ALEXSANDRO MATTOS CORRADI

UM *FRAMEWORK* DE SUPORTE A REQUISITOS NÃO-FUNCIONAIS PARA SERVIÇOS DE NÍVEL ALTO

Dissertação submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre. Área de concentração: Processamento Distribuído e Paralelo.

Orientador: Prof. Dr. ORLANDO GOMES LOQUES FILHO

NITERÓI
2005

Dedico essa dissertação aos meus pais,

que não mediram esforços para que eu

pudesse alcançar os meus objetivos.

Agradecimentos

Agradeço principalmente a Deus, por me permitir mais uma vez aqui estar, vivenciando as diversas circunstâncias que nos permite evoluir nas vertentes de crescimento moral e intelectual. Ao amado Mestre Jesus, por me amparar ao longo da caminhada, revitalizando as minhas energias e mantendo o equilíbrio necessário para que os obstáculos pudessem ser superados.

De forma muito especial, agradeço aos meus pais, que sempre me apoiaram, incentivaram e, principalmente, acreditaram na minha capacidade. Serei eternamente grato pela educação que me deram e por não terem medido esforços para que eu pudesse alcançar os meus objetivos. Eu sei o quanto vocês trabalharam para que eu pudesse “vencer” na vida. Muito obrigado, papai e mamãe, por tudo que vocês fizeram por mim.

Agradeço com muito carinho à minha querida irmã, que confiou no meu trabalho e me apoiou ao longo da minha caminhada. Agradeço também aos meus parentes mais próximos que me incentivaram e me apoiaram de alguma forma nessa investida.

Agradeço de forma muito carinhosa à minha querida Monique, por todo apoio, carinho, amor e compreensão. Obrigado por ter estado ao meu lado, me apoiando e me dando força para que eu pudesse alcançar mais essa conquista.

Ao Prof. Orlando Loques pela orientação, paciência, persistência e por acreditar que eu fosse capaz de realizar o trabalho necessário para defender o título de mestre. Obrigado por ter me acompanhado e me incentivado ao longo do mestrado. Sem isso, eu não teria conseguido alcançar o objetivo.

Aos professores do Instituto de Computação da Universidade Federal Fluminense que contribuíram para minha formação. Em especial, aos professores Julius Leite e Alexandre Sztajnberg, pela orientação e dedicação nos trabalhos desenvolvidos. Às secretárias Izabela, Ângela e Maria, pela atenção e apoio no que foi preciso.

Aos professores da Universidade Federal de Ouro Preto, que se empenharam em passar a base necessária para a continuação dos estudos e para o mercado de trabalho. Em particular, aos professores Marcone Jamilson Freitas Souza, Marcelo de Almeida Maia e Álvaro Guarda, que me apoiaram para que eu desse mais esse passo em minha vida.

Aos companheiros de república Glauco, Renathinha, Denis, Euler, Luís, Vanda, Hébio, Leandro (Bixão), Kennedy, Thiago (Facada), Raphael e Renan, pelos momentos de alegria e aprendizado vividos. Ao grande amigo Mokado, pelo companheirismo, atenção, apoio e pelas cuias de chimarrão.

Aos demais amigos do mestrado, Novelli, Allysson, Idalmis, Cris, Vivi, Dani, Rone, André (Totó), Taz, Sidney, Jacques, Bertini, Tiago (Jovem), Aline, Alexandre Senna, Ivairton, Rodrigo Toso, Jonildo, Igor e todos aqueles que comigo conviveram e contribuíram para essa conquista. Em particular, aos amigos Jonivan, Paulo Motta, Marcos Simão, Eduardo Fontana, André (Mokado), Léo e Glauco pela paciência, atenção e cooperação nos trabalhos desenvolvidos.

Bem, eu não sei se consegui agradecer a todos que de alguma forma participaram e contribuíram ao longo do mestrado, mas tenho a certeza que sem o apoio e incentivo de todos, eu não teria concluído com vitória. A todos vocês, meu MUITO OBRIGADO.

“O rio atinge os seus objetivos porque

aprendeu a contornar os obstáculos.”

André Luiz

Índice

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Siglas

ADL	Architecture Description Language
AOP	Aspect-Oritented Programming
API	Application Programming Interface
CBabel	Building Applications by Evolution with Connectors
CCM	CORBA Component Model
CORBA	Common Object Request Broker Architecture
CR-RIO	Contractual Reflective - Reconfigurable Interconnectable Objects
DCOM	Distributed Component Object Model Technologies
EJB	Enterprise JavaBeans
GNP	Global Network Positioning
HQML	Hierarchical QoS Markup Language
JMX	Java Management Extensions
MJPEG	Motion Joint Photographic Experts Group
MPEG	Moving Picture Experts Group
NSSD	Network-Sensitive Service Discovery
NWS	Network Weather Service
ORB	Object Request Broker
PDA	Personal Digital Assistants
QDL	Quality Description Language
QML	QoS Modeling Language
QoS	Quality of Service
QuO	Quality Objects
Q-RAM	QoS-based Resource Allocation Model
Remos	REsource MOnitoring System
SDR	Session Directory Tool
SIP	Session Initiation Protocol
RTP	Real-Time Transport Protocol
RMI	Remote Method Invocation
VoD	Video on Demand
XML	Extensible Markup Language

Resumo

Este trabalho aborda questões relacionadas à especificação e à garantia de requisitos não-funcionais de qualidade (ou de QoS) para sistemas distribuídos baseados em componentes ou serviços. O tema é fundamental em várias áreas de interesse atual, dentre elas a computação ubíqua, a computação autônoma (*autonomic computing*) e projetos de sistemas auto-adaptáveis.

Em particular, tomando como base o *framework* CR-RIO (*Contractual Reflective - Reconfigurable Interconnectable Objects*), que oferece suporte à especificação e ao gerenciamento de contratos de QoS para aplicações, aperfeiçoamos a sua infra-estrutura de suporte, delimitando as responsabilidades de cada um de seus elementos e otimizando a configuração de sua arquitetura. Além disso, consideramos alguns aspectos formais necessários para a interpretação consistente da linguagem utilizada no *framework.*

A nova versão do *framework* CR-RIO foi utilizada no desenvolvimento de exemplos de aplicações, o que permitiu verificar sua utilidade no gerenciamento dos serviços por elas oferecidos.

Abstract

This work tackles issues related to the specification and the guarantee of non-functional requirements of quality (or QoS) for distributed systems based on components or services. The subject is fundamental in some areas of current interest, such as ubiquitous computing, autonomic computing, and auto-adaptable systems.

In particular, starting from the original CR-RIO framework (Contractual Reflective - Reconfigurable Interconnectable Objects), which offers support to the specification and QoS contract management for applications, we refined its support infrastructure, delimiting the responsibilities of its elements and optimizing the configuration of its architecture. In addition, we have considered some formal aspects required for the consistent interpretation of the contract language integrated in the framework.

The new version of the CR-RIO framework was used for the development of some example applications, which allowed us to verify its suitability for the management of the services offered by these applications.

Capítulo 1

Introdução

O cenário no qual os sistemas de softwares são desenvolvidos está se tornando cada vez mais complexo. Isso se deve principalmente ao caráter distribuído das aplicações, onde recursos são compartilhados pelos diversos serviços por elas oferecidos. Além disso, as aplicações requerem que níveis de qualidade de serviço (QoS) – como disponibilidade de um recurso, confiabilidade, segurança, sincronização e requisitos de tempo real – sejam satisfeitos para garantir que seus serviços possam ser oferecidos.

Aplicações nesse contexto têm que ter a capacidade de lidar com mudanças na disponibilidade dos recursos compartilhados e ter a garantia que os requisitos para o funcionamento dos seus serviços sejam satisfeitos de forma transparente. Por exemplo, em uma aplicação de vídeo sob demanda (VoD), onde um servidor multimídia fornece vídeos para clientes através da rede, a banda passante disponível no canal de comunicação e a utilização do processador do cliente podem variar ao longo da execução da aplicação. Assim, torna-se necessário garantir que requisitos mínimos associados a esses recursos sejam atendidos, possibilitando o funcionamento dos serviços da aplicação de forma eficiente. Caso contrário, adaptações na infra-estrutura de suporte à aplicação ou nela própria podem ser efetuadas com o intuito de garantir esses requisitos [Ansaloni 2003, Karr et al. 2001]. Um exemplo seria, quando a disponibilidade da banda passante degradar, transcodificar o vídeo sendo transmitido para um formato de pior qualidade que exija menos desse recurso. Poderia também ser aumentada a prioridade do processo no cliente responsável pela reprodução do vídeo, caso o seu processador esteja sobrecarregado.

Com o intuito de oferecer suporte ao desenvolvimento e implantação dessas aplicações atendendo os requisitos de adaptação dinâmica, manutenibilidade e QoS, algumas propostas atuais de *middlewares* utilizam-se de técnicas, isoladas ou em conjunto, tais como reflexão, notificação de eventos e arquitetura de meta-nível [Agha 2002, Kon et al. 2002, Tripathi 2002, Venkatasubramanian 2002]. Em [Tripathi 2002], por exemplo, é proposto um *framework* conceitual que envolve arquiteturas de *middleware* utilizando essas técnicas para suportar o desenvolvimento de aplicações distribuídas com esses requisitos.

