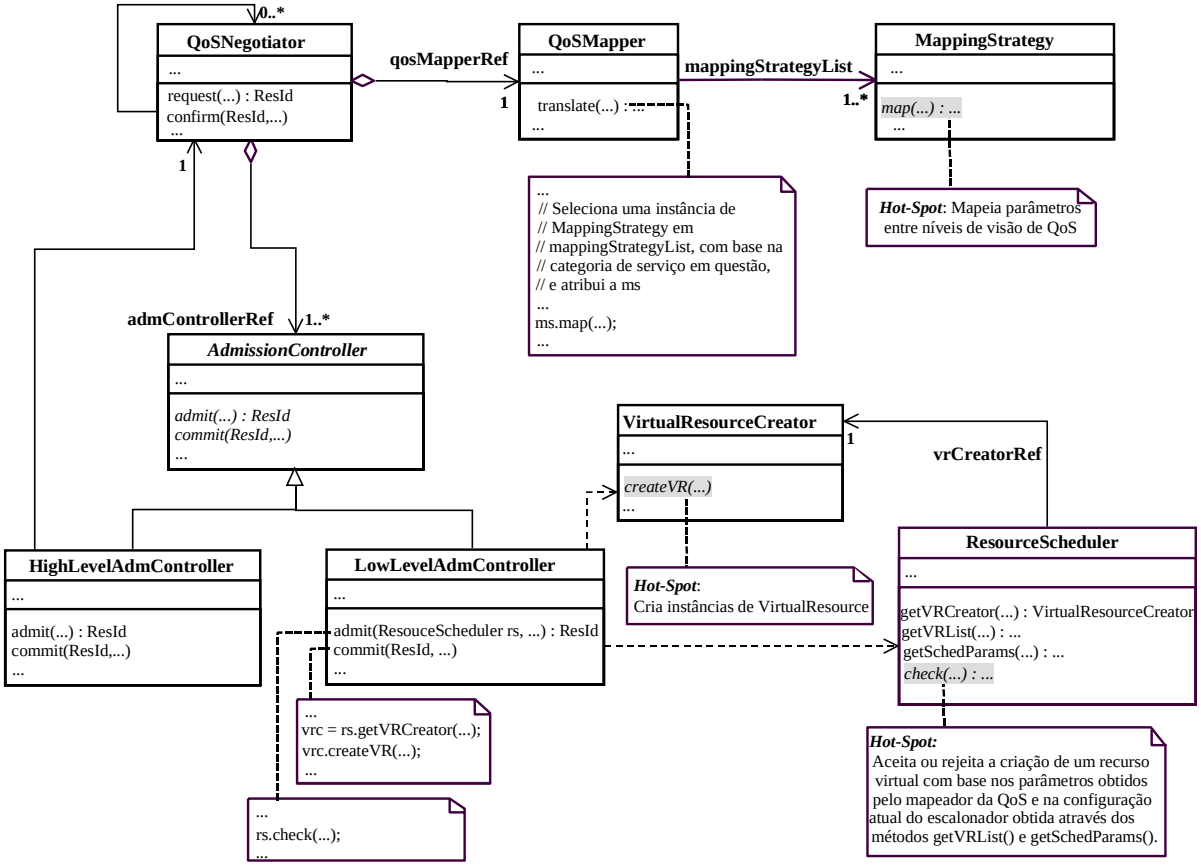


Figura 4.11: Framework para estabelecimento de contratos de serviço.

se, por intermédio do método `admit()` (do *`AdmissionController`*), a admissão de um novo contrato. O pattern estrutural *Facade* [53] é utilizado para mascarar, sob uma interface única de requisição de serviços (provida pelos métodos `admit()` e `commit()` em *`AdmissionController`*), toda a complexidade associada ao estabelecimento de contratos de serviço. Em todos os casos de uso possíveis do framework de estabelecimento, 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.

Caso o provedor seja formado por vários provedores internos, o seu *`AdmissionController`* será uma instância de um `HighLevelAdmController`, cuja função é acionar o mecanismo de negociação (seja ele centralizado ou distribuído) através do método `QoSNegotiator::request()`. O mecanismo de negociação irá, por sua vez, identificar todos os provedores internos que podem se envolver no fornecimento do serviço requisitado, e associar a cada um deles uma parcela da responsabilidade pela provisão da QoS especificada na requisição. Este processo é efetuado, em parte, pelo mecanismo de *mapeamento da QoS*, responsável por traduzir a requisição em parâmetros conhecidos internamente ao provedor, em parte pelo *mecanismo de roteamento* (se for o caso), que escolhe os subambientes adequados

baseando-se em alguma informação de alcançabilidade dos componentes envolvidos, e em parte por uma heurística que define, quantitativamente, as parcelas de responsabilidade de cada subambiente escolhido.

As funções de roteamento e a heurística mencionada são funções internas da classe `QoSNegotiator` e não foram modeladas. O relacionamento recursivo em `QoSNegotiator` indica a possibilidade do mecanismo de negociação ser implementado de modo centralizado ou distribuído. O mapeamento da QoS é delegado a instâncias da classe `QoSMapper`. Essa classe modela os mapeadores da QoS que, em conjunto, definem os mecanismos de mapeamento. Cabe a estes componentes traduzir as requisições de serviço feitas no nível mais alto de abstração de um provedor em requisições de serviço condizentes com os níveis de visão de QoS dos subambientes envolvidos no fornecimento do serviço. A Figura 4.12 ilustra a utilização desses componentes no caso da sinalização em uma rede ATM.

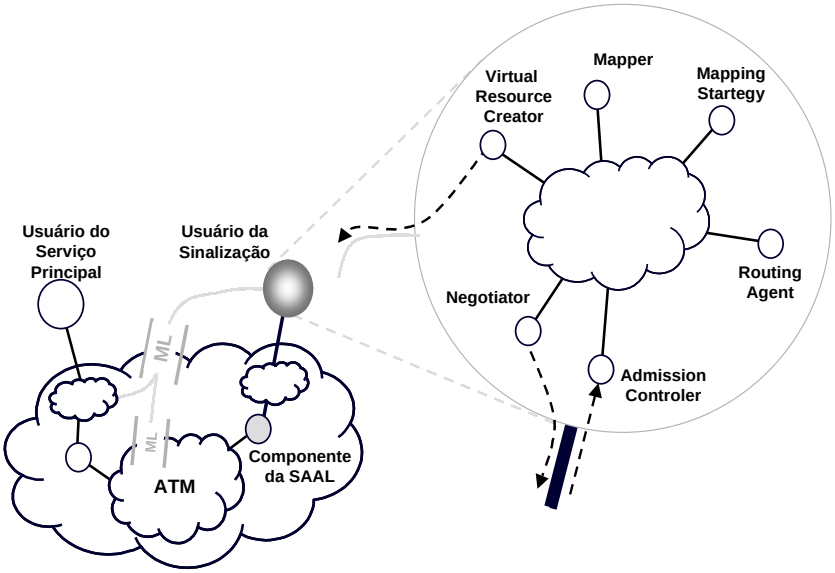


Figura 4.12: Componentes do framework de estabelecimento de contratos de serviço aplicados para o caso da sinalização ATM.

O processo de tradução de requisições de serviço é efetuado mediante o disparo do método `translate()` em um mapeador da QoS. A função deste método consiste basicamente em mapear parâmetros de caracterização de carga e de especificação da QoS entre níveis distintos de visão de QoS. Para possibilitar a flexibilidade dos mapeadores de QoS quanto às categorias de serviço oferecidas por um provedor nos seus vários níveis de abstração, o mapeamento de parâmetros é realizado pela implementação do método *`map()`*. O método *`map()`* é um dos hot-spots presentes no framework de estabelecimento de contrato de serviço.

Após efetuadas as escolhas dos provedores internos, componentes internos (intermediários) envolvidos e das respectivas parcelas com os parâmetros mapeados, uma nova requisição de estabelecimento de contrato é feita a cada um desses provedores. O processo então se repete até chegar-se ao nível em que o mecanismo de controle de admissão pode efetuar testes diretamente sobre um recurso virtual sem repassá-los a nenhum provedor interno. Nesse caso, o *`AdmissionController`* será uma instância de um `LowLevelAdmController`, na qual o método `admit()` cuidará de verificar a disponibilidade de criação do novo recurso virtual diretamente na árvore de recursos virtuais utilizando

o método *check()* do ResourceScheduler. Esse método, também um hot-spot definido pela classe ResourceScheduler, deverá verificar a viabilidade de que um novo recurso virtual obtenha a parcela desejada de utilização de um recurso, sem interferir em outros recursos virtuais já criados na árvore associada ao recurso. Se o teste ratificar essa disponibilidade, o controle de admissão retorna uma resposta afirmativa e devolve uma identificação de uma *reserva* que, posteriormente, poderá ser utilizada para efetivar a alocação dos recursos (vide retorno de ResId nos métodos request() e admit()na Figura 4.11). No nível do provedor externo, a negociação receberá de volta todas as respostas de todos os seus provedores internos. Se todas forem positivas, esse controle de admissão também pode retornar ao seu provedor externo uma resposta positiva. Se, em algum dos níveis, um ou mais dos provedores internos não oferecer a disponibilidade desejada, a negociação tem ainda a possibilidade de tentar alterar o balanceamento gerado pela heurística daquele nível ou alterar a escolha dos provedores envolvidos (por meio de um novo roteamento, p. ex.), resubmetendo as novas parcelas aos respectivos controle de admissão.

Se, ao final de todo o processo, o provedor apresentar os recursos necessários, ele admitirá o fluxo do usuário e fará a alocação dos recursos necessários para servi-lo (levando possivelmente a uma mudança de estado interno), de tal forma que a QoS especificada seja satisfeita. Para efetivar essa alocação dos recursos reservados, o usuário invoca o método commit(), utilizando como parâmetro a identificação da reserva anteriormente realizada. A seqüência de invocações de métodos, ilustrada na Figura 4.13, exemplifica todo o processo de admissão e criação de recursos virtuais em um subambiente específico.

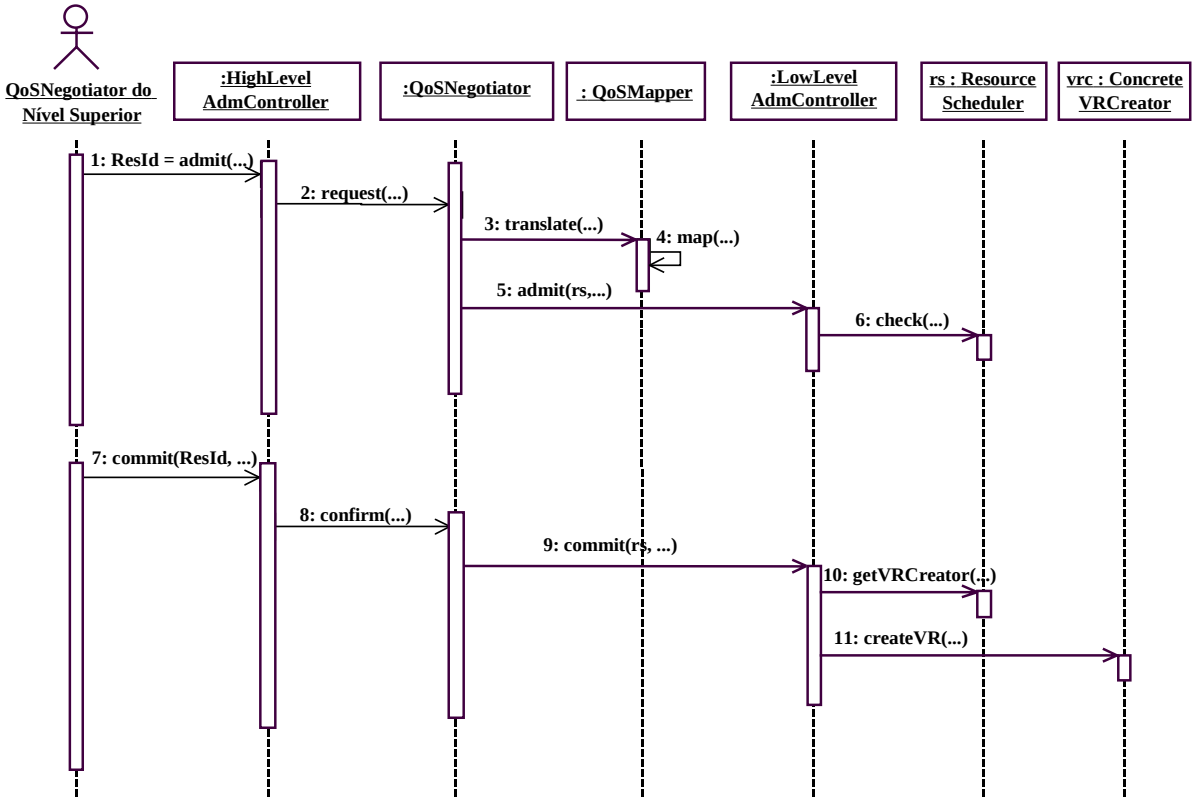


Figura 4.13: Seqüência de admissão e criação de recursos virtuais.

O método commit() em *AdmissionController* é o responsável por iniciar o pro-

cesso de criação de recursos virtuais, através do disparo do método `createVR()` no componente de criação ligado ao escalonador escolhido durante a admissão. Maiores detalhes sobre a criação de recursos virtuais podem ser encontrados em [57, 56], bem como um exemplo completo de utilização do framework de estabelecimento de contrato de serviço em protocolos de comunicação fim-a-fim.

4.7 Framework para Controle e Gerência da QoS

A modelagem dos mecanismos envolvidos no controle e gerência da QoS será, sob vários aspectos, semelhante àquela dos mecanismos de estabelecimento. Essa semelhança é conseqüência do fato de que ambas baseiam-se na estrutura recursiva de recursos virtuais modelados pelas respectivas árvores. Enquanto que no estabelecimento os provedores eram recursivamente percorridos em busca de se determinar a disponibilidade e posteriormente de se efetuar a alocação dos recursos, no controle e gerência, a determinação das violações e as atitudes cabíveis serão efetuadas com base nos recursos virtuais e contratos estabelecidos, sendo portanto estruturados também de maneira recursiva e nas mesmas bases.

Dentre os mecanismos responsáveis pelo controle e gerência da QoS estão incluídos os de de *monitorização de fluxos*, que policiam os recursos virtuais aos quais o fluxo do usuário tem acesso, e os de *sintonização da QoS*, responsáveis por efetuar adaptações nos contratos de pipes internos componentes de um MediaPipe, sem que haja necessidade de interrupção (i.e., renegociação) do fornecimento do serviço.

A monitorização pode ser efetuada, por exemplo, através da geração periódica de medições relativas à real QoS sendo oferecida pelos recursos virtuais. Tais medições são obtidas a partir de parâmetros de desempenho do provedor relacionados (direta ou indiretamente) à carga gerada pelos recursos virtuais. Um exemplo típico de monitorização é a medição do retardo de trânsito médio dos dados de um fluxo em uma rede de distribuição. Através de marcas de tempo (timestamps) colocados nos pacotes referentes a esse fluxo, o mecanismo de monitorização consegue estimar o retardo de um pacote isolado, e, a partir daí, gerar medições para o retardo de trânsito médio dos dados do fluxo. Esse retardo está relacionado, entre outras coisas, ao modo como os vários enlaces presentes na rede de distribuição estão sendo escalonados entre os pacotes desse fluxo.