Uma outra abordagem que contribui para a concepção de sistemas de softwares com a capacidade de adaptação dinâmica é baseada no princípio de Arquitetura de Software [Shaw et al. 1995]. Esse princípio considera que um sistema pode ser desenvolvido a partir de sua organização como uma composição de componentes e interações entre eles, em vez de ser desenvolvido a partir da especificação de algoritmos e estruturas de dados. Essa abordagem permite a reutilização dos módulos do sistema e incentiva a separação de interesses [Curty 2002], permitindo separar os requisitos funcionais, que representam a funcionalidade básica de uma aplicação, dos requisitos não-funcionais, ou de QoS, tais como distribuição e tolerância à falhas. Assim, é possível que um sistema de software seja concebido através da sua descrição arquitetural, que inclui os componentes funcionais, as interações entre eles e os requisitos não-funcionais [Carvalho 2001]. Isso possibilita a criação de um ambiente que permite a implantação e execução de um sistema, seguindo as informações contidas em sua configuração arquitetural (seus módulos e as instruções para instanciá-los e conectá-los) descrita em uma linguagem de descrição de arquitetura (ADL – *Architecture Description Language*) [Lisbôa 2003]. Há também a possibilidade de descobrir a arquitetura do sistema dinamicamente através de uma estrutura de meta-nível que ofereça suporte a isso [Yan et al. 2004].

Nesse contexto é necessário uma maneira de especificar as restrições de QoS, permitindo definir os recursos mínimos necessários para que os serviços de uma aplicação possam ser oferecidos [Frolund e Koistinen 1998, Aagedal e Ecklund 2002, Becker e Geihs 1997]. Além disso, é necessário que esses recursos possam ser monitorados e gerenciados em tempo de execução, o que possibilita detectar quando os níveis de QoS não são mais satisfeitos. Nesse caso, pode ser efetuada uma tentativa de adaptação dinâmica para que os serviços continuem a ser oferecidos. Uma maneira de especificar essas restrições é através de um contrato de QoS, que expressa um relacionamento formal entre duas ou mais partes que utilizam ou forneçam recursos [Beugnard et al. 1999]. Em um contrato podem ser definidos

os direitos e as obrigações dos participantes e as regras de adaptação e negociação, que consistem das ações a serem tomadas quando ocorrem mudanças na disponibilidade dos recursos que violam os níveis de qualidade declarados no contrato [Curty 2002, Loyall et al. 1998].

Vários *frameworks* vêm sendo propostos para dar suporte ao desenvolvimento de aplicações distribuídas que oferecem serviços com qualidade diferenciada [Curty 2002, Loques e Sztajnberg 2004, Loyall et al. 1998, Cheng et al. 2004, Wang et al. 2001, Nahrstedt et al. 2001]. Eles permitem que os requisitos de QoS sejam especificados e gerenciam a adaptação do sistema de acordo com a variação na disponibilidade dos recursos monitorados. Particularmente, em [Curty 2002] é proposto o *framework* CR-RIO (*Contractual Reflective - Reconfigurable Interconnectable Objects*), que foi projetado para permitir a especificação e o gerenciamento de contratos de QoS associados aos componentes da arquitetura de software de uma aplicação.

O objetivo deste trabalho é refinar o CR-RIO através de modificações da sua linguagem de descrição de contratos e da sua infra-estrutura de suporte. As modificações na linguagem têm como objetivos a simplificação de sua sintaxe, resultando em uma linguagem mais legível, e a resolução de alguns aspectos formais necessários para sua interpretação consistente. Já na infra-estrutura de suporte, são reavaliadas as responsabilidades de cada componente e proposta uma otimização na configuração da sua arquitetura. Para validar as modificações realizadas, são descritos alguns exemplos de casos de uso que permitem demonstrar o funcionamento da nova versão do *framework*. A implementação do CR-RIO é então proposta, e através de uma aplicação exemplo foi possível avaliar o seu funcionamento e desempenho no gerenciamento de aplicações com requisitos de QoS, constatando a sua viabilidade. Para a elaboração deste trabalho, foi feito inicialmente um estudo do CR-RIO e da sua linguagem de descrição de arquiteturas com suporte a contratos de QoS. Foi feito também um estudo de comparação com outras propostas correlatas, o que nos permitiu identificar e aperfeiçoar alguns pontos ainda não resolvidos desse *framework*.

1.1 Organização do Trabalho

Os capítulos restantes deste trabalho estão organizados da seguinte forma:

- Capítulo 2, descreve algumas questões envolvidas no gerenciamento de aplicações com requisitos de QoS, mostrando, de uma forma geral, a arquitetura de uma infra-estrutura de suporte a esse fim;
- Capítulo 3, descreve o *framework* CR-RIO e apresenta a proposta de refinamentos na sua linguagem de descrição de contratos e na sua infra-estrutura de suporte a gerenciamento de aplicações com requisitos não-funcionais;
- Capítulo 4, apresenta exemplos de casos de uso que permitem exemplificar e validar os refinamentos propostos no *framework* CR-RIO;
- Capítulo 5, expõe alguns aspectos da implementação do *framework* CR-RIO e mostra, através de um exemplo de aplicação, o seu funcionamento e sua viabilidade no gerenciamento de aplicações com requisitos não-funcionais;
- Capítulo 6, apresenta alguns trabalhos correlatos estudados, comparando-os com o CR-RIO;
- Capítulo 7, apresenta a conclusão deste trabalho e descreve alguns trabalhos futuros.

Capítulo 2

Gerenciamento de Aplicações com Requisitos Não-Funcionais

Este capítulo apresenta o contexto de aplicações com requisitos não-funcionais (ou requisitos de QoS) e discute o suporte requerido para o gerenciamento dessas aplicações. Inicialmente é discutida a motivação para o desenvolvimento de suporte a gerenciamento desse tipo de aplicação. Depois é apresentada uma infra-estrutura geral de suporte concebida com o intuito de identificar as atividades envolvidas no provimento desse tipo de gerenciamento. É mostrado que essas atividades se refletem em elementos com funções definidas, os quais interagem no processo de gerenciamento de aplicações com requisitos de QoS, permitindo constatar um modelo padrão para esse processo. Por fim, são apresentadas em maiores detalhes cada uma das atividades do processo de gerenciamento.

2.1 Motivação

O funcionamento de sistemas em várias áreas de interesse atual, dentre elas a computação ubíqua, a computação autônoma (*autonomic computing*), contextos de Internet, grades computacionais e o projeto de sistemas auto-reparáveis ou auto-adaptáveis, depende de requisitos não-funcionais de qualidade que se sobrepõem aos requisitos funcionais. Em uma aplicação no contexto de grades computacionais, por exemplo, pode ser necessário que requisitos relacionados com a capacidade de processamento das máquinas pertencentes à grade, como o poder de processamento e a disponibilidade dos processadores, satisfaçam restrições de QoS exigidas pelos serviços da aplicação. Além disso, os requisitos por recursos

de processamento podem variar ao longo do tempo, tornando necessário o monitoramento dos níveis desses recursos, para que seja possível verificar se ainda satisfazem as restrições exigidas pela aplicação. Perante uma variação nos níveis dos recursos monitorados, pode ser necessário que alguma ação seja efetuada, tentando adaptar a aplicação ao novo estado do sistema ou encerrando-a, caso nenhuma outra possibilidade seja possível.

Um outro exemplo de aplicação com requisitos não-funcionais é um sistema cliente-servidor Web, onde vários clientes enviam requisições de documentos a servidores, que processam essas requisições e enviam o documento requisitado diretamente para o cliente. Nessa aplicação, um requisito de qualidade importante é a banda passante disponível entre o cliente e o servidor. Caso em um determinado momento a banda passante fique congestionada, seria conveniente que o cliente pudesse ser conectado a um outro servidor cujo canal de comunicação esteja atendendo o nível de qualidade desejado.

Em certas aplicações, o gerenciamento dinâmico dos seus serviços, de acordo com os níveis dos recursos por elas utilizados, pode implicar em economia financeira ou em otimização desses recursos. Por exemplo, em uma aplicação onde há mais de uma opção para o estabelecimento da comunicação entre dois pontos, como via rede sem fio (*wireless*) ou via conexão discada, cada opção possui capacidade e custos diferentes. Assim, a opção de menor custo poderia ser a preferencial, e quando ela se tornar indisponível, a opção de maior custo seria utilizada até que a preferencial pudesse ser novamente estabelecida. Ou seja, a aplicação se adaptaria dinamicamente de acordo com as opções disponíveis, prezando pela opção de menor custo.

Com o intuito de gerenciar aplicações com requisitos não-funcionais, de forma que o sistema possa se adaptar dinamicamente perante variações nos níveis dos recursos utilizados, é necessária uma infra-estrutura que ofereça suporte a esse tipo de gerenciamento. Ela deve permitir especificar as restrições de qualidade desejadas por uma aplicação e as configurações de sistema necessárias para implantar ou adaptar os serviços por ela oferecidos. Além disso, é preciso que essa infra-estrutura suporte o monitoramento dos recursos utilizados pela aplicação, a busca por novos recursos necessários e o gerenciamento de adaptações no sistema, quando se fizerem necessárias.

2.2 Infra-estrutura Geral de Suporte a Gerenciamento de Aplicações com Requisitos Não-Funcionais

Uma infra-estrutura de suporte a gerenciamento de aplicações com requisitos não-funcionais pode ser concebida, de uma forma geral, pelas atividades que se refletem nos seguintes elementos:

- **Especificação**, descreve os requisitos não-funcionais de uma aplicação e as configurações necessárias para o estabelecimento dos serviços da aplicação com a qualidade desejada;
- **Monitoramento**, mecanismos que efetuam a monitoração dos níveis dos recursos utilizados pela aplicação;
- **Gerência**, responsável por gerenciar os serviços oferecidos por uma aplicação, levando em consideração os níveis dos recursos e o nível de qualidade desejado;
- **Configuração**, encarregado de efetuar as configurações necessárias no sistema para suportar os serviços oferecidos pela aplicação, assim como a alocação dos recursos a serem utilizados.

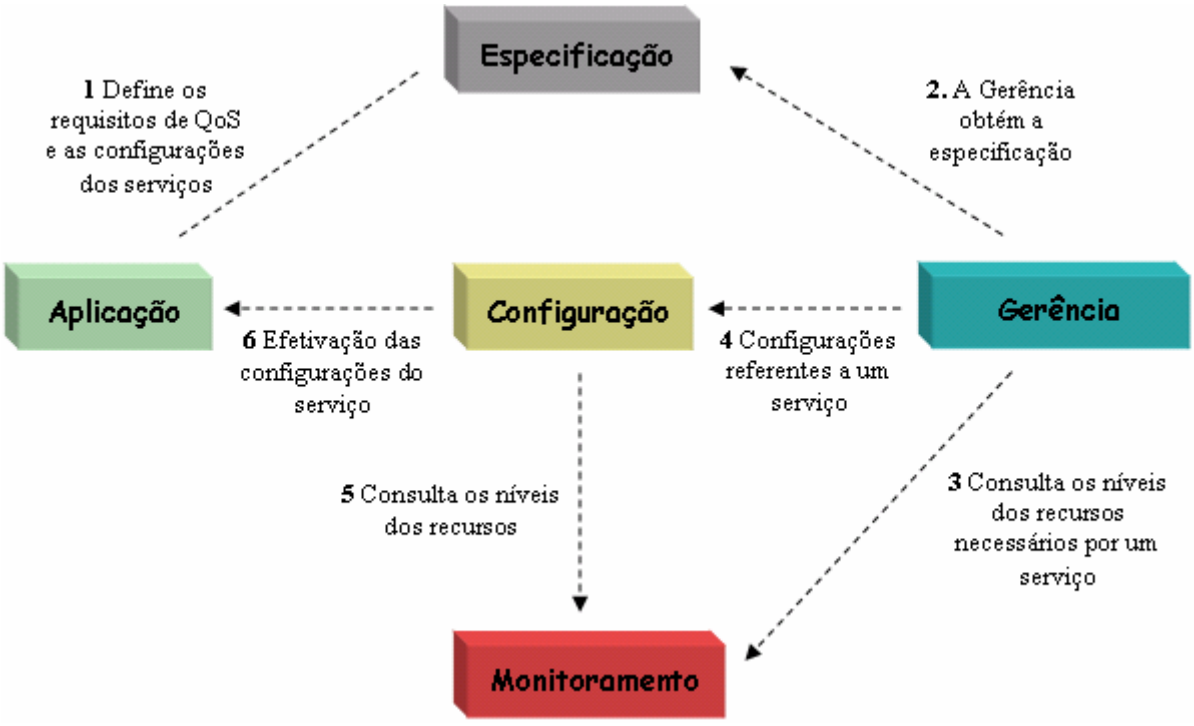

Figura 2.1 – Infra-estrutura geral de suporte a gerenciamento de aplicações com requisitos de QoS.

O diagrama da Figura 2.1 ilustra a interação entre os elementos da infra-estrutura geral de suporte responsáveis pelas atividades envolvidas no processo de gerenciamento de aplicações com requisitos de QoS. Para facilitar o entendimento do seu funcionamento, considere o exemplo cliente-servidor Web citado na seção 2.1. No elemento Especificação seriam descritos os requisitos de QoS desejados para os serviços desta aplicação e as configurações necessárias para implantar ou adaptar os serviços oferecidos (passo 1, na Figura 2.1). Uma restrição, por exemplo, seria que a banda passante disponível no canal de comunicação entre o cliente e o servidor deva ser maior que 2 Mbps. Ao iniciar o gerenciamento de uma aplicação, de acordo com seus requisitos não-funcionais, o elemento Gerência consulta os requisitos de QoS da aplicação em Especificação (passo 2), para o exemplo em questão, o requisito do canal de comunicação. Conhecendo os requisitos da aplicação, Gerência precisa obter informações sobre os níveis dos recursos a serem utilizados no elemento Monitoramento (passo 3), no caso do exemplo, seria feita uma consulta a um mecanismo que permitisse medir a banda passante disponível entre o cliente e os servidores. Caso seja constatado que o nível do recurso satisfaz a restrição de QoS para a aplicação, Gerência passa as configurações do serviço a ser estabelecido, também obtidas de Especificação, para o elemento Configuração (passo 4). No exemplo, as configurações orientariam a ligação do cliente com um determinado servidor. O elemento Configuração, então, mapeia as informações contidas nas configurações recebidas em ações de sistema que efetivarão a configuração do serviço da aplicação (passo 6). Nesse processo pode ser necessário consultar o elemento Monitoramento, a fim de obter alguma informação sobre os recursos do sistema, por exemplo, a informação sobre o servidor que possui mais recursos de processamento disponíveis (passo 5). Neste diagrama foi assumido que os recursos de sistema são conhecidos. Contudo, em uma visão mais refinada poderia ser incluído um mecanismo de descoberta de recursos, que poderia ser responsável, por exemplo, pela descoberta de servidores disponíveis na rede.

Depois que o serviço da aplicação já está em execução, o elemento Gerência coordena a monitoração dos recursos utilizados para verificar se as restrições de QoS da aplicação estão sendo satisfeitas. No caso do exemplo cliente-servidor, se em um determinado momento a banda passante fica congestionada, violando os requisitos da aplicação, Gerência coordenará as ações necessárias para tentar manter a aplicação em funcionamento. Por exemplo, ele poderá solicitar ao elemento Configuração que procure um outro servidor onde o cliente possa ser conectado, respeitando os requisitos de qualidade exigidos para o canal comunicação.

As próximas seções descrevem melhor cada um dos elementos da infra-estrutura geral de suporte a aplicações com requisitos não-funcionais.

2.2.1 Especificação

Como uma evolução natural da tecnologia de desenvolvimento de sistemas distribuídos através de composição de componentes, torna-se atrativo o desenvolvimento desses sistemas através de composição de serviços implementados a partir de componentes. Na primeira fase do desenvolvimento baseado em componentes, a preocupação básica concentrou-se em mecanismos de definição de interfaces (de programação e comunicação) e de encapsulamento de componentes de maneira a permitir que eles fossem interligados e configurados para atender os requisitos funcionais das aplicações de interesse (e.g., CORBA, Java/RMI, DCOM/.NET). Na fase atual, onde a tecnologia básica de componentes apresenta-se mais madura (e.g., EJB, CCM, *Web Services*), a preocupação concentra-se em caracterizar e especificar os diversos aspectos não-funcionais (ou operacionais), e.g., desempenho e disponibilidade, que podem ser associados aos serviços ou componentes. Esta especificação fornece os fundamentos necessários para o estabelecimento de responsabilidades individuais e mútuas desses componentes e para estabelecer a infra-estrutura de suporte, visando a garantia de requisitos de qualidade dos serviços oferecidos por uma aplicação.

A especificação de aspectos não-funcionais, ou de QoS, pode se apresentar em vários formatos [Jin e Nahrstedt 2004], seja através de linguagens de descrição [Loques et al. 2004, Pal et al. 2000, Duran-Limon e Blair 04, Gu et al. 2002, Lamanna et al. 2003, Florissi e Yemini 1994], seja via regras e funções de associação, como em [Li e Nahrstedt 1999], ou mesmo por meio de funções de utilidade de recursos, como em Q-RAM (*QoS-based Resource Allocation Model*) [Rajkumar et al. 1997].

Há várias propostas de linguagens para descrição de serviços [Loques et al. 2004, Duran-Limon e Blair 04, Lamanna et al. 2003], incluindo seus aspectos não-funcionais, essenciais para a composição de aplicações a partir de componentes mais primitivos. A descrição de um serviço contém as informações referentes às configurações necessárias para que ele possa ser estabelecido. O uso destas linguagens permite a elaboração de contratos, os quais visam formalizar as exigências básicas para a instalação e operação desses serviços. Os contratos definem perfis de qualidade – que especificam os níveis desejados dos requisitos não-funcionais – a serem estabelecidos ou mantidos, permitindo o conhecimento das

propriedades não-funcionais que devem ser monitoradas durante a etapa de funcionamento de uma aplicação.

Portanto, para que seja possível oferecer suporte a gerenciamento de aplicações com requisitos não-funcionais, torna-se apropriado o uso de um mecanismo que permita descrever os requisitos de QoS desejados e as configurações necessárias para o estabelecimento dos serviços oferecidos. É essa descrição que possibilita conhecer os níveis de qualidade desejados por uma aplicação, orientar o monitoramento dos recursos por ela utilizados e determinar as configurações necessárias para o estabelecimento dos serviços por ela oferecidos.

2.2.2 Monitoramento

Na medida em que requisitos de qualidade são adicionados às aplicações, vem ganhando cada vez mais importância a monitoração de parâmetros de elementos da infra-estrutura de suporte a serviços. Essa monitoração permite verificar se o estado do sistema satisfaz os requisitos mínimos para que um serviço de uma aplicação possa ser iniciado com a qualidade desejada. Além disso, para muitas aplicações, uma degradação no nível de QoS oferecido ao consumidor pode tornar o serviço inutilizável. Assim, é de interesse dos participantes do serviço monitorar o nível de QoS para poder garantir a entrega do mesmo, ou para poder adaptar a aplicação ou a infra-estrutura de suporte de forma a ser possível operar sobre diferentes níveis de qualidade de serviço. Nesse sentido, conforme discutido na seção anterior, contratos podem ser usados pelas aplicações para informar às entidades participantes o nível de QoS requisitado e, adicionalmente, algumas estratégias de adaptação a serem usadas.