O mecanismo de monitorização de um fluxo é também responsável por enviar alertas ao mecanismo de sintonização, que, ao recebê-los, reavalia as medições feitas pelo mecanismo de monitorização. Dependendo da política de provisão de QoS adotada, o mecanismo de sintonização pode agir somente para remediar a ocorrência de uma violação de contrato, ou então para prevenir a mesma, quando há risco iminente do contrato ser violado. A escolha do tipo de ação a ser tomada pelo mecanismo de sintonização está sujeita também ao grau de depreciação da QoS anteriormente negociada. Dentre as ações mais comuns, estão:

- o repasse do alerta, juntamente com as medições associadas, ao provedor de nível superior (ou ao usuário final, se for o caso), o que implica na tradução, através do mecanismo de monitorização, dessas medições em informações condizentes com o nível de visão de QoS do usuário;

- a reconfiguração de um ou mais escalonadores envolvidos no fornecimento do serviço. Na maioria dos casos, uma violação de contrato está relacionada ao fato de que um escalonador não consegue mais oferecer a um recurso virtual envolvido no fornecimento do

serviço a sua parcela de utilização do recurso real associado. Quando o fluxo do usuário tem acesso a mais de um recurso real, a reconfiguração de escalonadores permite que, dado que um recurso virtual falhe em contribuir com a sua parcela de responsabilidade pela provisão da QoS, outros recursos virtuais tenham suas parcelas de utilização de recursos redimensionadas, de modo que a QoS fim-a-fim, conforme observada pelo usuário, não seja afetada;

- a escolha de recursos alternativos, quando a reconfiguração de escalonadores se mostra insuficiente. Neste caso, ocorre uma espécie de renegociação sem intervenção do usuário, podendo, inclusive, haver a participação de outros mecanismos como roteamento;
- o acionamento do mecanismo de negociação, para que este renegocie com o usuário um novo contrato de serviço factível de ser satisfeito.

4.7.1 Elementos do Framework de Controle e Gerência

As classes e os respectivos relacionamentos que constituem o framework para controle da QoS encontram-se representados na Figura 4.14. Sua estrutura geral é bem semelhante àquela apre-

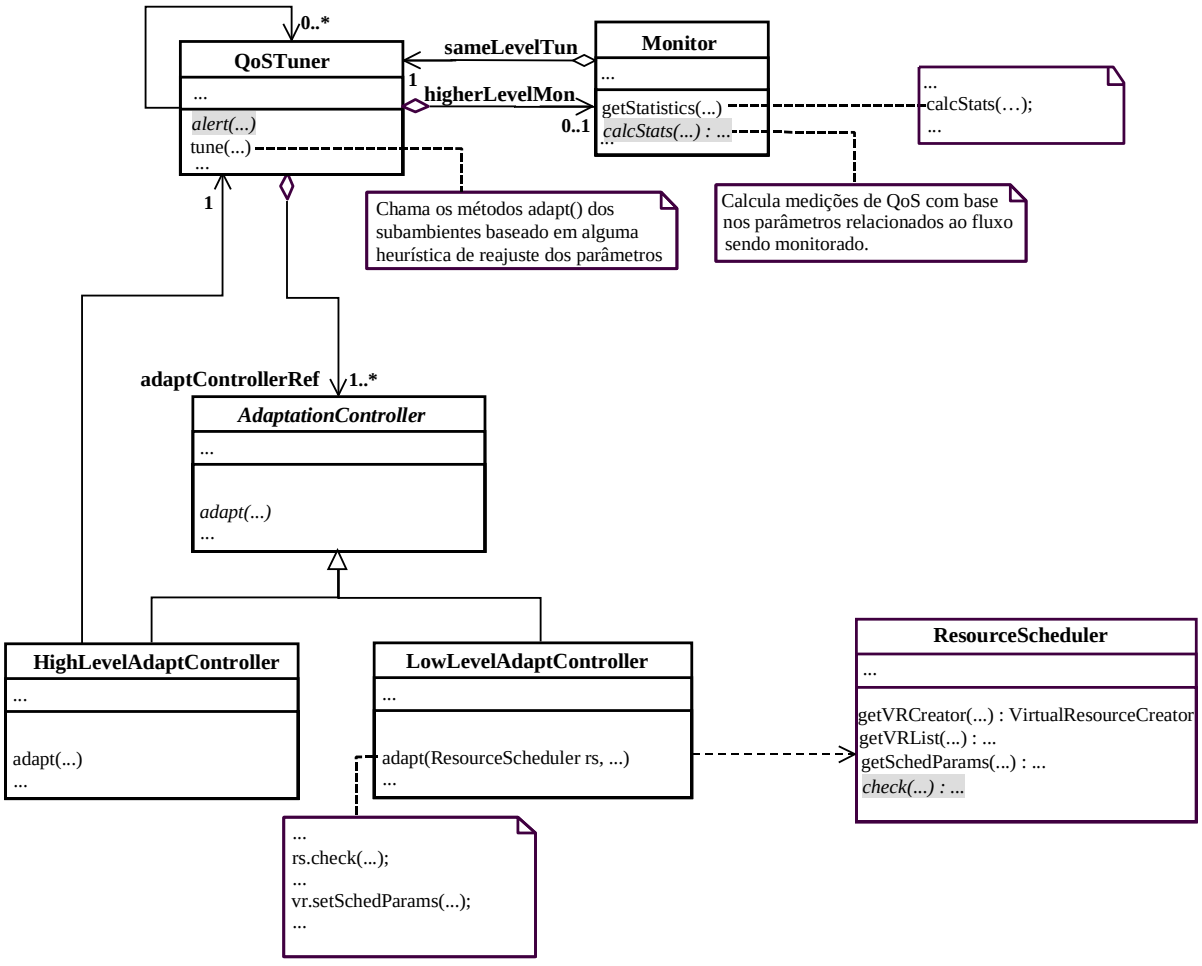


Figura 4.14: Framework para controle e gerência da QoS.

sentada para o framework de estabelecimento (vide Figura 4.11).

Os mecanismos de controle e gerência sempre são iniciados a partir da geração de um evento qualquer no ambiente que ocasione o disparo do método `getStatistics()` em alguma instância da classe `Monitor`. Esses componentes, em conjunto, definem os mecanismos de monitorização. O método `getStatistics()` é responsável por obter as medições relativas à real QoS sendo oferecida aos usuários, e por enviar alertas ao mecanismo de sintonização. O cálculo, em si, das medições de QoS de um fluxo, é um hot-spot definido pelo método *`calcStats()`*. Já o envio de alertas ao mecanismo de sintonização é feito por meio do disparo de `tune()` na instância correspondente de `QoSTuner`. Assim como `QoSNegotiator` (vide Figura 4.11), `QoSTuner` possui um relacionamento recursivo que permite a implementação do mecanismo de sintonização de modo centralizado ou distribuído.

Um exemplo da dinâmica envolvida nos mecanismos desse framework pode ser extraído da Figura 4.15.

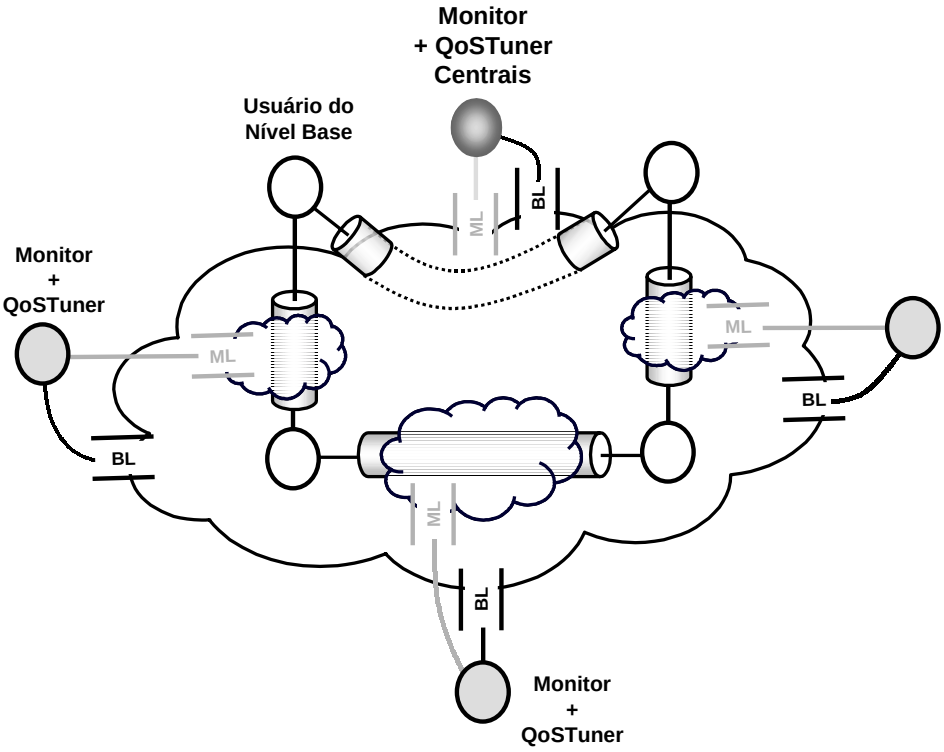


Figura 4.15: Exemplo de configuração centralizada para controle e gerência da QoS.

Nessa figura, representou-se uma configuração centralizada para monitores e sintonizadores. Por simplicidade, cada par de instâncias `QoSTuner` e `Monitor` é representado conjuntamente. Sempre que um dos monitores internos percebe uma violação do contrato no seu provedor (monitorado utilizando a interface de nível meta), ele avisa a seu sintonizador parceiro (um `QoSTuner`). Este tentará reajustar os parâmetros ou o balanceamento internamente ao seu provedor. Caso isso não seja possível, esse `QoSTuner` passará um novo sinal de alerta (representado na Figura 4.14 por `alert()`) para o monitor do nível do provedor externo (comunicando-se com ele através da interface de nível base de um provedor de infra-estrutura). Assim, o processo se repetirá para o nível do provedor externo. Essa seqüência de ações pode ser visualizada na Figura 4.16.

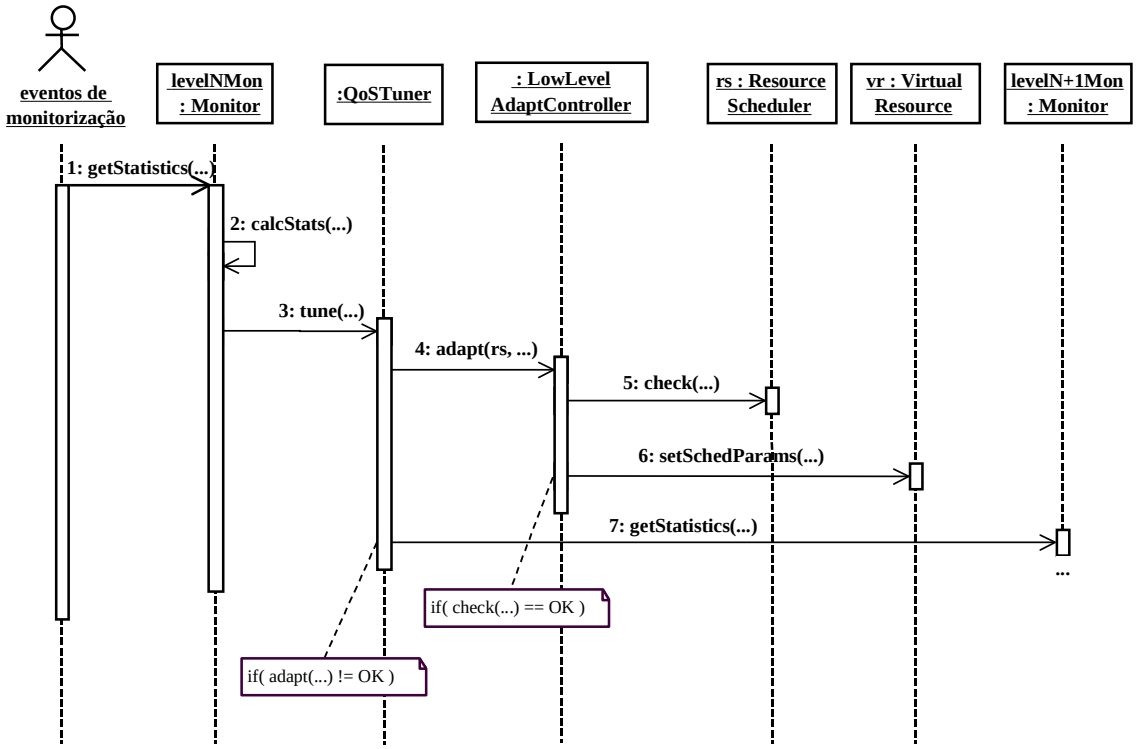


Figura 4.16: Seqüência de controle e gerência da QoS.

4.8 Cenário de Utilização do Framework

Visando testar o framework em pelo menos algumas das situações possíveis, vem sendo desenvolvido, no Laboratório TeleMídia da PUC-Rio, um protótipo de um sistema de transmissão de mídias contínuas. O ambiente de implementação utilizado consta de (i) duas estações SPARC Ultra 1 e uma SPARC Station 5, todas com sistema operacional Solaris 2.5 e placas ATM Efficient 155 Mbps, e (ii) um comutador ATM IBM 8260 interligando as três estações. A Figura 4.17 ilustra, de modo simplificado, a arquitetura do sistema. Os seguintes elementos do sistema (hachurados na figura) correspondem a utilizações distintas do framework: (i) a biblioteca de escalonamento de threads, (ii) o controlador de processos, e (iii) o protocolo de negociação da QoS.

4.8.1 Biblioteca de Escalonamento de Threads

A biblioteca de escalonamento de threads permite a uma aplicação Java especificar a QoS de processamento de suas threads, e oferece mecanismos de escalonamento que permitem a provisão da QoS desejada. A biblioteca permite também que o desenvolvedor de uma aplicação se valha de várias estratégias concorrentes de escalonamento de threads na mesma aplicação, e que defina conjuntos adequados de parâmetros que descrevam a QoS das threads da aplicação, segundo as estratégias definidas.

O framework para parametrização de serviços é aplicado diretamente na estruturação de parâmetros da biblioteca. Já o mecanismo de escalonamento no qual se baseia a biblioteca corresponde à utilização da estrutura definida no framework de escalonamento de recursos.

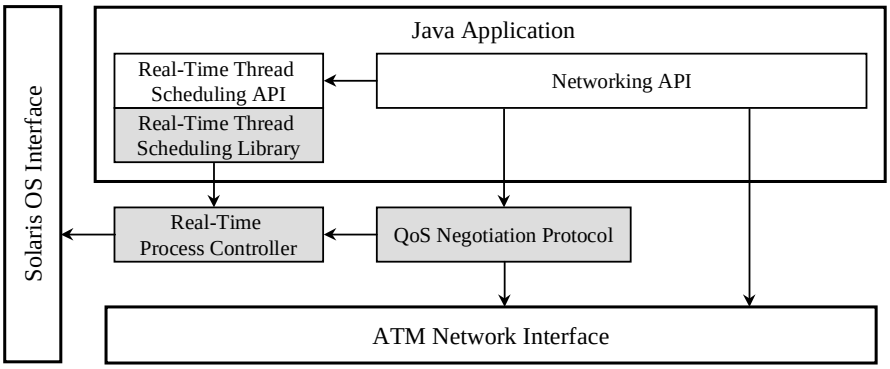


Figura 4.17: Arquitetura do sistema de transmissão de vídeo.