A tarefa de monitoração consiste na coleta de métricas como as de desempenho, por exemplo, latência, largura de banda, *jitter*, tamanho de *buffers*, % de uso de CPU, necessárias para verificar se os níveis de QoS requeridos são atendidos, na implantação de um serviço e ao longo de seu funcionamento. Tal monitoração freqüentemente deve ser realizada com a ajuda de uma entidade externa, de forma que os resultados sejam confiáveis tanto para o provedor de recurso quanto para o usuário do mesmo. Várias plataformas, ferramentas, mecanismos e APIs foram propostas nos últimos anos, tais como:

- **NWS** (*Network Weather Service*), que provê um serviço de monitoração e previsão de características de desempenho dos recursos de um sistema distribuído, tais

como carga e latência de rede [Wolski et al. 1999]. O NWS pode adquirir várias métricas de desempenho: carga da CPU de uma máquina, uso de memória, tempo de conexão TCP, latência de rede e largura de banda disponível entre dois pontos;

- **Remos** (*REsource MOnitoring System*), permite que aplicações distribuídas obtenham informações sobre os recursos do ambiente de rede onde elas se encontram [Dinda et al. 2001];
- **GNP** (*Global Network Positioning*), que propõe o uso de um mecanismo baseado em coordenadas, computadas a partir de uma modelagem da Internet como um espaço geométrico, para predizer distâncias de rede (e.g., atraso e *round trip time* – RTT) [Eugene e Zhang 2002].

No contexto de gerenciamento de aplicações com requisitos não-funcionais, essas ferramentas são utilizadas para prover as informações necessárias sobre os níveis dos recursos utilizados por uma aplicação. Com base nessas informações é feita a verificação se os níveis dos recursos satisfazem os requisitos de QoS desejados, descritos na especificação.

2.2.3 Gerência

A gerência de uma aplicação com requisitos não-funcionais consiste em coordenar as atividades inerentes ao processo de estabelecimento dos serviços por ela oferecidos, respeitando os níveis de qualidade desejados. Para isso, é preciso primeiramente identificar os parâmetros de qualidade descritos em Especificação. É necessário também gerenciar o monitoramento dos recursos utilizados, obtendo as informações referentes aos seus estados, de forma a validar os parâmetros de qualidade especificados. Para que seja possível a uma aplicação se adequar a variações no estado do sistema, é preciso ainda, coordenar um mecanismo que permita configurar a infra-estrutura de suporte do sistema ou até mesmo a própria aplicação.

A maioria dos mecanismos de gerenciamento existentes é dirigida a requisitos específicos e localizados, que buscam impor as propriedades contratadas por meio de procedimentos codificados internamente, como comumente observado nos serviços e protocolos de comunicação. Contudo, torna-se atrativo a utilização de um mecanismo externo de gerenciamento, com baixa interferência na programação dos elementos funcionais, e que possa ser reutilizado em diversas aplicações. A generalização da capacidade desse gerenciamento em sistemas distribuídos envolve diversos conceitos e mecanismos que têm

que ser considerados em conjunto para superar a complexidade inerente da tarefa. Neste contexto, uma das questões relevantes é a capacidade de realizar adaptações dinâmicas na arquitetura de uma aplicação, visando dar continuidade à provisão dos serviços por ela oferecidos.

Com o objetivo de prover o gerenciamento de aplicações com requisitos de QoS, algumas propostas vêm sendo desenvolvidas [Ansaloni 2003, Loyall et al. 1998, Garlan et al. 2004, Wang et al. 2001, Moura et al. 2002, Florissi 1996]. Em comum, essas propostas permitem especificar os níveis de qualidade desejados, determinar as alterações a serem feitas quando esses níveis forem violados, usam mecanismos de monitoração de recursos sobre os quais os níveis de qualidade são declarados e incluem um suporte que permite que adaptações sejam efetuadas no sistema quando se fizerem necessárias.

2.2.4 Configuração

Para lidar com o requisito de adaptação dinâmica de uma aplicação perante mudanças no estado dos recursos por ela utilizados, é conveniente o uso de algum mecanismo que ofereça esse suporte. Ele deve permitir o gerenciamento da aplicação de forma que ela ou a infra-estrutura de sistema que a suporta possam ser dinamicamente alteradas, sem a necessidade de parar os serviços por ela oferecidos.

Algumas tecnologias, como *Java Management Extensions* (JMX), programação orientada a aspecto (AOP, do inglês *Aspect-Oritented Programming*) e reflexão computacional, vêm sendo utilizadas no desenvolvimento de *frameworks*, *middlewares* e servidores que oferecem suporte à implementação de adaptação dinâmica de aplicação. Por exemplo, a implementação do núcleo do JBoss – um extensível, com suporte a reflexão e dinamicamente configurável servidor de aplicação Java – baseia-se em JMX, para prover a capacidade de configuração dinâmica de seus componentes [Fleury e Reverbel 2003]. Outro exemplo é o Javassist (*Java Programming Assistant*), que utiliza reflexão computacional para permitir que classes de uma aplicação Java sejam modificadas, antes de serem carregadas pela máquina virtual Java [Chiba 2000].

Além dessas tecnologias, com o objetivo de oferecer suporte aos requisitos de adaptação dinâmica, algumas abordagens utilizam técnicas como reflexão, notificação de eventos e arquitetura de meta-nível [Agha 2002, Kon et al. 2002, Tripathi 2002, Venkatasubramanian 2002]. Em [Kon et al. 2002] por exemplo, é discutida a necessidade de

middlewares que lidam com ambientes altamente dinâmicos, suportando flexibilidade e capacidade de adaptação. É mostrado como esse tipo de suporte pode ser provido através do mecanismo de reflexão. Dois projetos de sistema de *middleware* reflexivo são apresentados como exemplo: DynamicTAO e Open ORB. O primeiro é uma extensão do C++ TAO ORB, suportando configuração dinâmica dos componentes do *middleware*, controlando concorrência, segurança e monitoração. Open ORB oferece suporte a projetos de plataformas de *middlewares* dinamicamente configuráveis para suportar aplicações com requisitos dinâmicos.

Uma linha alternativa nesse contexto adota o modelo de Arquitetura de Software [Oreizy et al. 1999], que considera que uma aplicação pode ser manipulada a partir de um nível arquitetural, onde são representados seus componentes e as interligações entre eles [Shaw et al. 1995]. Essa abordagem incentiva a separação de interesses, facilitando a separação dos requisitos funcionais da aplicação dos não-funcionais. Assim, é possível que um sistema de *software* seja concebido a partir da sua descrição arquitetural, que inclui os componentes funcionais, as interações entre eles e os requisitos não-funcionais. Essa descrição provê uma visão da arquitetura da aplicação, o que facilita a sua adaptação. Baseado neste conceito, foi proposto em [Lisbôa 2003] um ambiente de suporte a configuração de arquitetura de *software*. O ambiente possui uma estrutura que consegue, a partir da interpretação de comandos presentes na descrição arquitetural, escrita em uma linguagem de descrição de arquitetura (ADL), obter informações sobre a configuração arquitetural do sistema. A partir dessas informações, o ambiente instancia os módulos e conectores do sistema, realizando as ligações entre eles e efetivando a execução do sistema.

Uma visão mais recente relacionada com a capacidade de adaptação dinâmica de sistemas é a computação autônoma [Horn 2001]. Ela considera que um sistema pode ser formado por um conjunto de elementos autônomos interconectados, onde cada um é capaz de se autogerenciar (monitorar o comportamento do sistema e adaptar-se quando necessário) [Ganek e Corbi 2003]. Um exemplo de aplicação desta abordagem é apresentado em [Melcher e Mitchell 2004], onde são discutidas questões sobre o gerenciamento dinâmico de serviços de rede de forma autônoma.

Ainda no que diz respeito ao processo de configuração de um sistema, pode ser necessária a utilização de mecanismos que permitam localizar serviços que são oferecidos no ambiente distribuído de uma aplicação. Por exemplo, a busca por um servidor que satisfaça os requisitos funcionais (e.g., o provimento de um determinado serviço) e não-funcionais (e.g.,

atraso de rede e capacidade de processamento) de uma aplicação. Um exemplo de sistema que oferece esse suporte é o *network-sensitive service discovery* (NSSD) [Huang e Steenkiste 2003], que permite a procura e a seleção de um conjunto de serviços que satisfaça um conjunto de propriedades funcionais e não-funcionais desejadas.

Além dos trabalhos supracitados, várias outras propostas de *frameworks* vêm sendo desenvolvidas com o intuito de prover suporte a adaptação dinâmica em um sistema de *software* [Moreira et al. 2003, Yahiaoui et al. 2004, Dowling e Cahill 2001, Keeney e Cahill 2003].

2.3 Conclusão do Capítulo

Este capítulo mostrou a motivação para o desenvolvimento de suporte a aplicações com requisitos não-funcionais e uma infra-estrutura geral que permite gerenciar aplicações com esses requisitos. Foi mostrado que uma infra-estrutura para esse fim segue um modelo geral composto por quatro elementos principais: Especificação, Monitoramento, Gerência e Configuração. Através desses elementos é possível especificar os níveis de qualidade desejados pelos serviços oferecidos por uma aplicação, especificar as configurações necessárias para que esses serviços possam ser implantados ou adaptados, monitorar os recursos utilizados pela aplicação para que seja possível verificar a observância dos requisitos de qualidade, e gerenciar todo o processo de estabelecimento e adaptação dos serviços, de acordo com variações nos níveis de QoS monitorados.

O próximo capítulo descreve a proposta CR-RIO, um *framework* que permite gerenciar aplicações com requisitos de QoS, descritos através de contratos no nível arquitetural de *software*.

Capítulo 3

A Proposta CR-RIO

O *framework* CR-RIO (*Contractual Reflective - Reconfigurable Interconnectable Objects*) [Curty 2002, Ansaloni 2003, Loques et al. 2004] consiste em uma infra-estrutura projetada para suportar a especificação e o gerenciamento de contratos de QoS para aplicações. Os contratos permitem especificar os serviços de uma aplicação, em nível arquitetural de abstração, e associá-los a requisitos não-funcionais que expressam os níveis desejados de qualidade. Esses requisitos são monitorados no sistema e quando um nível desejado não pode ser mais atendido o *framework* tenta estabelecer um outro serviço cuja qualidade seja compatível com os níveis atuais.

O CR-RIO utiliza o conceito de Arquitetura de Software centrado no uso de uma ADL (linguagem de descrição de arquitetura) que permite a descrição da configuração arquitetural de uma aplicação, em termos de seus módulos e a interligação entre eles, associada à descrição de um contrato de QoS que define os níveis de qualidade desejados pelos serviços da aplicação. A meta informação contida nessas descrições é usada para orientar as adaptações na configuração da aplicação ou na infra-estrutura de suporte a ela, quando variações nos níveis de qualidade dos recursos utilizados violarem os níveis aceitáveis declarados no contrato. O gerenciamento do contrato é realizado automaticamente segundo um padrão arquitetural que pode ser diretamente mapeado em componentes específicos inclusos no ambiente de suporte, permitindo ao projetista escrever um contrato e seguir uma receita padrão para inserir o código adicional requerido para a sua implantação no ambiente de suporte.

Tomando como base a proposta original do CR-RIO [Curty 2002] e alguns refinamentos propostos [Ansaloni 2003], modificações na linguagem de descrição de contrato e na infra-estrutura de suporte do *framework* são sugeridas neste trabalho. As modificações na linguagem visam simplificar sua sintaxe, resultando em uma linguagem mais legível, e resolver alguns aspectos formais necessários para sua interpretação consistente. Na infra-estrutura de suporte, foram reavaliadas as responsabilidades de cada componente e proposta uma otimização na sua configuração. As modificações realizadas serão explicadas em mais detalhes ao longo deste capítulo.

As seções deste capítulo descrevem o *framework* CR-RIO contemplando as modificações propostas. Inicialmente é apresentada a ADL utilizada para descrever a arquitetura de software de uma aplicação e o contrato de QoS que permite especificar os níveis de qualidade requeridos pelos serviços por ela oferecidos. Em seguida é mostrada a infra-estrutura de suporte do CR-RIO e o seu funcionamento no gerenciamento de aplicações com requisitos não-funcionais.

3.1 A ADL CBabel

CBabel (*Building Applications by Evolution with Connectors*) [Sztajnberg 2002] é a ADL utilizada pelo CR-RIO para descrever os componentes funcionais de uma aplicação e a topologia de interconexão desses componentes. Ela também permite descrever contratos especificando interesses não-funcionais associados aos componentes da aplicação. Como outras ADLs, CBabel provê mecanismos básicos para suportar extensibilidade de software e re-configurações arquiteturais planejadas.

Uma extensão da ADL CBabel propõe uma nova forma de descrever os contratos para os aspectos não-funcionais de uma aplicação [Curty 2002]. Nessa extensão, um contrato de QoS é descrito na forma de serviços, especificados em nível arquitetural, associados às restrições de qualidade requeridas pela aplicação. O contrato também possui uma cláusula de negociação que define uma máquina de estado, onde cada estado representa um serviço e as transições definem as possibilidades de negociação de serviços, de acordo com a disponibilidade nos recursos requeridos por eles. Em [Ansaloni 2003] CBabel foi estendida, refinando essa forma de descrição de contrato através do emprego de alguns conceitos da linguagem QML (*QoS Markup Language*) [Frolund e Koistinen 1998], com o intuito de facilitar a descrição de contratos de serviços não-funcionais para aplicações.

A descrição desses contratos é feita de forma separada dos componentes funcionais da aplicação. Essa abordagem incentiva a separação de interesses (ou aspectos), o que facilita o reuso dos módulos que implementam a computação em outros sistemas. Uma especificação CBabel corresponde a uma meta-descrição de uma aplicação que fica disponível em um repositório, de modo que enquanto a aplicação está em operação, sua meta-descrição oferece informações que facilitam o gerenciamento de adaptações arquiteturais.

3.1.1 Descrição Arquitetural

O *framework* CR-RIO explora o conceito de Arquitetura de Software pelo uso da ADL CBabel, que permite descrever a arquitetura de uma aplicação, em termos de seus elementos arquiteturais e topologia de interconexão entre eles. De forma geral, existem três tipos de elementos fundamentais em uma arquitetura de software [Loques et al. 2000]:

- Módulos

São as unidades primárias de computação que implementam os requisitos funcionais de uma aplicação. No nível de implementação, um módulo corresponde a uma unidade de código ou dados identificável, tais como processo, objeto ou arquivo, ou ainda, no nível de linguagens de programação, correspondem a módulos de programação, classes, objetos ou um conjunto de funções. Em geral, um módulo possui uma interface, através da qual pode fornecer ou requerer serviços. No nível de uma linguagem de programação orientada a objeto, por exemplo, os métodos públicos definidos podem representar os serviços oferecidos e invocações a métodos externos os serviços requeridos.

- Portas

São os elementos da arquitetura que definem os pontos lógicos de interação entre os componentes da arquitetura (módulos e conectores). O conjunto de tais pontos de interação representa a interface do componente. As portas podem ser do tipo de saída ou de entrada. Portas de saída representam serviços que podem ser requisitados pelo módulo (invocações de métodos) e portas de entrada os serviços fornecidos pelo módulo (métodos públicos definidos).

- Conectores

São responsáveis por governar a interação entre os módulos da arquitetura. Em princípio, eles encapsulam a funcionalidade necessária à interação, sem que seja necessário

modificar ou adaptar as interfaces dos módulos envolvidos. Além de intermediar a interação entre os módulos, os conectores podem especificar mecanismos auxiliares de implementação [Shaw et al. 1996], tais como compartilhamento de dados, protocolos de comunicação, protocolos de interação, modelos de sincronização e concorrência.

Os módulos e conectores podem ser definidos e especificados de forma independente um do outro e, posteriormente, configurados em uma arquitetura, ou ainda reutilizados em diferentes contextos. Essa abordagem facilita a obtenção da propriedade de separação de interesses. Com essa propriedade, as funções básicas de uma aplicação (propriedades funcionais) são descritas separadamente dos aspectos operacionais ou propriedades não-funcionais, ou de qualquer outro aspecto não coberto na descrição funcional. Dessa maneira, o projetista de software pode se concentrar de forma separada nos aspectos funcionais da aplicação, podendo encapsulá-los nos módulos, e nos não-funcionais, podendo encapsulá-los nos conectores [Sztajnberg 2002]. Isso possibilita um maior reaproveitamento de software, facilidade de extensão e manutenção, contribuindo assim para a redução do custo de desenvolvimento de aplicações.

Para exemplificar, considere a arquitetura de uma aplicação cliente-servidor simples, onde o cliente solicita serviços providos por um servidor (ilustrada na Figura 3.1). Ela é composta por dois módulos (**Cliente** e **Servidor**), a porta **requer**, pela qual o cliente requisita o serviço desejado, a porta **prover**, por onde o servidor provê os serviços oferecidos, e um conector (**C-S**) que interliga os dois módulos da arquitetura.

O Código 3.1 descreve através da ADL CBabel essa mesma arquitetura. Na linha 02 são definidas as portas que representam os pontos de interação entre os módulos da aplicação. Nas linhas 03 a 08 são definidas as classes de módulos `Cliente` e `Servidor` e suas respectivas instâncias (`cliente` na linha 05 e `servidor` na linha 08). Uma instância da porta `requer` é declarada como porta de saída (`out`) em `Cliente` (linha 04), ou seja, representa a requisição de serviço ao servidor. Já a instância da porta `prover`, é definida como porta de entrada (`in`) em `Servidor` (linha 07), indicando o provimento de serviço. O conector da arquitetura (`C-S`) é declarado nas linhas 09 a 12. Nele, a porta `requer` é declarada como porta de entrada, pois ela oferece o ponto de conexão lógica com o `Cliente`, e a porta `prover` é definida do tipo saída, pois é através dela que o conector acessa o serviço oferecido pelo `Servidor`. Nas linhas 13 e 14 os módulos são instanciados (através do uso da primitiva arquitetural `instantiate`), sendo que as palavras `HostServidor` e

`HostCliente` representam, respectivamente, os nomes remotos dos *hosts* onde serão instanciados o servidor e o cliente. Para que o nome real de um *host* possa ser conhecido, pode-se utilizar um serviço de nomes ou o próprio endereço remoto do *host*. Por fim, na linha 15 o módulo `Cliente` é ligado ao `Servidor` através de uma instância do conector definido (primitiva arquitetural `link`).

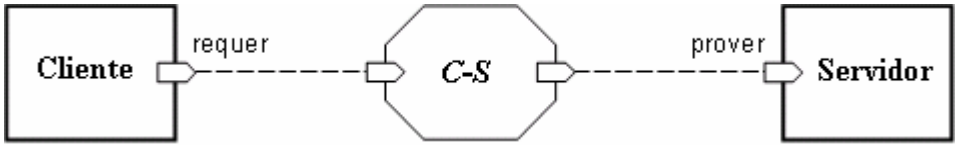

Figura 3.1 – Arquitetura de uma aplicação cliente-servidor simples.

```
01  module ClienteServidor {
02    port requer, prover;
03    module Cliente {
04       out port requer;
05    } cliente;
06    module Servidor {
07       in port prover;
08    } servidor;
09    connector C-S {
10       in port requer;
11       out port prover;
12    } cs;
13    instantiate servidor at HostServidor;
14    instantiate cliente at HostCliente;
15    link cliente to servidor by cs;
16 } cliente_servidor;
```

Código 3.1 – Descrição em CBabel da arquitetura de uma aplicação cliente-servidor.