Este esquema de escalonamento, que estende o mecanismo de escalonamento de threads em Java [96], permite o escalonamento hierárquico de threads de maneira simples e flexível.

4.8.2 Controlador de Processos

Para uma aplicação que requer garantias maiores de processamento, a biblioteca de escalonamento de threads é insuficiente. Isto porque a plataforma Solaris vale-se de um esquema próprio de escalonamento hierárquico, onde em um primeiro nível encontram-se os processos leves (LWPs), associados às aplicações, e em um segundo nível encontram-se as threads, que são escalonadas pelos LWPs, que por sua vez são escalonados pelo sistema. Para que se possa garantir efetivamente o processamento de um conjunto de threads segundo os requisitos por ela impostos, é necessário, primeiramente, que as LWPs associadas a esse conjunto de threads também tenham garantias de escalonamento.

O controlador de processos foi implementado com o objetivo de permitir a gerência de escalonamento de LWPs. Esta gerência é feita de modo semelhante a da biblioteca de escalonamento de threads, correspondendo também à utilização da estrutura definida no framework de escalonamento. Este esquema de escalonamento, que estende o mecanismo de escalonamento de processos proposto em [22], permite o escalonamento hierárquico de LWPs. Em conjunto com a biblioteca de escalonamento de threads, o controlador de processos oferece às aplicações Java garantias efetivas de escalonamento de suas threads.

4.8.3 Protocolo de Negociação da QoS

O protocolo de negociação da QoS, em sua versão inicial, provê o mapeamento de parâmetros relativos a uma categoria genérica de serviços multimídia em parâmetros de contrato de tráfego utilizados nas redes ATM. O protocolo inclui também um esquema primitivo de negociação que, assim como o mapeamento, é definido com base no framework para estabelecimento de contrato de serviço.

4.9 Trabalhos Relacionados

Até recentemente, boa parte do desenvolvimento relacionado ao suporte à provisão de QoS ocorreu no contexto de subsistemas isolados, abrangendo, em geral, partes específicas das ar-

quiteturas de protocolos de comunicação e de sistemas operacionais.

4.9.1 Sistemas Operacionais

Na área de sistemas operacionais, a maior parte dos trabalhos relacionados à provisão de QoS têm focalizado o escalonamento de processadores. Os sistemas operacionais tradicionais adotam estratégias de escalonamento fixas, normalmente baseadas em prioridades definidas pelas aplicações [122, 52]. Entretanto, requisitos de processamento podem variar de aplicação para aplicação, e descrever esses requisitos somente por intermédio de prioridades inviabiliza uma provisão adequada de QoS. Por exemplo, threads periódicas requerem que o parâmetro período seja disponibilizado pelo sistema operacional. A adoção de parâmetros mais específicos permite que estratégias adequadas de escalonamento possam ser utilizadas pelo sistema operacional. Como exemplo, a estratégia Earliest Deadline First (EDF) [83] vale-se do período para escalonar threads periódicas corretamente.

A necessidade de dar suporte ao processamento integrado de diferentes mídias requer que o sistema operacional permita o uso de diferentes conjuntos de parâmetros pelas aplicações, e conseqüentemente, que diferentes estratégias de escalonamento possam atuar concorrentemente sobre um mesmo processador. Neste sentido, uma abordagem interessante é a do particionamento hierárquico da capacidade do processador, utilizado em trabalhos como o Hierarchical CPU Scheduler [59] e o CPU Inheritance Scheduling [50]. Esta abordagem permite a um sistema operacional particionar a capacidade do processador entre diferentes categorias de processamento, através de uma determinada estratégia de escalonamento. Estas categorias, por sua vez, podem ter suas parcelas da capacidade do processador particionadas (possivelmente usando uma estratégia diferente) entre subcategorias, de modo recursivo. A abordagem de particionamento hierárquico é suficientemente flexível para permitir a introdução de novas estratégias de escalonamento. Devido a essa característica, no presente trabalho a abordagem de particionamento hierárquico é generalizada para o caso do compartilhamento de recursos quaisquer.

4.9.2 Protocolos de Comunicação

Na parte de protocolos de comunicação, os trabalhos relacionados à provisão de QoS têm seguido, em geral, na direção da formulação de protocolos que envolvam a negociação da QoS e a reserva de recursos. A característica desses protocolos varia de acordo com o nível de operação dos mesmos, que pode ser: (i) fim-a-fim, como os protocolos RTP [121] e OSI95 [68], e as camadas de adaptação (AALs) definidas em redes com tecnologia ATM, ou (ii) operar em elementos intermediários, como os protocolos RSVP [43] e ST-II [131].

A definição de protocolos fim-a-fim é normalmente dependente do suporte e da flexibilidade oferecidos pela infra-estrutura de transmissão. Por exemplo, uma infra-estrutura baseada na tecnologia ATM deverá permitir uma gama maior de serviços de transmissão e um transporte mais eficiente das diferentes mídias do que outra baseada em tecnologias tradicionais de comutação de pacotes. A definição das diferentes AALs e os serviços específicos que sobre elas têm sido especificados correspondem a protocolos fim-a-fim que fazem uso da flexibilidade oferecida pela tecnologia ATM. Apesar disso, essa flexibilidade é limitada por um número finito de classes de serviço modeladas pelas AALs. O presente trabalho segue uma linha mais flexível, no sentido de que a definição do framework fornece uma capacidade de extensão que pode ser utilizada para configuração de protocolos voltados especificamente ao serviço desejado, sem a

necessidade da definição de um conjunto previamente estabelecido de classes ou parâmetros que caracterizem o serviço.

Muitos trabalhos também têm investido na pesquisa de novos modelos de caracterização de tráfego (como o D-BIND [76]) e de especificação da QoS (como os definidos em [64, 134, 12]). Em particular, a questão do mapeamento de parâmetros de QoS entre camadas, tratada em trabalhos como [76], é um problema que ainda necessita de maior aprofundamento. O mapeamento está também ligado à questão da orquestração, que é tratada de modo mais abrangente nos trabalhos ligados às arquiteturas de QoS.

4.9.3 Arquiteturas de QoS

Tratar a questão da QoS nos diferentes subsistemas isoladamente potencializa a geração de sistemas incompletos, pouco integrados e muito específicos [4]. Por esta razão, vários grupos de trabalho (como os de Illinois [94] e Lancaster [15]) têm proposto novas abordagens de arquitetura de sistema como solução para o problema da provisão de QoS fim-a-fim, sendo coletivamente denominadas de arquiteturas de QoS [14].

Independentemente da abordagem tomada pelas diferentes arquiteturas de QoS, o que elas tem basicamente em comum é a definição de conjuntos de interfaces que formalizam o conceito de QoS em diferentes subsistemas. Esta formalização é semelhante a que é feita pelo OSI-RM com relação ao conceito de serviço. Interfaces são em geral apresentadas sob a forma de frameworks, linguagens de programação ou bibliotecas desenvolvidos sobre subsistemas com características de QoS predefinidas. Com esta abordagem, as arquiteturas de QoS permitem a um projetista de sistemas integrar os mecanismos de provisão de QoS presentes nos diferentes subsistemas.

4.9.4 Modelos de Objetos com QoS

A maior limitação da abordagem adotada pelas arquiteturas de QoS é que a uniformização da provisão da QoS ocorre somente na fronteira entre os subsistemas. A definição da arquitetura interna dos subsistemas fica totalmente a cargo dos projetistas de sistema, o que é indesejável por vários motivos.

Um dos problemas das arquiteturas de QoS relacionado à ausência de uma arquitetura interna é que a orquestração dos recursos, tanto interna a um subsistema quanto entre subsistemas, não é modelada. Isto dificulta a tarefa do projetista em realizar uma provisão de QoS verdadeiramente fim-a-fim.

Outro obstáculo que um projetista pode encontrar, ao se valer de uma arquitetura de QoS para a implementação de um sistema multimídia, é a diversidade de requisitos existentes. As arquiteturas de QoS sugerem, intrinsecamente, o desenvolvimento de conjuntos finitos de mecanismos para provisão de QoS, em cada subsistema, que atendam a todos os possíveis requisitos de processamento e comunicação, o que torna tais subsistemas muito complexos e pouco flexíveis. Além disso, a ausência de uma arquitetura interna impede que determinados mecanismos para provisão de QoS potencialmente presentes em vários sistemas distintos (ou mesmo em subsistemas de um mesmo sistema) possam ser facilmente reutilizados.

Outra abordagem para o problema da provisão de QoS que tem sido considerada em recentes trabalhos, como o Modelo Binding [80] e o Modelo de Referência Unificado [24], é a do oferecimento de um único serviço configurável de acordo com a necessidade dos usuários. Tais trabalhos, em grande parte baseados no RM-ODP [69] e no CORBA [97], caracterizam-se pela

definição de ambientes genéricos de processamento e comunicação que permitem a implementação de sistemas a partir da conexão entre componentes OO reutilizáveis.

4.10 Conclusões

Os tópicos relacionados à provisão de QoS em sistemas multimídia têm sido, em sua maioria, parcialmente tratados na literatura. A maioria das abordagens apresenta pelo menos uma das seguintes características: (i) elas são restritas a subsistemas específicos, notadamente os sistemas operacionais e de comunicação; (ii) elas tratam a questão da integração da QoS (incluindo aí a orquestração de recursos) sob o ponto de vista de interfaces somente; e (iii) elas não facilitam a introdução de novos serviços. Neste capítulo, foi apresentada uma abordagem que permite o tratamento da questão da QoS sob uma visão mais abrangente, que age desde a interface com o usuário humano até a definição de mecanismos de provisão de QoS incluindo componentes das aplicações e dos protocolos de comunicação. A abordagem proposta consiste na definição de um framework cujo domínio de aplicação é suficientemente genérico para incluir quaisquer ambientes de processamento e comunicação, desde subsistemas específicos até plataformas para sistemas distribuídos. Com estas características, o framework possibilita (i) a construção de sistemas configuráveis no que se refere à introdução de novos serviços, e (ii) facilita a implementação dos mecanismos de compartilhamento e orquestração.

Capítulo 5

Framework para Comunicação de Grupo

RECONHECENDO que a maioria das soluções que têm sido apresentadas como candidatas à implementação do serviço de multicast tem sido projetadas tendo em mente determinadas condições específicas de infra-estrutura para distribuição de mensagens, ou características do serviço às quais são destinadas, ou ainda, a forma de gerenciamento dos grupos de usuários, este capítulo apresenta a proposta de um framework para a definição de um serviço de multicast cuja principal característica é a sua generalidade em relação a esses aspectos. Trata-se de uma revisão do trabalho apresentado em [112, 111].

5.1 Introdução

Um serviço de multicast pode ser definido como um conjunto de procedimentos e interfaces que permitem enviar mensagens para um grupo de participantes em um ambiente de processamento e comunicação. A definição de um serviço multicast pode ser realizada de forma independente da implementação da infra-estrutura de distribuição de mensagens e das aplicações específicas às quais o serviço é destinado. Um serviço de multicast provê uma forma primitiva de comunicação de grupo que pode ser utilizado como base para o oferecimento de serviços mais complexos, destinados a diversas áreas específicas como trabalho cooperativo, por exemplo.

A arquitetura genérica de um serviço de multicast pode ser dividida em duas partes: o *gerenciamento de grupo* e a *construção de uma infra-estrutura de distribuição*. Um grupo é definido como um subconjunto de usuários para o qual é possível a transmissão de mensagens, estando associado a um endereço de multicast [36]. O gerenciamento de grupo diz respeito a todas as ações relacionadas a composição do grupo, como manipulação de informações sobre os seus participantes e o controle sobre a entrada e saída de participantes ao grupo. A construção da infra-estrutura de distribuição está relacionada à forma de coordenação de recursos de processamento e comunicação para que a distribuição das mensagens possa ser efetuada. Em geral, a construção dessa infra-estrutura é efetuada de forma a tentar minimizar as replicações de mensagens desnecessárias e aumentar o desempenho dos recursos. Protocolos de roteamento são, em geral, responsáveis por grande parte desse trabalho no que concerne o sistema de comunicação propriamente dito.

A maioria das soluções que têm sido apresentadas como candidatas à implementação do serviço de multicast foram projetadas tendo em mente determinadas condições específicas de infra-estrutura para distribuição de mensagens, ou características do serviço às quais são destinadas,

ou ainda, a forma de gerenciamento dos grupos de usuários. Com relação à infra-estrutura de distribuição, serviços tem sido implementados, em geral, de forma totalmente dependente de fatores como a "topologia virtual" do sistema comunicação. Em relação às características do serviço às quais são destinadas, muitas propostas são dependentes de características ou propósitos específicos tais como os de confiabilidade, ou garantias de retardo máximo e jitter. Em relação a gerenciamento dos grupos de usuários, a forma de distribuição e armazenamento das informações relativas aos grupos determina, muitas vezes, toda a arquitetura e implementação do serviço.

Este capítulo apresenta a especificação de um framework para a implementação de serviços de multicast cuja principal característica é a generalidade e independência em relação aos possíveis sistemas de comunicação, aplicações e formas de armazenamento e gerenciamento de grupos. Partindo-se desse framework, é possível implementar serviços de multicast específicos de uma forma organizada e rápida através do reuso de toda a estrutura genérica apresentada, aliada a configuração de determinados componentes do serviço. É importante salientar que o framework especificado não propõe inovações no roteamento, ou gerenciamento de grupo e endereços multicast, limitando-se à descrição dos patterns para especificação de componentes utilizados para a provisão de um serviço de multicast genérico.

Os pontos de flexibilização do framework para o gerenciamento de grupo dizem respeito a escala de distribuição das informações do grupo, além da estrutura do esquema de endereçamento utilizado pelo sistema de comunicação específico, como descrito na Seção 5.4.1. No serviço de roteamento, descrito na Seção 5.4.2, utiliza-se uma única estrutura para a construção do protocolo de roteamento, a partir da qual aplicações ou serviços específicos podem configurar a estratégia de construção da infra-estrutura de distribuição multicast.

5.2 Trabalhos Relacionados

Serviços de multicast são oferecidos em diferentes sistemas e diferentes níveis de arquitetura. Usuários desses serviços também variam de acordo com a característica do ambiente em que o serviço é oferecido. Em um sistema operacional, por exemplo, um grupo pode ser definido como um conjunto de processos, que se comunicam através da troca de mensagens [128]. No sistema ANSA (Advanced Network Systems Architecture), grupos de objetos são introduzidos como uma abstração para comunicação de grupo [87]. Grupos possuem uma interface de gerenciamento que oferece funções para a adesão e abandono de grupo, adição e remoção de categorias de policiamento, etc. Em sistemas de comunicação, multicast é oferecido por protocolos de uma camada a entidades das camadas imediatamente superiores.

Em redes locais, a comunicação de grupo depende, em geral, da capacidade da rede subjacente para a difusão de mensagens. Os protocolos padronizados pelo IEEE 802.x e FDDI suportam transmissão multicast.

O trabalho de Deering [36], IP Multicast, foi um dos primeiros a considerar o suporte a multicast num ambiente composto por diversas redes. Essencialmente, Deering projetou um método para roteadores determinarem, dinamicamente, como datagramas IP devem ser transmitidos para grupos de multicast. Quando um roteador recebe um pacote endereçado a um grupo, envia-o por todas as interfaces que levam a membros do grupo. Se um grupo multicast estende-se por múltiplas redes, informações de membros de grupo são trocadas entre os roteadores pelo Protocolo de Gerenciamento de Grupo Inter-rede (IGMP - Internet Group Management Protocol) [46].