É possível que uma aplicação seja implantada mediante a interpretação de sua descrição arquitetural elaborada por uma ADL, como a exemplificada no Código 3.1. Ambientes com suporte a gerenciamento de configuração arquitetural permitem que uma aplicação seja implantada e executada, com a instanciação de seus módulos funcionais e ligações apropriadas entre eles sendo feitas através de um interpretador de comandos de uma ADL específica [Lisbôa 2003]. Num nível maior de complexidade, tais ambientes podem ainda permitir a (re)configuração dinâmica de um sistema, através da alteração de sua topologia.

3.1.2 Descrição de Contratos de QoS

Através do conceito de contrato de qualidade de serviço (QoS), CBabel permite que os níveis de qualidade desejados para os serviços de uma aplicação sejam especificados, juntamente com os requisitos não-funcionais necessários à garantia da qualidade desejada [Curty 2002, Ansaloni 2003]. Um contrato é um relacionamento formal entre duas ou mais partes que utilizam ou fornecem recursos, onde são expressos direitos, obrigações e regras de negociação sobre os recursos utilizados. Consideramos recurso todo elemento que executa ou fornece suporte para a execução das atividades que compõem os serviços de uma aplicação. Nesse sentido, não só os elementos de sistema (e.g., processador, memória, rede), mas também módulos de suporte de uma aplicação (e.g., réplicas de servidores em uma aplicação cliente-servidor com mecanismo de replicação) podem ser considerados recursos.

Descrevendo, em tempo de projeto, o uso a ser feito dos recursos necessários por uma aplicação, assim como as variações aceitáveis na disponibilidade desses recursos, um contrato de QoS em CBabel regula os aspectos de qualidade da aplicação. Por exemplo, uma aplicação com requisitos de tempo real pode possuir um contrato de QoS definindo regras para a utilização dos recursos de processamento e restringindo o tempo de execução das tarefas da aplicação [Freitas et al. 2005]. Essa aplicação pode ter seus recursos de processamento expandidos (pelo uso de mais processadores, por exemplo) por um ambiente de configuração, quando for monitorada a existência de processadores disponíveis e a restrição de tempo não estiver sendo atendida pela atual configuração. Dessa maneira, o gerenciamento do contrato para essa aplicação observará os limites de tempo para execução de suas tarefas, permitindo a adição de mais recursos de processamento, caso as restrições de tempo sejam violadas.

A sintaxe de descrição de contrato de QoS CBabel especifica os serviços de uma aplicação, associando a eles restrições de qualidade exigidas. Desta forma, um serviço é definido como uma atividade que é governada por um conjunto de restrições para a utilização e provimento de recursos. Serviços são constituídos por uma ou mais descrições de configuração arquitetural, declaradas através das primitivas arquiteturais de CBabel (por exemplo, *instantiate* e *link*). A cada descrição de configuração são associados perfis que especificam os níveis de qualidade requeridos. Assim, a descrição de um serviço é feita pela seguinte sintaxe, descrita na notação BNF:

```
SERVICO       := service { SCORPO } NOMESERVICO ;
SCORPO        := CONF+
CONF          := COMANDOADL with NOMESPERFIS+ ;
COMANDOADL    := INSTANTIATE | REMOVE | LINK
INSTANTIATE   := instantiate NOMEINSTANCIA LOCAL
LOCAL         := at NOMEHOST | NULL
REMOVE        := remove NOMEINSTANCIA
LINK          := link NOMEINSTANCIA to NOMEINSTANCIA CONECTOR
CONECTOR      := by NOMEINSTANCIA | NULL
NOMEINSTANCIA := ID
NOMESPERFIS   := NOMESPERFIS , NOMEPERFIL | NOMEPERFIL
NOMEPERFIL    := ID
NOMESERVICO   := ID
ID            := STRING
```

As primitivas arquiteturais CBabel utilizadas na configuração de um serviço (`link` e `instantiate`) são refletidas em primitivas de alocação de recursos no nível de suporte. Cada primitiva CBabel contida em um serviço possui um ou mais perfis (`NOMESPERFIS`) associados que determinam os níveis desejados para os recursos a serem utilizados. Nas versões anteriores da linguagem de descrição de contrato [Curty 2002, Ansaloni 2003], os perfis eram associados às primitivas CBabel através da palavra reservada `with profile`. Porém, com o intuito de simplificar a sintaxe da linguagem a palavra `profile` foi removida. Os níveis desejados são declarados nos perfis através de predicados lógicos formados pela valoração de propriedades não-funcionais da categoria de um recurso (por exemplo, processamento), de acordo com os níveis desejados para elas. A sintaxe para a definição de perfis é apresentada a seguir na notação BNF:

```
PERFIL           := profile { PCORPO } ID ;
PCORPO           := RESTRICAO+
RESTRICAO        := NOMECATEGORIAQOS.NOMEPROPRIEDADE OPERADOR VALOR ;
NOMECATEGORIAQOS := ID
NOMEPROPRIEDADE  := ID
```

```
OPERADOR        := < | > | = | == | <= | >=
VALOR           := STRING | NUMBER
ID              := STRING
```

Os predicados lógicos de um perfil impõem valores limites (através do `OPERADOR`) para parâmetros ou propriedades dos recursos (`NOMEPROPRIEDADE`) envolvidos no provimento de um serviço, de acordo com o desejado (`VALOR`). Cada recurso é descrito pela primitiva *QoSCategory*, que identifica o recurso pelo seu nome e pelas declarações das propriedades pertencentes a ele. As descrições de recursos não pertencem a um contrato em particular. Na verdade, todo recurso disponível deve ser descrito separadamente da descrição de contratos e deve ser mantido em um repositório especial para que seja compartilhado por todos os contratos que necessitem de sua descrição. A sintaxe para a definição de recursos é apresentada a seguir na notação BNF:

```
QOSCATEGORY     := QoSCategory ID { QCORPO } ;
QCORPO          := DECLARACAO+
DECLARACAO      := NOMEPROPRIEDADE : TIPO UNIDADE ORIENTACAO ; |
                   NOMEPROPRIEDADE : TIPO ORIENTACAO ;
NOMEPROPRIEDADE := ID
TIPO            := ENUM | NUMERIC
UNIDADE         := ID
ORIENTACAO      := in | out
ENUM            := enum { ENUMERACOES+ }
ENUMERACOES     := ENUMERACOES , ID | ID
NUMERIC         := PREFERENCIA numeric
PREFERENCIA     := increasing | decreasing | NULL
ID              := STRING
```

Na descrição de uma categoria de QoS (`QoSCategory`) as propriedades de um recurso podem ser de um conjunto enumerado (`enum`) ou assumir um valor numérico (`numeric`). A palavra reservada `increasing` indica que quanto maior o valor associado à propriedade, maior é o nível de qualidade oferecido, e `decreasing` representa o contrário,

quanto menor o valor associado à propriedade maior é o nível de qualidade. Opcionalmente pode ser especificada a unidade de medida de uma propriedade do tipo numérica (`UNIDADE`). Para que fosse possível distinguir se uma propriedade da categoria será utilizada como condição de validade de um perfil ou apenas orientará alguma ação na configuração dos recursos, foram introduzidas por este trabalho as palavras reservadas `in` e `out`. Isso surgiu da necessidade de se identificar quais restrições em um perfil serão levadas em consideração para a sua validação e quais orientam as configurações dos recursos a serem utilizados. Com esse fim, `in` determina que uma propriedade será utilizada para especificar uma restrição que será avaliada como condição de validade de um perfil (entrada). Já a palavra `out`, determina que o valor da propriedade descrito em uma restrição apenas orientará uma configuração ou forma de utilização do recurso (saída ou resultado da aplicação do valor declarado para a propriedade).

Como exemplo, o recurso de transporte do canal de comunicação em uma rede poderia ser descrito pelas propriedades atraso (*delay*), banda passante (*bandwidth*) e protocolo de comunicação (UDP ou TCP, por exemplo), conforme mostrado a seguir:

```
QoSCategory Transport {
    delay: decreasing numeric ms in;
    bandwidth: increasing numeric Mbps in;
    protocol: enum (UDP, TCP) out;
};
```

Baseado na descrição dessa categoria, um perfil de utilização do recurso transporte (`Transport`) poderia ser descrito da seguinte maneira:

```
profile {
    Transport.delay < 50;
    Transport.bandwidth >= 1;
    Transport.protocol = UDP;
} transport_01;
```

Na declaração do perfil `transport_01` as duas primeiras restrições (`Transport.delay < 50 ms` e `Transport.bandwidth >= 1 Mbps`) representam as

condições de validação deste perfil, já que na definição da categoria `Transport` as propriedades `delay` e `bandwidth` foram declaradas como do tipo entrada (*in*). A última restrição apenas orienta que o protocolo de comunicação a ser utilizado será o UDP.

Um contrato deve ainda possuir uma única cláusula de negociação de serviços. Essa cláusula deve prever todas as possibilidades de transição entre os serviços a serem gerenciados. Ela é definida pela seguinte sintaxe:

```
NEGOCIACAO  := negotiation { REGRA+ }
REGRA       := not NOMESERVICO -> SERVICOS ; |
               NOMESERVICO -> SERVICOS
SERVICOS    := NOMESERVICO  |
               ( NOMES )
NOMES       := NOMES || NOMESERVICO | NOMESERVICO
NOMESERVICO := STRING
```

Uma linha de pesquisa em nosso grupo investiga a utilização de técnicas formais de validação para a ADL CBabel [Rademaker et al. 2004, Rademaker 2005]. Nesse sentido, foram sugeridas algumas modificações na sintaxe e na semântica da cláusula de negociação de serviços descrita em um contrato de QoS CBabel [Loques et al. 2005], proposta originalmente em [Curty 2002]. A cláusula de negociação define uma máquina de estados onde cada estado representa um serviço e as transições representam a disponibilidade do serviço. Ou seja, o símbolo `->` define uma transição do serviço à sua esquerda para o serviço à sua direita e, de acordo com a nova semântica proposta, deve ocorrer quando um dos serviços da direta puder ser provido. Um serviço pode ser provido caso todas as restrições de entradas (*in*) dos perfis a ele associados forem satisfeitas. É possível que mais de um serviço tenha suas restrições de validação satisfeitas ao mesmo tempo, neste caso é respeitada a prioridade dos serviços, que é considerada como a ordem em que eles são descritos, isto é, da esquerda para direita. A modificação realizada na sintaxe refere-se à inclusão da palavra reservada `not`, que é utilizada em uma regra de negociação para indicar o desejo da transição somente ocorrer caso o serviço à esquerda não puder ser mais provido. Isso contempla o caso do serviço à esquerda ser o de maior preferência, sendo desejado, portanto, que somente ocorra a troca desse serviço caso suas restrições de qualidade não possam mais ser satisfeitas.

Na proposta [Curty 2002] o estado `out-of-service` – que indica que nenhum serviço da cláusula pode ser estabelecido – é declarado explicitamente na regra de transição. Porém, foi observado que em todas as regras de transição este estado tinha que ser declarado, já que, se as restrições do serviço à esquerda forem violadas e nenhum dos serviços à direita puder ser estabelecido, o estado `out-of-service` será atingido. Levando isso em consideração, foi proposto neste trabalho que, se mais nenhum serviço puder ser provido, o estado `out-of-service` é automaticamente alcançado, sendo um estado final, implicando no encerramento do gerenciamento de um contrato.

Um contrato então, deve ser composto pela declaração dos serviços a serem contratados, a cláusula de negociação entre as possibilidades de serviços e os perfis especificando os níveis de qualidade desejados. Atendendo a isso, um contrato é especificado pela seguinte sintaxe:

```
CONTRACT := contract { CCORPO } ID ;
CCORPO   := SERVICO+ NEGOCIACAO
ID       := STRING
```

Para exemplificar a declaração de um contrato, considere uma aplicação cliente-servidor, cuja descrição arquitetural está contida no Código 3.1. Uma aplicação nessa arquitetura poderia ter, por exemplo, uma restrição de qualidade no canal de comunicação entre o cliente e o servidor que exija uma disponibilidade mínima na banda passante desse canal, de forma a garantir que a comunicação possa ocorrer dentro de limites razoáveis de tempo. O Código 3.2 descreve um contrato de QoS para essa aplicação. Esse contrato possui apenas um serviço declarado (`servico_qos_transporte`, linhas 02 a 04), que orienta, através da primitiva arquitetural `link` (linha 03), a ligação do módulo `cliente` ao módulo `servidor`, caso a restrição de qualidade definida no perfil associado (`perfil_transporte`, linhas 11 a 14) possa ser satisfeita. O perfil especifica, utilizando a propriedade `bandwidth` da categoria `Transport` descrita anteriormente nesta seção, que a banda passante do canal de comunicação deve ser maior ou igual a 1 Mbps (linha 12). Assim sendo, o serviço somente será estabelecido caso essa restrição possa ser satisfeita, já que a propriedade `bandwith` é do tipo de entrada (*in*). A outra restrição apenas orienta que o protocolo de comunicação a ser utilizado será o UDP (linha 13). A cláusula de negociação do contrato (linhas 06 a 08) descreve apenas o único serviço do contrato, significando que se ele

não puder ser provido o estado *out-of-service* será atingido, indicando a escassez do recurso requerido. Na detecção desse estado, a aplicação pode ser notificada que houve uma degradação nos níveis de recurso e que nenhum serviço pode ser estabelecido, podendo a aplicação ser finalizada, dependendo do comportamento desejado.

```
01  contract {
02    service {
03          link cliente to servidor with perfil_transporte;
04    } servico_qos_transporte;
05
06    negotiation {
07          servico_qos_transporte;
08    };
09  } cliente_servidor;
10
11  profile {
12          Transport.bandwidth >= 1;
13          Transport.protocol = UDP;
14  } perfil_transporte;
```

Código 3.2 – Contrato de QoS para uma aplicação cliente-servidor.

Desta maneira, um contrato pode descrever o uso que uma aplicação irá fazer dos recursos compartilhados e as variações aceitáveis sobre a disponibilidade desses recursos. Um contrato é imposto em tempo de execução através de um conjunto de componentes de suporte que mapeiam a semântica do contrato em elementos computacionais. Esses componentes serão descritos na próxima seção.

3.2 Infra-estrutura de Suporte

Para suportar o gerenciamento de aplicações com requisitos não-funcionais, segundo a descrição de um contrato de QoS CBabel associado, o *framework* CR-RIO provê uma infra-estrutura de suporte composta por um conjunto de componentes que implementam a semântica do contrato [Curty 2002, Ansaloni 2003]. Neste trabalho, essa infra-estrutura foi reavaliada, identificando a responsabilidade de cada um de seus componentes, e então, propondo um refinamento da sua arquitetura. O refinamento consistiu na resolução das responsabilidades dos componentes, o que resultou na otimização da infra-estrutura proposta

inicialmente [Curty 2002]. Assim, a arquitetura do *framework* refinada é composta pelos seguintes componentes: Gerenciador de Contratos (*Contract Manager*, CM), Contratador de QoS (*Contractor*), Agente de Recurso (*Resource Agent*) e o Configurador (*Configurator*). A Figura 3.2 ilustra esses componentes e a interação entre eles no processo de gerenciamento de contrato, onde cada um possui as seguintes responsabilidades:

- Contract Manager

O *Contract Manager* (CM) representa a autoridade principal no gerenciamento de contratos de QoS. Ele é responsável pela interpretação da descrição de um contrato (*Contract*), extraindo os serviços contratados, juntamente com suas respectivas restrições de qualidade (*profiles*), e a máquina de estados de negociação de serviços. Obtidas as informações necessárias do contrato, o CM inicia tentativas de estabelecimento de serviço, respeitando a preferência estabelecida na ordem dos serviços declarados na cláusula de negociação do contrato. Para verificar se um serviço pode ser estabelecido, o CM solicita aos *Contractors* a verificação das restrições exigidas pelo serviço. Se as restrições forem satisfeitas, o serviço é estabelecido e as configurações por ele necessárias (primitivas arquiteturais descritas no serviço especificado no contrato) são passadas para o configurador (*Configurator*).

Depois de um serviço ser estabelecido, se em um determinado momento o nível de algum recurso utilizado por esse serviço degradar, não satisfazendo mais as restrições de qualidade por ele requeridas, o CM é notificado que o serviço não pode continuar a ser provido. Neste caso, o CM consulta a cláusula de negociação de serviços do contrato a fim de verificar a existência de candidatos a próximo serviço. Caso exista, ele tenta estabelecer o serviço cujas restrições sejam compatíveis com os atuais níveis de recursos, obedecendo à ordem de preferência descrita na cláusula de negociação de serviços. Se nenhum serviço puder mais ser provido, o estado *out-of-service* é alcançado e o gerenciamento do contrato para a aplicação é encerrado. Quando isso ocorrer, o CM notifica a aplicação sobre a escassez de recursos para os serviços requeridos e, dependendo do comportamento desejado, o *Configurator* pode finalizar a execução da aplicação. O CM pode também iniciar uma nova negociação quando ele conclui que um serviço de maior preferência torna-se disponível.

- Contractor

O *Contractor* gerencia o processo de monitoração das propriedades de recursos especificadas nas restrições associadas aos serviços. Esse gerenciamento consiste do envio de

requisições solicitando os valores dessas propriedades aos agentes de recursos (*Resource Agent*) e da observância das restrições do serviço corrente em face aos valores monitorados. Caso ocorra de alguma restrição do serviço corrente ser violada, o *Contractor* notifica essa ocorrência ao CM. O CM também é notificado pelo *Contractor*, quando as propriedades de recursos monitoradas satisfazem as restrições, referentes a essas propriedades, de um serviço de maior prioridade que o serviço corrente.

Para cada contrato particular, uma especialização do *Contractor* deve ser implementada. Essa especialização se encarrega de informar quais recursos necessitam ser monitorados e de verificar se as restrições específicas aos serviços do contrato em questão são satisfeitas ao longo da execução de uma aplicação (maiores detalhes dessa especialização serão vistos no Capítulo 5).

- Resource Agent

O *Resource Agent* também é especializado. Ele encapsula o acesso aos mecanismos que executam ou fornecem suporte para a execução das atividades que compõem os serviços de uma aplicação, provendo interfaces para efetivar a alocação de recursos, iniciar serviços locais de sistema e monitorar os valores de propriedades requeridas. Assim, um *Resource Agent* é específico a um determinado recurso. Se uma aplicação define restrições de qualidade para os recursos de processamento e canal de comunicação, por exemplo, será necessário um *Resource Agent* para cada um desses recursos.

No processo de gerenciamento de contratos, os *Resource Agents* são responsáveis pela monitoração das propriedades de recurso sobre as quais foram definidas as restrições de qualidade. Os valores monitorados são passados para o *Contractor* quando mudanças são detectadas. Isso dispara no *Contractor* o processo de verificação de validade das restrições associadas ao serviço corrente. Além disso, os *Resource Agents* são responsáveis por reservas de recursos e por configurações em serviços locais de sistema, que podem ser requeridos no estabelecimento de um serviço ou para garantir o seu provimento, perante mudanças no estado dos recursos monitorados.