Uma vez comunicadas as informações dos grupos através do IGMP, o IP Multicast provê uma distribuição ponto-a-multiponto eficiente, além de fornecer mecanismos para agrupar logicamente um conjunto de estações e roteadores. Entretanto, os mecanismos propostos pelo IGMP são baseados na difusão das informações de grupo para os roteadores [86], haja visto que o mecanismo de gerência de grupo do IGMP adota uma abordagem distribuída, cabendo a cada roteador manter as informações dos participantes dos grupos que estejam presentes em alguma das subredes por ele delimitadas.

Várias estratégias têm sido propostas para suporte ao roteamento multicast [86]. Dentre elas podemos citar as Árvores Baseadas na Origem (SBT - Source Based Tree), e as Árvores Baseadas em um Núcleo (CBT - Core Based Tree). Outros tipos de esquemas de roteamento multicast têm sido estudados, como Árvores Steiner [77]. Cada uma desses esquemas tem características específicas, que levam em consideração o ambiente de comunicação utilizado. A partir dessas estratégias, vários protocolos tem sido propostos como solução para o problema de roteamento multicast.

O DVMRP (Distance Vector Multicast Routing Protocol) [107] é um protocolo para construção de uma árvore baseada na origem. O funcionamento desse protocolo é intimamente ligado ao algoritmo RPM (Reverse Path Multicasting) [86]. De acordo com esse algoritmo, o primeiro pacote de uma transmissão multicast é difundido através de uma árvore de difusão de menor caminho.[1] A partir deste ponto, mensagens de desligamento são geradas de forma a isolar as sub-redes que não possuem membros do grupo. Para isso, o DVMRP necessita das informações de gerenciamento de grupo providas pelo IGMP. O protocolo DVMRP é um protocolo de roteamento distribuído, isto é, as informações de roteamento (tabelas de rotas) estão distribuídas por todo o sistema de comunicação.

O MOSPF (Multicast Extensions to OSPF) [93] é um protocolo que mantem em todos os roteadores o estado completo dos enlaces da rede, possibilitando assim a criação da árvore de distribuição (SBT) utilizando alguma heurística de menor caminho. Como o DVMRP, o MOSPF também se utiliza de informações sobre participantes de grupos obtidas através do IGMP.

O CBT (Core Based Tree) [5], ao contrário dos protocolos anteriores, cria apenas uma única árvore para cada grupo, ou seja, não depende da localização da fonte emissora dos pacotes multicast. Um roteador, ou um conjunto de roteadores são escolhidos para formarem o centro de distribuição dos pacotes multicast. Todas as mensagens destinadas a um grupo multicast são direcionadas ao centro da árvore, que encarrega-se de replicar o pacote por toda a árvore de distribuição.

O PIM (Protocol Independent Multicast) [37] é único no sentido de que ele suporta tanto árvores baseadas na origem (SBT), como árvores compartilhadas (CBT). Este protocolos concentram-se em técnicas de roteamento intra-domínio (conjunto de redes sob controle administrativo de uma única organização).

O BGMP (Border Gateway Multicast Protocol) é um protocolo inter-domínio que constrói árvores compartilhadas bidirecionais inter-domínios, permitindo que qualquer protocolo de roteamento multicast seja usado em domínios individuais.

Em redes ATM, a implementação do serviço de multicast e broadcast não é trivial. Um transmissor precisa obter os endereços de todos os receptores para o estabelecimento de uma conexão ponto-a-multiponto. Para solucionar este problema, a IETF definiu, no documento

[1] Um roteador só retransmite um pacote se ele tiver chegado pela interface do menor caminho que leva a origem do pacote.

RFC 2022 [3], um mecanismo de registro e resolução de endereços e distribuição de informações de membros de grupo que permite oferecer suporte a um serviço de multicast sobre redes ATM baseadas nas versões UNI 3.0/3.1. O mecanismo é baseado na existência de servidores de resolução de endereços de multicast (MARS — Multicast Address Resolution Server), que associam endereços de grupos de multicast a interfaces ATM representando membros dos grupos. Entidades de resolução de endereços dos sistemas finais solicitam a um MARS um conjunto de endereços ATM (que formam um grupo), no momento em que um endereço de grupo de multicast precisa ser resolvido. Os sistemas finais, por sua vez, mantêm o MARS informado de sua entrada ou saída em um grupo.

Duas abordagens podem ser utilizadas para a construção da infra-estrutura de distribuição para redes ATM: malhas de VCs ponto-a-multiponto ou servidores de multicast (MCSs - Multicast Servers). Na abordagem da malha de VCs, cada origem estabelece um VC ponto-a-multiponto com o conjunto de destinos que são membros do grupo com o qual ela deseja se comunicar, criando assim uma malha de VCs. Na abordagem baseada em servidores de multicast, cada origem estabelece um VC ponto-a-ponto com um nó intermediário, denominado de MCS, responsável pelo estabelecimento e gerenciamento de uma única VC ponto-a-multiponto com os membros do grupo ao qual ele gerência.

A arquitetura do MARS permite que tanto malhas de VCs quanto MCSs sejam usados simultaneamente para grupos diferentes. Apesar da especificação do MARS ser independente do protocolo utilizado, ele apresenta as mesmas características e interfaces providas pelo IGMP, implementado, porém, de modo centralizado.

Os protocolos de transporte mais tradicionais como OSI-TP4 e TCP não suportam comunicação de dados multiponto. Novos protocolos como VMTP (Versatile Message Transaction Protocol) [21] e XTP (Xpress Transport Protocol) oferecem suporte ao serviço de multicast. O XTP é um protocolo de transporte[2] que oferece suporte a um serviço de transmissão com confiabilidade limitada. Não há nenhum mecanismo para o estabelecimento e gerenciamento de grupos multicast.

RMTP (Reliable Multicast Transport Protocol) [99] é um protocolo de transporte multicast confiável baseado em uma estrutura hierárquica, na qual receptores são agrupados em regiões locais (também denominados domínios), nas quais um receptor especial (receptor designado) é responsável por enviar reconhecimentos periódicos ao transmissor, processar reconhecimentos de receptores daquele domínio, e retransmitir pacotes perdidos para os receptores apropriados. RMTP é um protocolo bastante geral, uma vez que ele pode ser construído sobre qualquer rede, seja ela orientada ou não à conexão. Apesar disso, o RMTP espera que a rede subjacente seja capaz de gerar e estabelecer uma árvore de multicast do transmissor para os receptores

5.3 Interface e Arquitetura Genérica para um Serviço de Multicast

Nesta seção são apresentadas as características gerais de um ambiente que oferece um serviço de multicast. De uma forma genérica, um serviço de multicast está relacionado a algum serviço de comunicação de grupo, explícito ou implícito, entre entidades que podem ser objetos, processos, entidades de protocolos, etc. Assim, o referido ambiente pode abranger sistemas de processamento e comunicação representados por camadas de protocolos, sistemas operacio-

[2] Alguns autores classificam o XTP como um protocolo de transferência, uma vez que ele incorpora funcionalidades tanto da camada de rede como da camada de transporte do modelo de referência OSI.

nais, plataformas de componentes OO (como CORBA, COM ou Java Beans), entre outros. O serviço de multicast compõe a parte específica de um serviço de comunicação responsável pela construção de uma infra-estrutura de distribuição e o gerenciamento dos grupos.

O serviço geral de comunicação de grupo com qualidade de serviço pode ser composto de acordo com o ilustrado na Figura 5.1.

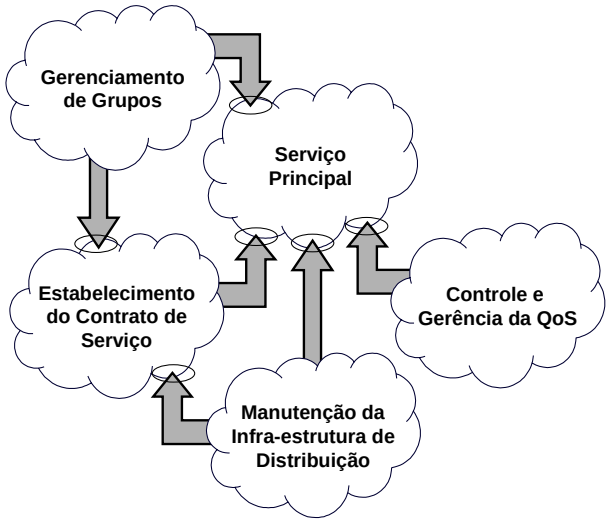


Figura 5.1: Serviços e seus meta serviços definidos pelo framework de multicast.

Tanto os serviços de gerenciamento de grupo como os de construção da infra-estrutura de distribuição poderão ser meta serviços do serviço principal e do serviço de estabelecimento de contratos de serviço. O primeiro caso acontecerá nos sistemas que não oferecem serviços com qualidade de serviço aos seus usuários.

Para que a interface de um serviço de multicast derivado do framework proposto seja genérica, ele é definido de forma a apresentar as seguintes características: (i) a possibilidade de definição de um endereçamento de grupo; (ii) a possibilidade de que o serviço seja orientado ou não orientado à conexão; (iii) o oferecimento de uma "topologia virtual" ponto-a-multiponto para a comunicação de seus usuários; e (iv) a possibilidade do gerenciamento de grupo ser centralizado ou distribuído.

O suporte ao serviço multicast é baseado nos conceitos de agrupamento e grupo. *Agrupamento* é definido como um conjunto de participantes que desejam fazer uso do serviço de comunicação multicast. A comunicação de dados no interior de um agrupamento é sempre realizada entre um membro e outro, ou entre um membro e um grupo de usuários. Cada *grupo* de usuários de um agrupamento detém um endereço de grupo.[3]

A interface genérica para um serviço de multicast define um conjunto de primitivas nas quais um *endereço de grupo* é fornecido. Cada endereço de grupo ou endereço multicast pode identificar um conjunto de endereços, cada um correspondendo a um dos membros do grupo. O transmissor não necessita conhecer os membros do grupo nem seus endereços particulares. Além disso, o transmissor não é necessariamente membro do grupo. Um grupo deste tipo é denominado um grupo aberto em contraste ao grupo fechado, no qual apenas os próprios membros podem transmitir para o grupo [36].

[3] Casos particulares desses grupos são aqueles compostos por um só participante e aquele formado por todos os participantes do agrupamento.

O serviço de multicast previsto pelo framework é ponto-a-multiponto considerando que os outros dois arranjos, ponto-a-ponto e difusão,[4] podem ser tratados como casos particulares onde o destino é formado respectivamente por apenas um participante (topologia virtual ponto-a-ponto), ou todos os participantes do sistema de comunicação (topologia virtual em barra ou por difusão).

A implementação de um serviço de multicast em um determinado nível depende do suporte dado pelo nível inferior, que pode oferecer ou não suporte ao endereçamento de grupo. Além disso, o serviço de comunicação do nível inferior pode ser ou não orientado à conexão e pode possuir capacidade de transmissão por difusão, ponto-a-ponto, ou ponto-a-multiponto. Assim, algumas adaptações serão necessárias para a provisão de um serviço específico dado um determinado sistema de comunicação do nível inferior. A modelagem do framework é genérica o suficiente de modo a permitir qualquer configuração.

A arquitetura genérica de um serviço de multicast pode ser dividida em duas partes: o gerenciamento de grupo e a construção de uma infra-estrutura de distribuição, que serão detalhados nas próximas subseções.

5.3.1 Gerenciamento de Grupo

Nesse trabalho, o termo gerenciamento de grupo é utilizado para descrever todas as ações relacionadas a composição do grupo, como manipulação de informações sobre os seus participantes, e o controle de acesso ao grupo, isto é, controle sobre a entrada e saída de participantes ao grupo.

Os esquemas de gerenciamento de grupo podem ser classificados em dois tipos: o gerenciamento de grupo distribuído, no qual as informações e o controle dos grupos estão distribuídos pelo sistema de comunicação, como é o caso do IGMP [46]; e gerenciamento de grupo centralizado, no qual existe a figura de um gerenciador de grupo centralizado, que controla todas as atividades de gerenciamento do grupo, como acontece com o MARS [3].

Para que um sistema possa prover um serviço de multicast é necessário, primeiramente, a criação da infra-estrutura de gerenciamento, que implica na criação de agrupamento e na geração de um identificador para esse agrupamento. Adicionalmente, é criado o grupo all-users, que conterá todos os participantes do agrupamento em um determinado instante.

Após criada a infra-estrutura de gerenciamento, usuários podem registrar-se junto ao serviço de multicast para fazer uso de sua infra-estrutura. O registro é feito através da entrada no agrupamento, momento em que um usuário é automaticamente inserido no grupo all-users e em um grupo onde apenas ele será elemento. Os endereços desses grupos de apenas um participante definem os identificadores de usuário. Pertencendo ao agrupamento, usuários podem criar grupos, ou juntar-se a grupos existentes.

5.3.2 Construção da Infra-Estrutura de Transmissão Multicast

A função de roteamento em um sistema de comunicação é responsável por, dado um endereço de destino, escolher uma rota a ser utilizada para o encaminhamento do tráfego a ela destinado. O roteamento nem sempre é realizado de forma atômica, podendo estar espalhado ao longo do tempo pelas diversas fases da operação de um sistema de comunicação. Em sistemas com serviço orientado a conexão implementado através de circuito virtual, por exemplo, rotas

[4] Nesse trabalho, não foi considerada comunicação multiponto-a-multiponto.

são estabelecidas a priori, no momento do estabelecimento da conexão, de forma separada do encaminhamento das mensagens propriamente dito. Em outros tipos de serviço, rotas são estabelecidas dinamicamente conforme cada mensagem é enviada. Os algoritmos utilizados para escolha de rotas também são os mais variados, como já apresentado na Seção 5.2, podendo levar em consideração diversas métricas para a escolha do caminho mais adequado como, por exemplo, a QoS [77].

Independente das variações de algoritmos e escalas de tempo nas quais eles operam, para que uma função de roteamento possa funcionar de forma adequada, é necessária a criação de uma infra-estrutura de transmissão, i.e., os elementos do sistema que cooperam para a escolha de rotas devem ser alimentados de informações de alcançabilidade e disponibilidade de recursos, que sirvam de base para a tomada das decisões de roteamento.

A parte do framework de multicast responsável pela criação da infra-estrutura de transmissão cuida do fornecimento e distribuição das informações de alcançabilidade dos componentes endereçáveis do sistema. O formato e a maneira com que essas informações devem ser distribuídas depende do tipo de infra-estrutura que o ambiente ou sistema de comunicação inferior fornece. Em ambientes cujo suporte à transmissão é ponto-a-ponto, por exemplo, serviço de multicast pode ser implementado através do envio de várias cópias da mensagem para os vários destinos, sem o conhecimento do usuário (“Multicast by unicast”). Outra abordagem, utilizada quando a infra-estrutura provê suporte à transmissão por difusão, é o envio de uma única cópia de cada mensagem endereçada ao grupo para todos os usuários, cabendo a cada receptor aceitar apenas as mensagens direcionadas ao grupo que ele pertence (“Multicast by broadcast”). Finalmente, quando a infra-estrutura oferece suporte direto à comunicação ponto-a-multiponto, o serviço multicast pode se utilizar dessa capacidade, e a própria infra-estrutura encarrega-se de fazer as devidas replicações, evitando que cópias de uma mesma mensagem percorram um mesmo caminho mais de uma vez (“Multicast by multicast”). Essa última estratégia é também comumente associada à existência de uma topologia virtual em árvore.

A parte do framework responsável pelo gerenciamento de grupo está diretamente relacionada à criação da infra-estrutura de transmissão uma vez que, de acordo com a entrada ou saída de participantes de um grupo, as informações sobre rotas devem ser modificadas para refletir a nova estrutura de distribuição. Por conseguinte, o framework de transporte e o de QoS [57] também têm os respectivos comportamentos influenciados, já que alterações nos participantes de um grupo podem demandar o estabelecimento de novas conexões ou a adição de novos participantes em conexões já existentes.

5.4 Framework Para Serviço de Multicast

A definição do serviço através de um framework segue uma abordagem adotada em alguns trabalhos recentes [24, 57, 117], na qual um serviço pode ser configurado de acordo com as necessidades do usuário, ou do ambiente disponível. O framework aqui proposto pode ser aplicado independentemente da escala de distribuição de seus usuários (em um mesmo processo, em processos distintos de uma mesma máquina, em máquinas distintas de uma mesma rede ou de redes distintas interconectadas), assim como do tipo de sistema de comunicação utilizado para a troca de mensagens. Utilizou-se uma modelagem orientada a objetos, apresentada através da notação UML (Unified Modeling Language) [109].

O framework é dividido em duas partes: (i) Group Management Service, responsável por manter uma base de dados de informações sobre grupos (Group Information Base) e suas com-

posições, e, utilizando-se dessas informações, dar suporte por exemplo, ao mapeamento de endereços de grupos aos endereços de seus participantes, ou mapeamento de um endereço de grupo de um nível ao endereço de grupo do nível inferior, ou aos vários endereços individuais do nível inferior dos participantes do grupo (resolução de endereço); e (ii) Routing Support Service, responsável por criar e gerenciar a infra-estrutura de distribuição multicast.

5.4.1 Serviço de Gerenciamento de Grupo

A PARTE do framework responsável pelo serviço de gerenciamento de grupos oferece os mecanismos para manter a base de informações dos grupos e seus componentes, além das funções e informações necessárias para resolução de endereços de grupo, utilizadas pelos mecanismos de transmissão. A resolução de endereços é responsável por dar suporte ao mapeamento entre um endereço de nível *N* para um ou mais endereços de nível *N-1*. A maneira com que as informações de grupo estão distribuídas e a forma de resolução de endereços são dois aspectos intimamente relacionados.

A resolução pode ser feita de forma direta ou por intermédio de protocolos de resolução. Na forma direta, o protocolo da camada *N* é capaz de traduzir localmente o endereço para o endereço de nível *N-1*, podendo utilizar-se de uma tabela ou de uma função de mapeamento. No caso do IP Multicast sobre Ethernet, por exemplo, o mapeamento é bastante simples: os 23 bits menos significativos do endereço IP de classe D devem ser colocados nos 23 bits menos significativos do endereço de multicast Ethernet. Na resolução através de protocolos, a entidade da camada *N* faz uma requisição solicitando que alguma outra entidade (ou entidades) retornem os endereços de nível *N-1* correspondentes. O protocolo ARP [103] é um exemplo de protocolo de resolução para endereços IP unicast. Quando o nível *N-1* já oferece um serviço com endereçamento multicast, a resolução direta pode ser realizada através do mapeamento de cada endereço de multicast de nível *N* em um endereço de multicast do nível *N-1*, que corresponde, exatamente, à abordagem seguida pelo IP multicast sobre Ethernet. Caso não exista endereçamento multicast no nível inferior, as seguintes abordagens podem ser utilizadas, dependendo da distribuição das informações de grupo:

1. Quando as informações de grupo estão centralizadas em um servidor para resolução de endereços (SRE), a resolução através de protocolo pode ser utilizada requisitando-se desse servidor a resolução desejada; o SRE é responsável apenas pela resolução de endereços e não pela distribuição das mensagens para os integrantes do grupo. O transmissor, após receber os endereços traduzidos é responsável por distribuir mensagens para os integrantes do grupo.

2. Quando as informações de cada grupo estão centralizadas em um servidor para distribuição de mensagens (SDM), a resolução direta pode ser utilizada, mapeando-se cada endereço de grupo ao endereço *N-1* do SDM correspondente; assim, ao receber as mensagens, o SDM imediatamente as distribui aos integrantes do grupo. Dessa forma, clientes desconhecem o fato de que o sistema de nível *N-1* não possui endereçamento multicast já que o endereço do SDM passa a servir como o endereço de multicast sob o ponto de vista desses clientes.

3. Quando o nível inferior tem capacidade de difusão e as informações de grupo estão distribuídas, a resolução pode ser feita através de um protocolo no qual a requisição de

resolução é difundida e as entidades capazes de responder enviam ao cliente os endereços mapeados. A forma mais óbvia de distribuição é aquela na qual cada participante tem as informações dos grupos aos quais ele próprio pertence; dessa forma todos os componentes de um grupo respondem a requisições de resolução de endereços daquele grupo, cada um com seu próprio endereço de nível *N-1*.

A primeira abordagem foi utilizada pelo IETF na definição do MARS para a resolução de endereços de grupo em redes IP sobre ATM. A definição do MARS prevê ainda que a segunda abordagem pode ser utilizada em conjunto com o servidor de resolução, na qual o SDM (denominado Multicast Server — MCS) atua na parte da distribuição e o MARS propriamente dito na parte de resolução. Assim, o MARS é responsável por resolver os endereços de grupo alternativamente: nos endereços *N-1* dos componentes do grupo, ou no endereço *N-1* de um MCS. No caso em que não há MCS, o cliente, após a resolução, é responsável por criar uma conexão ponto-a-multiponto com todos os integrantes. Havendo um MCS, o cliente deve estabelecer com ele uma conexão ponto-a-ponto; o MCS, por sua vez, é que é responsável por manter uma conexão ponto-a-multiponto com todos os integrantes do grupo em questão.

Outras abordagens são possíveis através da combinação dessas três. A combinação da segunda abordagem com a primeira é conveniente pois o uso da segunda abordagem de forma isolada dificulta a criação e destruição de grupos de forma dinâmica. A terceira abordagem, apesar de viável, não é utilizada na prática para resolução de endereços de grupo. Porém, para a resolução de endereços unicast, essa abordagem corresponde exatamente ao ARP tradicional (ou considerando cada grupo como sendo formado por um só elemento).

A partir dessas análises sobre as possíveis formas de resolução e gerenciamento, que englobam inclusive as formas de resolução utilizadas para endereços unicast, chegou-se a uma modelagem genérica que constitui a parte do framework relativa ao gerenciamento de grupo, ilustrada na Figura 5.2.

O gerenciamento de grupo gira em torno da manutenção de uma base de informação de grupo. Essa base é composta por duas tabelas principais, que relacionam endereços de um nível *N*, sejam eles grupos de um ou mais participantes, a endereços do nível *N-1*. Por apresentar uma estrutura única, essas tabelas são implementadas a partir de uma classe comum (GroupTable), a partir da qual uma instância para o gerenciamento de grupo, e outra, para a resolução de endereço são criadas.

O principal componente do serviço de gerenciamento de grupo é o GroupManager, onde são definidas as operações básicas necessárias para a constituição e manutenção de um grupo (GroupCreate, GroupDestroy, GroupJoin, e GroupLeave). Essas operações possuem uma estrutura única, tanto para o modelo de gerenciamento centralizado como distribuído. Todas essas operações são formadas, basicamente, por uma atualização da base de dados local do grupo e pela notificação da alteração. O único ponto de configuração dessa classe está exatamente na definição do endereço utilizado para a notificação das alterações, obtido através da função getNotifyAddr(). Assim a estrutura é a mesma para qualquer abordagem de implementação.

Quando um usuário do serviço solicita uma operação, a notificação será feita através do envio de uma mensagem especial, determinada pelo tipo de operação (criação de grupo, destruição de grupo, adesão a grupo, ou saída de grupo) para um endereço configurado de acordo com o modelo do serviço. No modelo centralizado, essa notificação será enviada para o servidor de gerenciamento de grupo. No servidor, a base de dados dos grupos é alterada, e a notificação é enviada para todos os participantes do agrupamento (all-users), ou para o servidor de multicast.

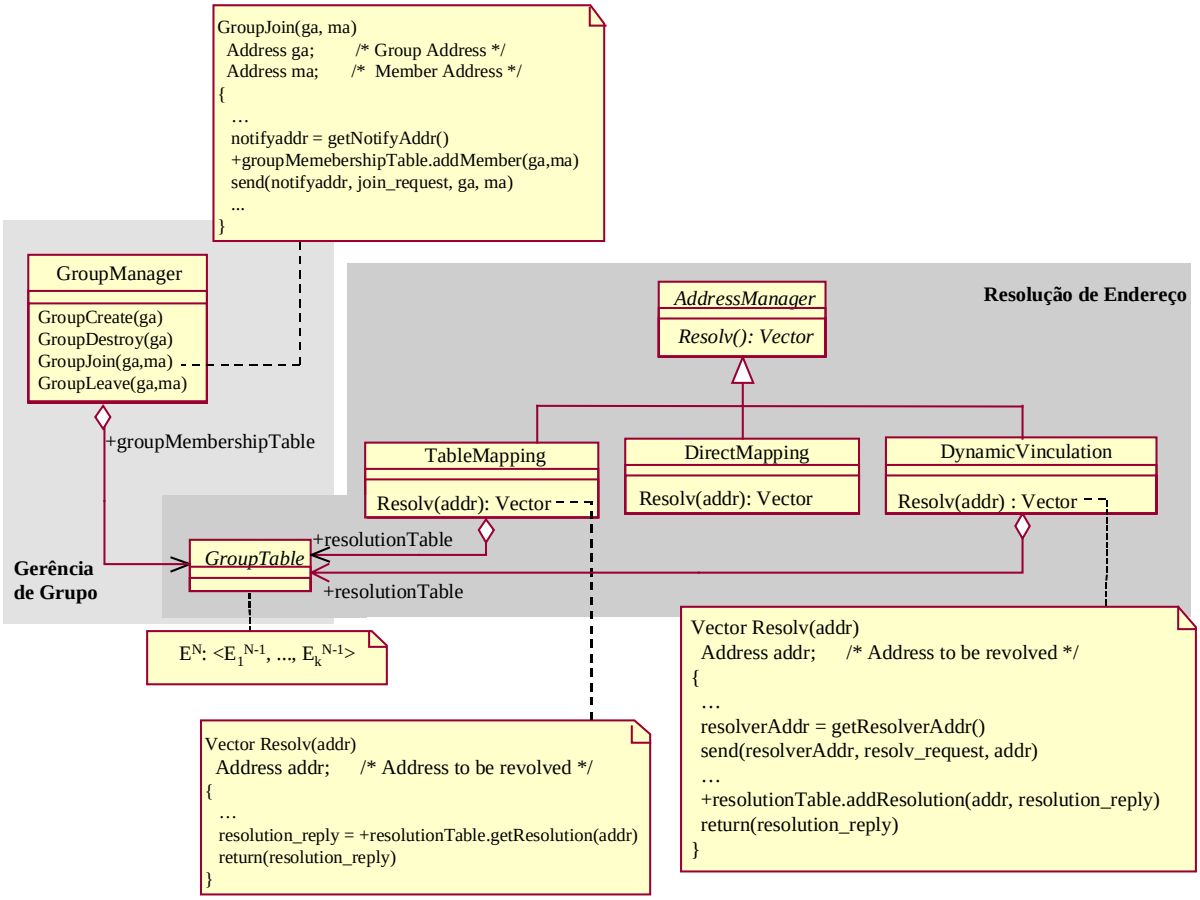


Figura 5.2: Modelo de Gerenciamento de Grupo

No modelo distribuído, a notificação do usuário será enviada para o endereço all-users. Em ambas as abordagens, as informações de notificação serão também utilizadas para a atualização das informações de resolução de endereço, tanto nos clientes, como no servidor de multicast.

O modelo para resolução de endereço é formado basicamente por uma operação Resolv, definida em AddressManager, que especifica a forma de resolução de endereço para um dado grupo. A subclasse DirectMapping é a responsável pelo mapeamento direto através de funções (como no caso do IP multicast sobre Ethernet), enquanto que TableMapping e DynamicVinculation cuidam, respectivamente, da resolução local (utilizada por servidores de resolução) e resolução por protocolo (utilizada pelo usuário que requisita a resolução). Um usuário em um ambiente sem mapeamento direto, envia uma requisição para um endereço que pode ser configurado, obtido através da função getResolverAddr(). Esse endereço será o endereço de um servidor, no caso centralizado, ou o endereço all-users, no caso distribuído. No servidor, uma instância da classe TableMapping será utilizada para a resolução do endereço. Ao receber a resposta, o cliente pode atualizar uma tabela local que funciona como uma "cache de resolução".

5.4.2 Construção da Infra-estrutura de Distribuição

O serviço de roteamento é baseado em dois patterns principais: Facade (RoutingSupportAgent) e Strategy (RoutingProtocol) [53]. A Figura 5.3 apresenta a estrutura do serviço de roteamento multicast. Na figura, contornado, pode-se ver a estrutura genérica do serviço, enquanto na parte

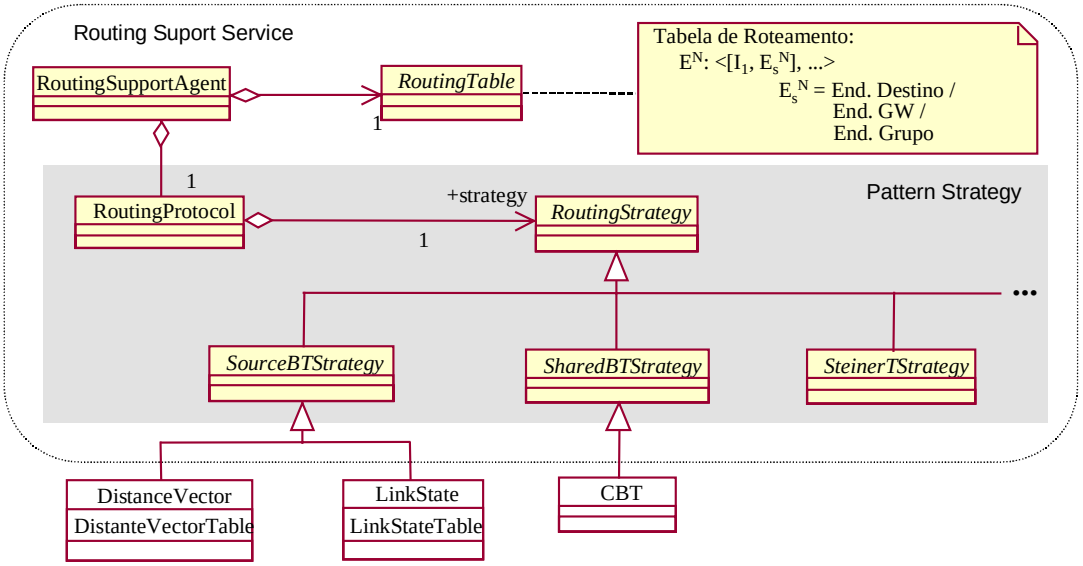


Figura 5.3: Framework do Serviço de Suporte ao Roteamento Multicast.

inferior, pode-se ver algumas especializações dos pontos de flexibilização do serviço.

O serviço de roteamento pode ser implementado de modo centralizado, ou distribuído. No modo centralizado, as funções de roteamento são concentradas em um único ponto do sistema de comunicação. Além das funcionalidades, também estão concentradas em um ponto todas as informações de roteamento no sistema. No modo distribuído, cada entidade do sistema de comunicação é responsável por manter e manipular suas próprias informações de roteamento.

A base de dados contida na tabela de roteamento (RoutingTable) representa uma abstração lógica da infra-estrutura de distribuição multicast. Basicamente, essa tabela mapeia endereços

do nível *N* a tuplas contendo interface de saída de dados, e endereços do próximo destino, que pode ser um usuário individual, um gateway, ou um grupo.

O protocolo de roteamento (RoutingProtocol) é responsável por manter as informações contidas na tabela de rotas do agente de roteamento. A escolha da heurística adotada pelo protocolo de roteamento é fundamentada nas características da aplicação. O framework de serviço de roteamento provê facilidades para se configurar um protocolo de roteamento, através da escolha e especialização da estratégia de roteamento (RoutingStrategy), baseado nas mais diferentes características das aplicações, e do sistema de comunicação utilizado.

5.4.3 Aplicação do Framework

Como diversas vezes mencionado, o framework proposto poderá ser utilizado em qualquer nível de comunicação, quer seja ao nível de objetos dentro de uma thread, quer seja de threads dentro de um processo, quer seja de processos em uma estação, e de estações em uma subrede e interredes, e assim por diante. Objetiva-se que o framework seja genérico de tal forma que possa ser empregado independente do nível de abstração e da distribuição das entidades participantes.

Mostraremos nessa seção dois casos de utilização do framework. A Seção 5.4.3.1 apresenta a configuração de um serviço de gerenciamento de grupo centralizado. A Seção 5.4.3.2 apresenta a implementação de um agente mediador de um serviço de multicast.

5.4.3.1 Configuração de um Serviço de Gerenciamento de Grupo e Resolução de Endereço Centralizado

A configuração de um serviço de gerência e resolução de grupo centralizado é ilustrada na Figura 5.4. Um servidor que concentra todas as informações dos grupos do agrupamento deve ser configurado. Esse servidor deve possuir uma instância de GroupManager e uma de AddressManager, responsáveis por gerenciar e manter todas as informações do grupo. A configuração em GroupManager diz respeito somente ao endereço utilizado para a notificação da alteração (obtido através da função getNotifyAddr()) que, nesse caso, por não possuir servidor de multicast, será o endereço all-users.

O cliente do serviço deve possuir uma instância de GroupManager, onde devem ser configuradas as funções de criação e manutenção de grupo. No cliente, o endereço de notificação da alteração do grupo deve ser configurado como sendo o endereço do servidor de gerenciamento de grupo. Deve ser configurado, no cliente, o endereço do gerenciador de resolução a ser utilizado para o mapeamento do endereço. O resultado da resolução é então mantida em um cache de resolução.

5.4.3.2 Implementação de um Agente Mediador de um Serviço de Multicast

A Figura 5.5 ilustra um caso de utilização do framework para o projeto de um agente mediador para comunicação de grupo centralizado (Multicast Service Broker - MSB). O MSB possui dois componentes: um Gerenciador de Grupo (GM), e um Gerenciador de Suporte a Roteamento (RM), instanciados a partir dos componentes descritos na seções 3.1 e 3.2, respectivamente. A configuração das classes do GM é feita como mostrado na seção anterior, adotando uma abordagem centralizada para a provisão do serviço. O RM é responsável pelo cálculo da árvore de menor caminho para a comunicação de grupo. A estratégia adotada pelo protocolo de construção da árvore foi a da árvore Steiner.

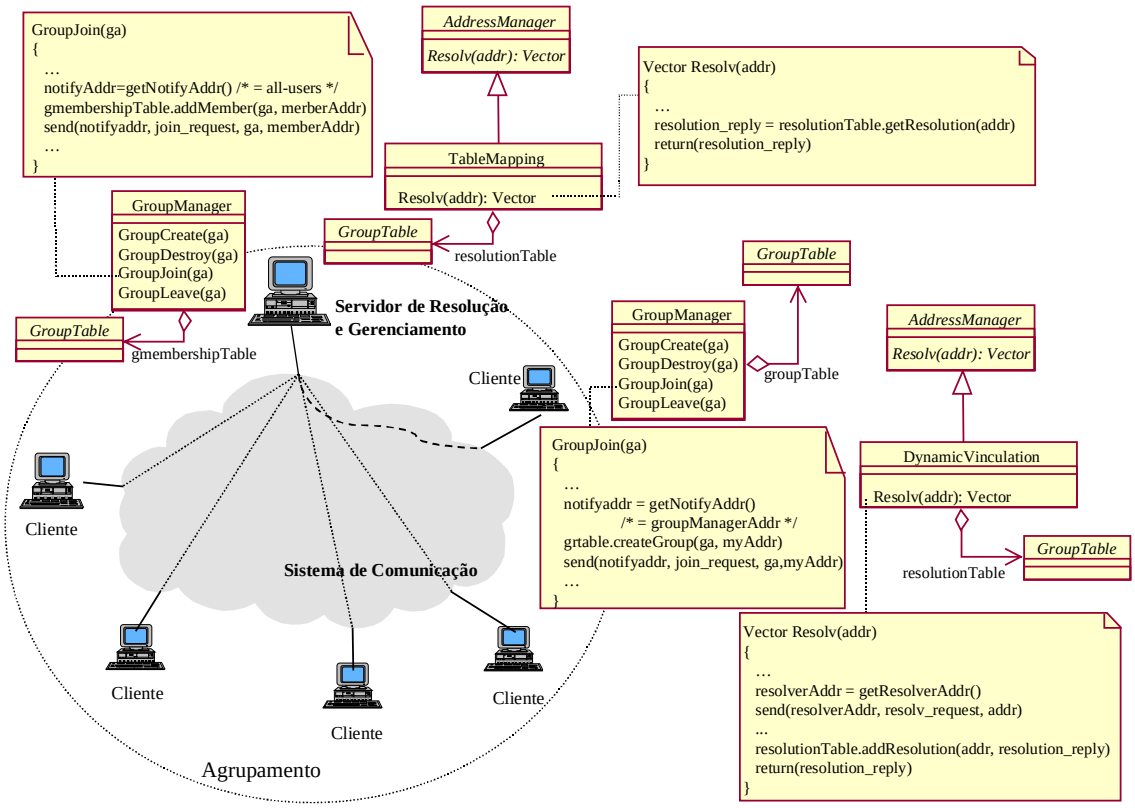


Figura 5.4: Exemplo de Configuração do Serviço de Gerenciamento de Grupo.

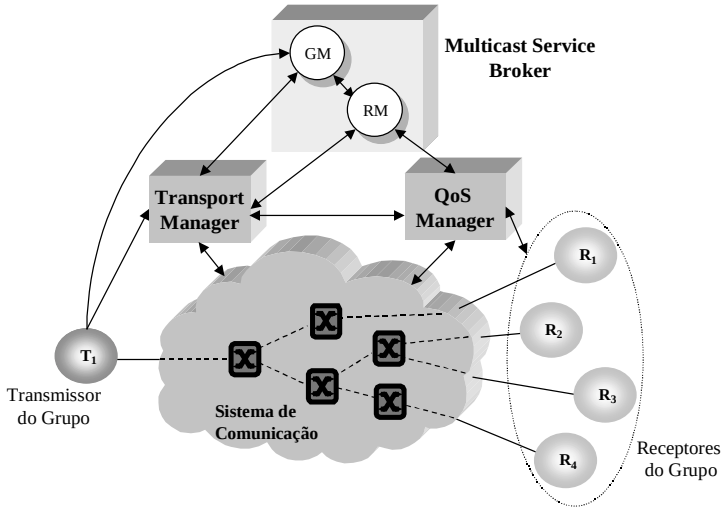


Figura 5.5: Arquitetura de Aplicação do Framework.

O MSB também faz uso de informações providas por um Gerenciador de Qualidade de Serviço do ambiente (QoS Manager - QoSM), definido utilizando-se um framework para provisão de serviço de QoS. O QoSM é responsável por prover informações sobre o estado do sistema de comunicação, enviando grafos de conectividade para o cálculo da árvore de multicast, de acordo com a QoS exigida pelo transmissor e aceita pelos receptores e entidades intermediárias do sistema de comunicação. A árvore de distribuição calculada, pelo RM, é então enviada para Gerenciador de Transporte (Transport Manager - TM) para que as conexões possam efetivamente ser estabelecidas, em conjunto com o QoSM.

Um protótipo para implementar a arquitetura proposta utilizando o framework para serviço de multicast está sendo desenvolvido em uma plataforma de simulação construída sobre comutadores virtuais. A plataforma consiste de um comutador ATM conectado à estações de trabalho. Cada estação de trabalho executa um servidor RMI [125] para sinalizar e controlar os comutadores virtuais. Um comutador virtual é representado por uma thread em um servidor RMI. O MSB é simulado como um cliente RMI (Remote Method Invocation) que realiza as funções do GM, RM, TM e QoSM. Na plataforma de prototipação, os enlaces que ligam os comutadores virtuais são emulados por enlaces ATM reais que conectam as estações através do comutador ATM.

Devido ao número relativamente pequeno de comutadores ATM instalados na plataforma de testes disponível, recorreu-se a uma técnica de prototipação usando comutadores virtuais implementados como objetos simulando o comportamento de comutadores reais, tal como proposto em [81]. A plataforma de simulação é criada em um ambiente distribuído com quatro estações conectadas a um comutador ATM IBM (IBM 8260), como mostrado na Figura 5.6. A razão de

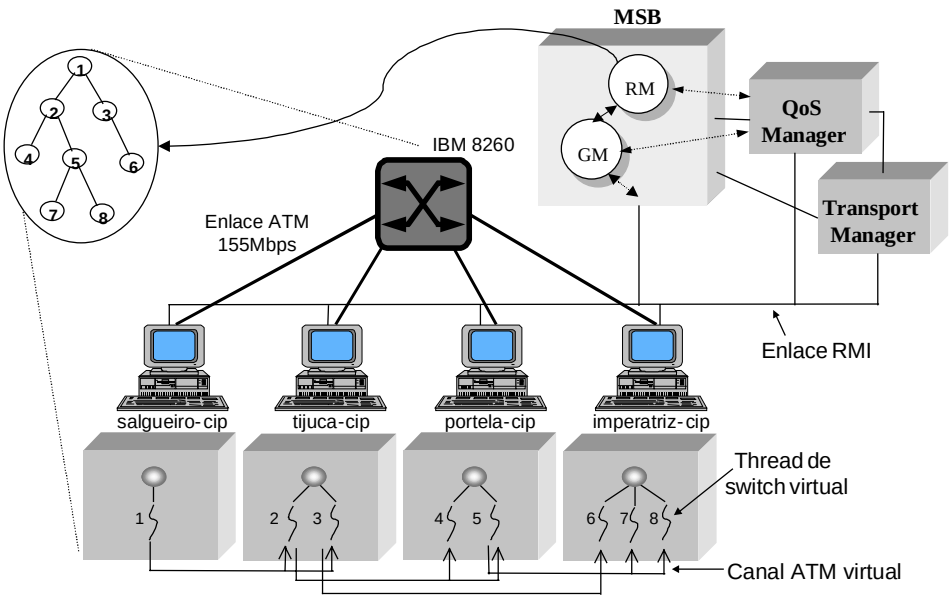


Figura 5.6: Plataforma de Simulação.

usar quatro estações é fundamentada na teoria de que qualquer região em um mapa pode ser rotulada por quatro cores, de tal sorte a garantir que nenhuma região adjacente possua a mesma cor. Portanto, para simular qualquer topologia de rede (grafo planar), os comutadores virtuais

podem estar distribuídos em quatro estações.[5] Assim é garantido que quaisquer dois comutadores vizinhos são simulados em estações diferentes e conectadas por circuitos virtuais (VC) através do comutador ATM. Esta estratégia de simulação facilita a emulação de reserva de QoS para cada enlace em uma árvore de multicast. Como indicado na figura, o enlace entre os nós 1 e 2 é mapeado no VC que liga o comutador virtual 1, na estação salgueiro-cip e o comutador 2 na estação tijuca-cip.

O TM (modelado como um cliente RMI) é responsável por estabelecer, através do QoSM (também modelado como um cliente RMI) a árvore de multicast criada pelo RM. Quando o TM recebe a topologia da árvore de multicast, ele sinaliza, através do QoSM, aos servidores RMI em diferentes estações de trabalho para que iniciem os comutadores virtuais e configurem enlaces ATM com a QoS negociada pelos participantes da aplicação multicast.

5.5 Conclusões

Neste capítulo apresentou-se uma especificação de um framework para a implementação de serviços de multicast cuja principal característica é a generalidade e independência em relação aos possíveis sistemas de comunicação, aplicações e formas de armazenamento e gerenciamento de grupos. Partindo-se desse framework, foi possível especificar serviços de multicast específicos através do reuso da estrutura genérica apresentada, aliada a configuração de determinados componentes do serviço. Mostrou-se como o framework pode ser utilizado para a definição de um serviço de gerência e resolução de endereço, adotando-se o modelo centralizado como exemplo.

A definição da estrutura básica do framework foi possível graças a uma análise dos vários cenários possíveis de configuração do serviço de gerência de grupo, isolando-se o modelo básico, e verificando sua validação nos mais diferentes casos de utilização.

A utilização do framework foi exemplificada através da implementação de um serviço de comunicação multicast, apresentado na seção 5.4.3.2.

[5] A configuração da topologia da rede e a distribuição do comutadores virtuais nas quatro máquinas é feita previamente.

Capítulo 6

Conclusões

ESTE capítulo é dedicado a uma reavaliação geral. Em primeiro lugar, uma revisão dos objetivos e contribuições desta tese será feita. Segue-se então uma parte de críticas, em especial, sobre os vários pontos em que o modelo do Capítulo 3 ainda requer investigações. Alguns deles, como já mencionado na Seção 1.2, foram propositadamente deixados em aberto como forma de generalidade do modelo. Outros, porém, foram resultado de dificuldades encontradas que aqui serão mencionadas. A relação desse modelo com as técnicas de engenharia de software utilizadas também será discutida. Por fim, procura-se delinear como este trabalho pode ser continuado.

6.1 Objetivos

O presente trabalho objetivou propor um modelo que fornecesse os mecanismos (ou abstrações) para a representação e programação de aspectos de QoS e comunicação de grupo (multicasting) em serviços de comunicação. Uma de suas principais características é seu aspecto recursivo, por meio do qual aplicações deverão poder ser construídas sobre os serviços de comunicação especificados que, por sua vez, poderão ser sucessivamente construídos sobre serviços mais simples. Dessa estratégia, esperou-se obter dois benefícios: (i) um mecanismo para o controle da complexidade de forma que os aspectos da implementação dos serviços de comunicação do nível inferior poderão ser ora revelados ao nível superior (com o intuito de efetuar as adaptações desejadas), ora abstraídos (de forma a esconder os detalhes em abstrações de alto nível que simplificam a utilização desses serviços); e (ii) uma diminuição da carga cognitiva, conseqüência da utilização de um mesmo modelo para os diferentes níveis de abstração.

O modelo proposto foi construído tendo em mente que o funcionamento de um sistema de comunicação pode ser totalmente regulado sobre uma *base de informações distribuída* que controla e auxilia o funcionamento de todo o sistema. Um dos pontos de partida foi o estudo da utilização de *frameworks* na modelagem de serviços de comunicação. O objetivo da construção e estudo dos dois frameworks apresentados foi o de exercitar e relacionar alguns dos conceitos do modelo e, ao mesmo tempo, levantar os pontos ou "peças", correspondendo a unidades de informação, que podem estar disponíveis na base de informações para permitir a adaptabilidade do serviço.

Procurou-se formar, por meio da definição dos elementos básicos e das restrições para sua configuração, uma espécie de "linguagem básica" para construção de serviços, a partir da qual

estruturas pré-montadas poderiam ser definidas para modelar necessidades comuns em serviços de comunicação (como a provisão da QoS e a comunicação de grupo). Essas estruturas pré-montadas poderiam ser, ao mesmo tempo, usuárias da linguagem básica e geradoras de linguagens mais específicas e sua definição corresponderia exatamente à definição de frameworks.

A partir da definição dessa unidades de informação, procurou-se comparar as formas de adaptação propostas na literatura com relação aos tipos de elementos manipulados, o momento ou fase em que tais manipulações podem ocorrer, e os responsáveis por tais manipulações. Os conceitos estabelecidos permitiram analisar alguns mecanismos comuns, presentes em redes tradicionais, bem como diferenciar de que forma as novas arquiteturas que tratam do problema da adaptabilidade se relacionam com esses mecanismos.

6.2 Contribuições

Uma das maiores contribuições que pode ser ressaltada dessa tese é o estabelecimento de uma relação mais estreita entre vários assuntos hoje tratados na área engenharia de software com os da nova disciplina de engenharia de serviços de telecomunicações. Em especial, enfatizou-se o aspecto de como se pode fazer uso de técnicas de orientação a objetos, frameworks e estilos arquiteturais para a construção de serviços adaptáveis.

A importância e o significado do tratamento de informações em arquiteturas de comunicação e o impacto causado nos modelos de serviço foram discutidos e, ao longo dessa discussão, alguns trabalhos relacionados à modelagem e à adaptabilidade desses serviços foram comentados. As diversas fases que, em geral, estão presentes no ciclo de vida de um serviço forma apresentadas e o impacto da adaptabilidade sobre esse ciclo foi estudado. Dessa análise, foi possível obter subsídios para classificar alguns mecanismos de adaptação e programação presentes em trabalhos encontrados na literatura.

O modelo proposto nesta tese pode ser comparado às visões de informação, computacional e de engenharia do ODP. Porém, a divisão dessas visões é feita de forma mais flexível já que a própria recursividade do modelo faz com que a visão computacional se transforme na visão de engenharia de um outro nível de abstração. A definição de meta serviços também pode ser encarada como uma forma de mudança no tipo de visão que se tem de um sistema. Os mecanismos que permitirão a abstração e reificação refletem esse tipo de mudança de visão.

A questão da reflexividade em plataformas de componentes, também abordada por outros trabalhos, teve nesta tese algumas características próprias: (i) a utilização de uma organização em camadas, que regula a definição dos mecanismos de adaptabilidade e orquestração, (ii) a definição de frameworks, que correspondem a estruturas ou modelos estabelecidos de forma a regular as adaptações a serem feitas no ambiente e (iii) a definição de uma modelagem que captura, especificamente, as abstrações comuns presentes em serviços de comunicação.

A classificação das estratégias de adaptação, apresentada na Seção 3.6.1 pode ser considerada mais genérica do que as encontradas em trabalhos similatres no sentido de que ela está baseada somente nos elementos do modelo conceitual definido para os serviços, que é, na realidade, um meta modelo a ser utilizado para a construção de arquiteturas.

Os dois frameworks apresentados fornecem uma visão e um estudo aprofundados dos assuntos a que se dedicam. Seu relacionamento com o restante de um sistema complexo pôde ser fornecido e, graças à organização dos serviços apresentada no Capítulo 3, baseada nos conceitos de serviços e meta serviços, apontou-se a estrutura geral de um serviço de comunicação no qual a QoS e comunicação de grupo se acomodam de maneira natural.

6.3 Críticas e Trabalhos Futuros

Várias questões levantadas no início desta tese ainda permanecem em aberto ou sem um tratamento adequado. Em primeiro lugar, se, por um lado, o modelo é genérico o suficiente para capturar os conceitos existentes em vários ambientes ou arquiteturas, ele foi, por vezes, demasiadamente sem consistência para responder a algumas perguntas importantes. Uma delas, diz respeito a forma exata com que os mecanismos de abstração e reificação deverão ser concretizados em um ambiente real. A estruturação em camadas por exemplo, coloca sérias e novas questões sobre a forma e a complexidade que esses mecanismos podem vir a assumir, sejam eles implementados através de linguagens ou não. Um dos pontos a investigar é o da utilização de técnicas relacionadas a *Aspect Oriented Programming* [74] para contornar esses problemas.

A maior conseqüência da introdução de flexibilidade fornecida por um modelo de arquitetura adaptável é o aumento da complexidade, principalmente para programação e manutenção das bases de informações, incluindo a configuração de protocolos. Linguagens de configuração adequadas deverão ser propostas de forma a refletir essas atividades. Modelos que permitam avaliar, de forma mais precisa, a adequação do modelo aqui proposto devem ser utilizados. Modelos formais como o apresentado em [100] podem auxiliar nessa tarefa, fornecendo uma maneira precisa de representar os processo de configuração dos elementos do modelo.

A diversidade de mecanismos foi um dos pontos de complexidade que se procurou comparar através do modelo. Ainda não é possível porém responder com precisão a todas as questões sobre a possibilidade de coexistência dos diversos mecanismos de adaptação ou sua compatibilidade. O modelo permitiu, no máximo, uma visualização das fases e alvos comuns da atuação de alguns dos mecanismos. Esse é um ponto que merece ser revisado.

Por fim, cabe comentar que o presente trabalho resultou de vários esforços que foram sendo conduzidos em paralelo e em grupo. O resultado desta tese é também a tentativa de reunir e consolidar os conceitos que foram se tornando úteis e comuns no desenvolvimento de ambos os frameworks, auxiliados pela visão estruturada a qual o modelo se propõe a fornecer. Todavia, nem sempre esses conceitos foram plenamente compatíveis e aplicáveis da forma genérica em que o modelo estava sendo proposto. Assim, foi por vezes difícil apresentar todos os aspectos modelados em cada um desses frameworks utilizando os conceitos apresentados no Capítulo 3. Nesse sentido, um levantamento desses aspectos deverá fornecer um grande material para a revisão do modelo.

Como trabalho futuro, a longo prazo, pretende-se chegar a criar um ambiente de desenvolvimento que permita a construção dos serviços que exibam as características aqui levantadas. Entre outras coisas, um tal ambiente deverá ser composto de linguagens e ferramentas que reflitam os conceitos básicos apresentados. Esta tese representou apenas o início deste trabalho, procurando contribuir em alguns de seus alicerces, iluminando o início desse longo caminho que a disciplina de engenharia de serviços de telecomunicações ainda terá de percorrer.

Referências Bibliográficas

[1] M. B. Abbott e L. L. Peterson. A language-based approach to protocol implementation. *IEEE/ACM Transactions on Networking*, 1(1):4–19, fevereiro 1993.

[2] R. Allen, R. Douence e D. Garlan. Specifying and analyzing dynamic software architectures. Em *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE'98)*, Lisbon, Portugal, março 1998.

[3] G. Armitage. Multicast over UNI 3.0/3.1 based ATM. RFC 2022, novembro 1996.

[4] The ATM Forum. *Integrated Layer Management Interface (ILMI) Specification*, setembro 1996. Document af-ilmi-0065.000.

[5] T. Ballardie, P. Francis e J. Crowcroft. Core base trees (CBT): An architecture for scalable inter-domain multicast routing. Em *Proceedings of the SIGCOMM '93 Symposium*, 1993.

[6] H. Berndt, P. Graubmann e M. Wakano. Service specification concepts in TINA-C. Em *International Conference on Intelligence in Broadband Services and Networks*, 1994.

[7] U. D. Black. *The Intelligent Network: Customizing Telecommunications Networks and Services*. Advanced Communications Technologies. Prentice Hall PTR, 1998.

[8] G. S. Blair e G. Coulson. The case for reflective middleware. Relatório Técnico MPG-98-38, Distributed Multimedia Research Group, 1998.

[9] G. S. Blair, G. Coulson, N. Davies, P. Robin e T. Fitzpatrick. Adaptive middleware for mobile multimedia applications. Em *Proceedings of the NOSSDAV'97*, págs. 259–273, St. Louis, Missouri, USA, maio 1997.

[10] G. S. Blair, G. Coulson, P. Robin e M. Papathomas. An architecture for next generation middleware. Em *Proceedings of the Middleware '98*, The Lake District, England, novembro 1998.

[11] G. S. Blair e J. B. Stefani. *Open Distributed Processing and Multimedia*. Addison Wesley Longman Ltd., 1997.

[12] R. Braden, D. Clark e S. Shenker. Integrated services in the internet architecture: An overview. RFC 1633, 1994.

[13] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad e M. Stal. *A System of Patterns*. Wiley, 1995.

[14] A. Campbell, C. Aurrecoechea e L. Hauw. A review of QoS architectures. *ACM Multimedia Systems Journal*, novembro 1995.

[15] A. T. Campbell. *A Quality of Service Architecture*. Tese de Doutorado, Computing Department – Lancaster University, janeiro 1996.

[16] A. T. Campbell et al. Spawning networks. *IEEE Network Magazine*, 13(4):16–29, julho 1999.

[17] A. T. Campbell, M. E. Kounavis, H. D. Meer, K. Miki e J. Vicente. A survey of programmable networks. *ACM Computer Communications Review*, abril 1999.

[18] A. T. Campbell, M. E. Kounavis, H. D. Meer, K. Miki e J. Vicente. Towards programmable virtual networking. Em *Proceedings of the 2nd IEEE Conference on Open Architectures and Network Programming (OPENARCH'99)*, 1999.

[19] CCITT Recommendation X.25. *Interface Between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode and Connected to Public Data Networks by Dedicated Circuit*, 1988.

[20] M. C. Chan e A. A. Lazar. Designing a CORBA-based high performance open programmable signaling system for ATM switching platforms. *IEEE Journal on Selected Areas in Communications*, 17(9), setembro 1999.

[21] D. Cheriton. VMTP: Versatile message transaction protocol — protocol specification version 0.7. Relatório técnico, Computer Science Department, Stanford University, 1988.

[22] H. Chu e K. Nahrstedt. A soft real time scheduling server in UNIX operating system. Em *European Workshop on Interactive Multimedia Systems and Telecommunication Services (IDMS'97)*, setembro 1997.

[23] S. Colcher e L. F. G. Soares. Designing configurable distributed multimedia services. Em *1o. Workshop Pronex Sobre Frameworks*, editado por S. Crespo, F. Lima, M. F. Fontoura, F. Porto e S. Colcher, págs. 33–35. Departamento de Informática — PUC-Rio, dezembro 1998. C. J. P. Lucena (Org.), TR MCC51/98.

[24] S. Colcher e L. F. G. Soares. Modelo de referência unificado para arquitetura de protocolos e programação de aplicações multimídia. Em *XVI Simpósio Brasileiro de Redes de Computadores (SBRC'98)*, editado por J. Leite e E. S. Silva, págs. 631–650, Rio de Janeiro, RJ, maio 1998.

[25] S. Colcher, L. F. G. Soares, M. A. Casanova e G. L. Souza. Modelo de objetos baseado no CORBA para sistemas hipermídia abertos com garantias de sincronização. Em *XIV Simpósio Brasileiro de Redes de Computadores (SBRC'96)*, editado por M. Oliveira, págs. 18–37, Fortaleza, CE, maio 1996.

[26] B. Collis. Applications of computer communications in education: An overview. *IEEE Communications Magazine*, 37(3):82–86, março 1999.

[27] J. Communications. Music industry and the internet: Revenue and technology projections in the age of digital distribution. Research Studies, agosto 1999.

[28] J. Communications. Television industry and the internet: Revenue and technology projections in the age of digital distribution. Research Studies, setembro 1999.

[29] G. Cornell e C. S. Horstmann. *Core Java*. Sunsoft Press — Prentice Hall PTR, segunda edição, 1997.

[30] F. M. Costa, G. S. Blair e G. Coulson. Experiments with reflective middleware. Em *Procedings of the ECOOP'98 Workshop on Reflective Object Oriented Programming and Systems*. Springer Verlag, 1998.

[31] G. Coulson. *Multimedia Application Support in Open Distributed Systems*. Tese de Doutorado, Computer Department — Lancaster University, abril 1993.

[32] G. Coulson e G. Blair. Micro-kernel support for continuous media in distributed systems. Relatório Técnico MPG-93-04, Department of Computing, Lancaster University, UK, 1993.

[33] G. Coulson, G. S. Blair, N. Davies e N. Williams. Extensions to ANSA for multimedia computing. *Computer Networks and ISDN Systems*, (25):305–323, 1992.

[34] G. Coulson, G. S. Blair, F. Horn, L. Hazard e J. B. Stefani. Supporting the Real-Time requirements of continuous media in open distributed processing. *Computer Networks and ISDN Systems*, 27(8), 1995.

[35] G. Coulson e D. G. Waddington. A CORBA compliant real-time multimedia platform for broadband networks. Em *Proceedings of TRENDS'96*, Germany, setembro 1996.

[36] S. Deering e D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, maio 1990.

[37] S. Deering, D. L. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu e L. Wei. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, abril 1996.

[38] P. Druschel. Efficient support for incremental customization of OS services. Em *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, págs. 186–190, Asheville, NC, dezembro 1993.

[39] P. Druschel. *Operating System Support for High Speed Networking*. Tese de Doutorado, Department of Computer Science — The University of Arizona, Tucson, agosto 1994.

[40] D. F. D'Souza e A. C. Wills. *Objects, Components and Frameworks with UML: The Catalysis Approach*. Object Technology Series. Addison Wesley, 1998.

[41] S. Ducasse. Message passing abstractions as elementary bricks for design pattern implementation: An experiment. Em *ECOOP'97 Workshops*, editado por J. Bosch e S. Mitchell, Jyvaskyla, Finland, junho 1997. Springer. Published in Lecture Notes in Computer Science (LNCS) 1357, Object-Oriented Technology.

[42] M. Duell. Managing change with patterns. *IEEE Communications Magazine*, 37(4):37–39, abril 1999.

[43] L. Z. et al. *RSVP Functional Specification*. IETF Working Draft, 1995.

[44] EUROSCOM Project P103 — Evolution of the Intelligent Network. *Deliverable No 6a: Framework for Service Description with Supporting Architecture*, fevereiro 1995.

[45] D. C. Feldmeier, A. J. McAuley, J. M. Smith, D. S. Bakin, W. S. Marcus e T. M. Raleigh. Protocol boosters. *IEEE Journal on Selected Areas in Communications*, abril 1998.

[46] W. Fenner. Internet group management protocol, version 2. Internet Draft, janeiro 1997. draft-ietf-idmr-igmp-v2-06.txt.

[47] D. Flinn. Tutorial on building a multimedia web site for e-commerce. Em *Multimedia-Com Conference & Exhibition*, 1999.

[48] M. F. M. C. Fontoura. *A Systematic Approach to Framework Development*. Tese de Doutorado, Computer Science Department — Pontifical Catholic University of Rio de Janeiro, Rua Marquês de São Vicente 225, julho 1999.

[49] M. F. M. C. Fontoura, C. J. P. Lucena, P. S. C. Alencar e D. D. Cowan. On expressiveness: Representing frameworks at design level. Submitted to TAPOS.

[50] B. Ford e S. Sai. CPU Inheritance Scheduling. Em *Operating Systems Design and Implementation (OSDI'96)*, Seattle, outubro 1996.

[51] M. Fowler. *UML Distilled — Applying The Standard Object Modeling Language*. The Addison-Wesley Object Technology. Addison Wesley Longman, Inc., 1997.

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

[53] E. Gamma, R. Helm, R. Johnson e J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[54] D. Garlan e M. Shaw. An introduction to software architecture. Em *Advances in Software Engineering and Knowledge Engineering*, editado por V. Ambriola e G. Tortora, volume I, págs. 1–39, New Jersey, 1993. World Scientific Publishing Company.

[55] D. Garlan e M. Shaw. An introduction to software architecture. Relatório Técnico CMU-CS-94-166, Carnegie Mellon University, janeiro 1994.

[56] A. T. A. Gomes. *Um Framework Para Provisão de QoS Em Ambientes Genéricos de Processamento e Comunicação*. Dissertação de Mestrado, Departamento de Informática — PUC-Rio, maio 1999.

[57] A. T. A. Gomes, S. Colcher e L. F. G. Soares. Um framework para provisão de QoS em ambientes genéricos de processamento e comunicação. Em *XVII Simpósio Brasileiro de Redes de Computadores (SBRC'99)*, págs. 307–322, Salvador, BA, Brasil, junho 1999.

[58] R. Govindan e D. P. Anderson. Scheduiling and IPC mechanisms for continuous media. Em *XIII Symposium on Operating Systems Principles*, págs. 68–80, Pacific Grove, California, 1991.

[59] P. Goyal, X. Guo e H. M. Vin. A hierarchical CPU Scheduler for Multimedia Operating Systems. Em *Operating Systems Design and Implementation (OSDI'96)*, págs. 107–122, Seattle, outubro 1996.

[60] X. Han, M. A. Hiltunen e R. D. Schlichting. Supporting Configurable Real-Time Communication Services. Relatório Técnico TR 97-10, Department of Computer Science – The University of Arizona, Tucson, AZ 85721, maio 1997.

[61] J. Harju, B. Silverajan e I. Toivanen. Experiences in telecommunications protocols with an OO based implementation framework. Em *Procedings of the ECOOP'97 Workshops*, Jyväskylä, Finland, junho 1997. Simon Znaty, Jean-Pierre Hubaux, Springer, Lecture Notes In Computer Science, LNCS 1357.

[62] A. B. de Holanda. *Novo Dicionário Da Lingua Portuguesa*, segunda edição.

[63] K. Houaiss. *Enciclopédia e Dicionário Ilustrado*, 1999.

[64] J. F. Huard e A. A. Lazar. On end-to-end QoS mapping. Em *Proceedings of the IFIP Fifth International Workshop on Quality of Service (IWQoS'97)*, maio 1997.

[65] H. Huni, R. Johnson e R. Engel. A framework for network protocol software. Em *OOPSLA'95*. ACM SIGPLAN Notices, 1995.

[66] N. C. Hutchinson e L. L. Peterson. The *x*-Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, janeiro 1991.

[67] ISO/IEC 7498/ITU-T X.200. *Information Processing Systems – Open Systems Interconnection – Basic Reference Model*, 1984.

[68] ISO/IEC, JTC1/SC21, Document N9309. *QoS: Basic Framework*, 1995.

[69] ISO/IEC JTC1/SC21/WG7. *ISO/IEC DIS 10746-1/ITU-T X.901 — Reference Model of Open Distributed Processing, Part 1: Overview*, maio 1995. output from the editing meeting in Helsink (Finland).

[70] ITU-T Recommendation I.211. *B-ISDN Service Aspects*, novembro 1993.

[71] ITU-T Recommendation I.312/Q.1201. *Principles of Intelligent Network Architecture*, outubro 1992.

[72] ITU-T Recommendation I.324. *ISDN Network Architecture*, 1991.

[73] ITU-T Recommendation M.3010. *Principles of Telecommunications Management Network (TMN)*, dezembro 1991.

[74] G. Kiczales et al. Aspect-oriented programming. Em *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, 1997.

[75] G. Kiczales, J. D. Rivieres e D. Bobrow. *The Art of Metaobject Protocol*. MIT Press, 1991.

[76] E. Knightly e L. Zhang. D-BIND: An acurate TrafficModel for providing QoS. *IEEE/ACM Transactions on Networking*, abril 1997.

[77] V. P. Kompella, J. C. Pascale e G. C. Polyzos. Multicasting routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, junho 1993.

[78] A. Lazar, K. Lim e F. Marconcini. Realizing a foundation for programmability of ATM networks with the binding architecture. *IEEE Journal on Selected Areas in Communications*, 14(7):1214–1227., setembro 1996.

[79] A. A. Lazar. Programming telecommunication networks. *IEEE Network*, págs. 8–18, setembro 1997.

[80] A. A. Lazar, K. S. Lim e F. Marconcini. Binding model: Motivation and description. Relatório Técnico CTR 411-95-17, Comet Group, Department of Electrical Engineering and Center for Telecommunications Research, Columbia University, New York, setembro 1995. http://www.ctr.columbia.edu/comet/xbind/xbind.html.

[81] H. Li, H. K. Pung e L. Ngoh. Active multicast service architecture for user customized multimedia data transmission over ATM networks. Relatório técnico, Multimedia Systems Research Lab., National University of Singapore, 1997.

[82] R. van der Linden. An overview of ANSA. Relatório técnico, ANSA, julho 1993.

[83] C. L. Liu e J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Jornal of the ACM*, 20(1):46–61, janeiro 1973.

[84] C. J. P. Lucena et al. AulaNet: Um Ambiente Para Desenvolvimento e Manutenção de Cursos na WWW. Relatório Técnico 45/97, Pontifícia Universidade Católica do Rio de Janeiro, 1997.

[85] P. Maes. Concepts and experiments in computational reflection. Em *Procedings of OOPSLA'87*, págs. 147–155. ACM Press, 1987. vol. 22 of ACM SIGPLAN Notices.

[86] T. Maufer e C. Semeria. Introduction to IP multicast routing. Internet Draft, março 1997. draft-ietf-mboned-intro-multicast-02.txt.

[87] A. Mauthe, D. Hutchison, G. Coulson e S. Namuye. From requirements to services: Group communication support for distributed multimedia systems. Relatório técnico, Computing Department, Lancaster University, 1995.

[88] G. Meszaros. Design patterns in telecommunications systems architecture. *IEEE Communications Magazine*, 37(4):40–45, abril 1999.

[89] R. Meunier. The pipes and filters architecture. Em *Pattern Languages of Programming Design*, editado por J. Coplien e D. Schmidt, págs. 427–440. Addison Wesley, 1995.

[90] R. T. Monroe, A. Kompanek, R. Melton e D. Garlan. Architectural styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, janeiro 1997.

[91] A. B. Montz, D. Mosberger, S. W. O'Malley, L. L. Petersonand, T. A. Proebsting e J. H. Hartman. Scout: A communications-oriented operating system. Relatório Técnico 94-20, Department of Computer Science, University of Arizona, junho 1994.

[92] D. Mosberger. Message library design notes. x-Kernel Internal Report.

[93] J. Moy. Multicast extensions to OSPF. RFC 1584, março 1994.

[94] K. Nahrstedt e J. M. Smith. End-to-end QoS guarantees: Lessons learned from OMEGA. Relatório Técnico UIUCDCS-R-97-1990, UILU-ENG-97-1704, MONET (Multimedia Operating Systems and Networking Group) – University of Illinois, maio 1996.

[95] P. Nikander, J. Parssinen, B. Sahlin e K. Hoglund. A java based framework for criptographic protocols. Relatório técnico, Telecommunications Software and Multimedia, Helsinki University of Technology, maio 1997. Draft Research Report.

[96] S. Oaks e H. Wong. *Java Threads*. O'Reilly Associates, Inc., 1997.

[97] Object Management Group (OMG). *The Common Object Request Broker: Architecture and Specification*, fevereiro 1998. Revision 2.2.

[98] J. C. Pasquale, G. C. Polyzos e G. Xylomenos. The multimedia multicasting problem. *ACM Multimedia*, 6(1):43–59, 1998.

[99] S. Paul, K. K. Sabnani, L. C. Lin e S. Bhattacharyya. Reliable multicast transport protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 1996. special issue on Network support for Multipoint communication.

[100] V. C. C. de Paula. *ZCL: A Formal Framework for Specifying Dynamic Software Architectures*. Tese de Doutorado, Departamento de Informática — Universidade Federal de Pernambuco, 1999.

[101] K. S. Perumalla e R. M. Fujimoto. A C++ instance of TeD. Relatório Técnico GIT-CC-96-33, College of Computing — Georgia Institute of Technology, Atlanta, GA 30332, 1996.

[102] K. S. Perumalla, R. M. Fujimoto e A. T. Ogielski. MetaTeD — a meta language for modeling telecommunications networks. Relatório Técnico GIT-CC-96-32, College of Computing — Georgia Institute of Technology, Atlanta, GA 30332, 1996.

[103] D. C. Plummer. An ethernet address resolution protocol. Request for Comments 826, novembro 1982.

[104] W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley/ACM Press, 1995.

[105] W. Pree. *Framework Patterns*. SIGS Management Briefings. SIGS Books & Multimedia, 1996.

[106] K. Psounis. Active networks: Applications, security, safety, and architectures. *IEEE Communications Surveys*, (First Quarter):2–16, 1999.

[107] T. Pusateri. Distance vector multicast routing protocol (DVMRP). Internet Draft, fevereiro 1997. draft-ietf-idmr-dvmrp-v3-04.txt.

[108] R. Rao. Implementation reflection in silica. Em *Proceedings of ECOOP'91*, editado por P. America, págs. 251–267. Lecture Notes in Computer Science, Springer-Verlag, 1991.

[109] Rational Software Corporation. *UML Notation Guide – Version 1.1*, setembro 1997.

[110] Rational Software Corporation. *UML Semantics – Version 1.1*, setembro 1997.

[111] M. A. A. Rodrigues. *Um Framework Para Provisão de Serviço de Multicast Em Ambientes Genéricos de Comunicação de Dados*. Dissertação de Mestrado, Departamento de Informática — PUC-Rio, maio 1999.

[112] M. A. A. Rodrigues, S. Colcher e L. F. G. Soares. Um framework para a provisão de serviço de multicast em ambientes genéricos de comunicação de dados. Em *XVII Simpósio Brasileiro de Redes de Computadores (SBRC'99)*, págs. 206–221, Salvador, BA, Brasil, junho 1999.

[113] D. C. Schmidt. The ADAPTIVE communication environment: An Object-Oriented network programming toolkit for developing communication software. Em *Proceedings of the 12th Annual Sun Users Group Conference*, págs. 214–225, San Jose, CA, dezembro 1993. SUG.

[114] D. C. Schmidt. An OO encapsulation of ligtwight OS concurrency mechanisms in the ACE toolkit. Relatório Técnico WUCS-95-31, Department of Computer Science – Washington University, St Louis, 1995.

[115] D. C. Schmidt. Acceptor and connector: A family of object creational patterns for initializing communication services. Em *Proceedings of the 1st European Conference on Pattern Languages of Programming*, Koster, Irsee, germany, julho 1996.

[116] D. C. Schmidt. A family of design patterns for flexibly configuring network services in distributed systems. Em *Proceedings of the International Conference on Configurable Distributed Systems*, Maryland, maio 1996.

[117] D. C. Schmidt. Applying design patterns and frameworks to develop Object-Oriented communication software. Em *Handbook of Programming Languages*, editado por P. Salus, volume I. MacMillan Computer Publishing, 1997.

[118] D. C. Schmidt, D. F. Box e T. Suda. ADAPTIVE: A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment. *Journal of Concurrency: Practice and Experience*, 5:269–286, junho 1993.

[119] D. C. Schmidt, A. Gokhale, T. H. Harrison, D. Levine e C. Cleeland. TAO: a High-Performance ORB endsytem architecture for Real-time CORBA. Document submitted as an RFI response to the OMG Special Interest Group on Real-time CORBA.

[120] D. C. Schmidt, A. Gokhale, T. H. Harrison e G. Parulkar. A High-Performance endsystem architecture for Real-Time CORBA. *IEEE Communications Magazine*, 14(2), fevereiro 1997.

[121] H. Schulzrinne e S. Casner. *RTP: A Transport for Real-Time Applications*. IETF Request for Comments (RFC) 1889, 1996.

[122] A. Silberschatz e P. B. Galvin. *Operating System Concepts*. Addison Wesley, 1995.

[123] L. F. G. Soares, M. A. Casanova e S. Colcher. An architecture for hypermedia systems using MHEG standard objects interchange. *Information Services & Use*, 13(2):131–139, 1993. Special Issue: Hypermedia and Hypertext Standards.

[124] L. F. G. Soares, G. Lemos e S. Colcher. *Redes de Computadores: das LANs, MANs e WANs às Redes ATM*. Ed. Campus, 1995. Segunda Edição.

[125] P. Sridharan. *Advanced Java Networking*. Prentice Hall, 1997.

[126] W. R. Stevens. *Unix Network Programming — Networking APIs: Sockets and XTI*, volume 1. Prentice Hall PTR, segunda edição, 1998.

[127] P. Steyaert. *Open Design of Object-Oriented Languages, A Foundation for Specialisable Reflective Language Frameworks*. Tese de Doutorado, Vrije Universiteit Brussel, 1994.

[128] A. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, Inc., 1992.

[129] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall e G. MInden. A survey of active network research. *IEEE Communications*, janeiro 1997.

[130] D. L. Tennenhouse e D. J. Wetherall. Towards an active network architecture. Em *Keynote Session of Multimedia Computing and Networking Conference*, San Jose, CA, janeiro 1996.

[131] C. Topolcic. *Experimental Internet Stream Protocol Version II (ST-II)*. IETF Request for Comments (RFC) 1190, 1990.

[132] K. J. Turner. An architectural description of intelligent network features and their interactions. *Computer Networks and ISDN Systems*, 30(15):1389–1419, setembro 1998.

[133] G. Utas. An overview of selected call processing patterns. *IEEE Communications Magazine*, 37(4):64–69, abril 1999.

[134] D. G. Waddington, G. Coulson e D. Hutchinson. Specifying QoS for multimedia communications within distributed programming environments. Relatório Técnico MPG-97-34, Distributed Multimedia Research Group, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, 1997.

[135] D. Wetherall, U. Legedza e J. Guttag. Introducing new internet services: Why and how. *IEEE Network*, maio 1998.

[136] D. J. Wetherall, J. V. Guttag e D. L. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. Em *IEEE OPENARCH'98.*, San Francisco, CA, abril 1998.

[137] Y. Yemini e S. D. Silva. Towards programmable networks. Em *Proceedings of the IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, outubro 1996.

[138] S. Znaty e J. Hubaux. Telecommunications services engineering: Principles, architectures and tools. Em *Procedings of the ECOOP'97 Workshops*, Jyväskylä, Finland, junho 1997. Simon Znaty, Jean-Pierre Hubaux, Springer, Lecture Notes In Computer Science, LNCS 1357.

[139] S. Znaty e J. Hubaux. Workshop on object oriented technology for telecommunications services engineering. Em *Procedings of the ECOOP'97 Workshops*, Jyväskylä, Finland, junho 1997. Simon Znaty, Jean-Pierre Hubaux, Springer, Lecture Notes In Computer Science, LNCS 1357.