- Configurator

O *Configurator* representa o mecanismo responsável pela efetivação das configurações, na infra-estrutura de suporte à aplicação ou nela própria, requeridas pelos serviços descritos em um contrato. Como essas configurações são descritas na linguagem CBabel através de suas primitivas arquiteturais, torna-se necessário mapeá-las em ações do

nível de sistema que efetivarão as configurações requeridas. Por exemplo, a configuração de um serviço estabelecido em um contrato de gerenciamento de réplicas de servidor pode orientar o *Configurator* a adicionar uma nova réplica, com intuito de aumentar a capacidade de processamento de um determinado servidor. Assim, para que essa adição seja efetivada no sistema, o *Configurator* deve mapear a configuração arquitetural desse serviço em comandos de um mecanismo de gerenciamento de réplicas que efetue a adição de uma nova réplica.

Neste contexto, o ambiente de suporte a gerenciamento de configuração arquitetural proposto em [Lisbôa 2003] pode ser utilizado como *Configurator*. Como esse ambiente se baseia na ADL CBabel para descrever as configurações requeridas por uma aplicação (implantação e interligação de seus módulos e conectores), as configurações arquiteturais descritas nos serviços de um contrato, como a da linha 03 do Código 3.2 mostrado anteriormente, poderiam ser repassadas diretamente para esse ambiente, sem a necessidade de mapeamentos adicionais.

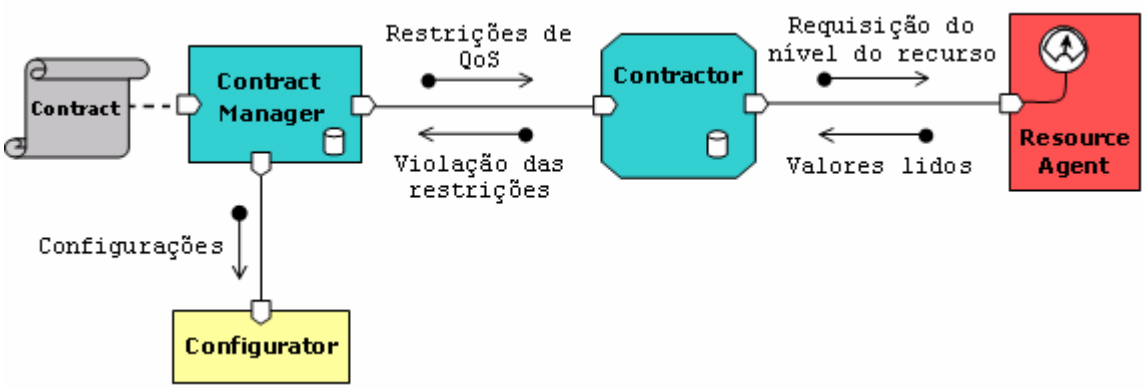

Figura 3.2 – Arquitetura de suporte do *framework* CR-RIO.

Como pode ser observado nesta seção, a infra-estrutura de suporte do *framework* CR-RIO contempla o modelo geral de suporte a gerenciamento de aplicações com requisitos não-funcionais apresentado na seção 2.2, permitindo o gerenciamento de aplicações com requisitos de QoS. Correlacionando com o modelo geral, o *Contract* refere-se ao elemento Especificação, *Contract Manager* e *Contractor* ao elemento Gerência, *Resource Agent* ao elemento Monitoramento e *Configurator* ao Configuração.

3.3 Conclusão do Capítulo

Foi apresentado neste capítulo o *framework* CR-RIO, que utiliza uma abordagem baseada no conceito de arquitetura de *software* para especificar e gerenciar aplicações com requisitos não-funcionais. O CR-RIO permite descrever contratos de QoS que especificam as restrições não-funcionais desejadas pelos serviços de uma aplicação, através da linguagem CBabel. Para suportar o gerenciamento desses contratos, o *framework* possui uma infra-estrutura de suporte composta por componentes que, através da interpretação de um contrato de QoS, gerenciam as restrições de qualidade requeridas por uma aplicação.

Refinamentos na linguagem utilizada para descrição de contratos de QoS e na infra-estrutura de suporte a gerenciamento de contatos foram propostos neste capítulo. A linguagem de descrição de contratos foi refinada com intuito de simplificar sua sintaxe, resultando em uma linguagem mais legível, e de resolver alguns aspectos formais necessários para sua interpretação consistente. Na infra-estrutura de suporte, as responsabilidades de cada componente foram delimitadas, resultando na otimização da configuração de sua arquitetura.

O próximo capítulo apresenta alguns casos de estudo que permitiram avaliar a proposta. Eles consistem de aplicações que possuem requisitos de qualidade para o oferecimento de seus serviços.

Capítulo 4

Exemplos de Aplicação

Este capítulo descreve alguns exemplos de aplicações com requisitos não-funcionais e como a proposta CR-RIO é utilizada para gerenciá-las, de acordo com as restrições de qualidade exigidas nos contratos de QoS definidos para essas aplicações. Esses exemplos permitem demonstrar através de casos de uso a utilidade do CR-RIO no gerenciamento de aplicações com esse tipo de requisito.

São descritos três exemplos de aplicações: uma aplicação de vídeo sob demanda, uma aplicação cliente-servidor Web e uma aplicação de videoconferência.

4.1 Aplicação Vídeo sob Demanda

O cenário desta aplicação de vídeo sob demanda (VoD) consiste de um servidor multimídia, que contém vídeos armazenados, e de clientes que recebem e exibem os vídeos transmitidos pelo servidor. Os clientes podem usar diferentes plataformas, desde dispositivos portáteis até estações de trabalho, criando um cenário heterogêneo que oferece diferentes níveis de disponibilidade de recursos, tais como CPU e *bandwidth*.

Nesta aplicação um vídeo é transmitido preferencialmente no formato MJPEG (*Motion* JPEG), que possui melhor qualidade, porém requer mais recursos do canal de comunicação para a sua transmissão e dos clientes para efetuar o seu processamento na exibição. Caso os recursos sejam insuficientes para esse formato, o vídeo pode ser transcodificado, por exemplo, para o formato H.263, que possui qualidade inferior ao MJPEG e exige menos recursos para sua transmissão e processamento. Um exemplo seria um cliente que conectado ao servidor

através de um canal de baixa capacidade (banda passante limitada), tenha que exibir o vídeo no formato inferior de qualidade.

4.1.1 Arquitetura da Aplicação

A Figura 4.1 ilustra a arquitetura desta aplicação para as duas opções de transmissão de vídeo: (a) os recursos são suficientes para a transmissão de vídeo no formato MJPEG, e (b) onde a limitação dos recursos exige a transmissão no formato H.263. Para este último caso, um conector responsável pela transcodificação do formato MJPEG para o H.263 é inserido entre o cliente e o servidor.

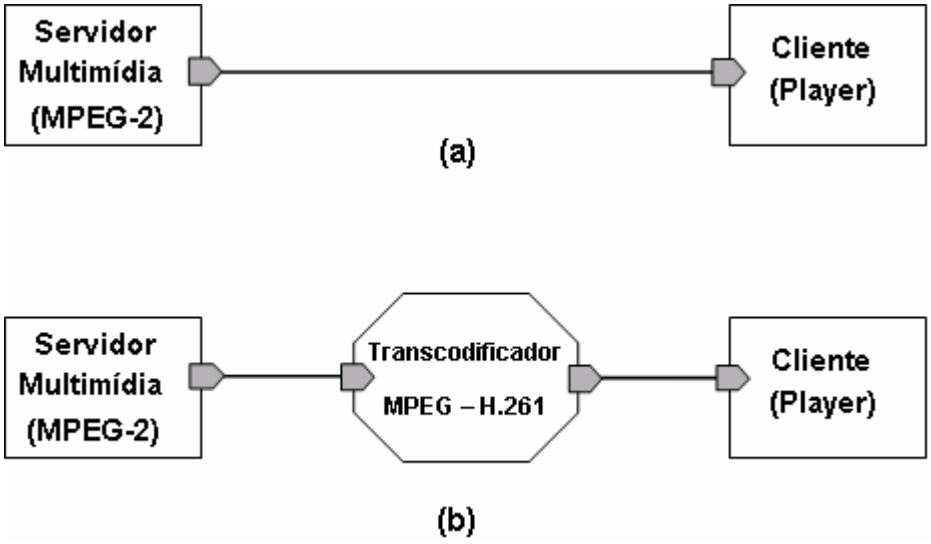

Figura 4.1 – Arquitetura da aplicação VoD.

O Código 4.1 descreve a arquitetura dessa aplicação através da ADL CBabel. Nele, são descritos os módulos `Cliente` e `Servidor` (linhas 03 a 08) e o conector responsável pela transcodificação do formato de vídeo (linhas 09 a 12). Na linha 13 o módulo `servidor` e o conector `H263` são instanciados em um determinado *host* (`HostServidor`). A partir dessa descrição, um ambiente de suporte a configuração poderia implantar o servidor no *host* desejado. Depois disso, o servidor multimídia estaria pronto para receber requisições por vídeos dos clientes da aplicação.

```
01  module VoD {
02    port requer, prover;
03    module Cliente {
04       out port requer;
```

```
05    } player;
06    module Servidor {
07       in port prover;
08    } servidor;
09    connector Transcodificador {
10       in port requer;
11       out port prover;
12    } H263;
13    instantiate servidor, H263 at HostServidor;
14 } vod;
```

Código 4.1 – Descrição em CBabel da arquitetura da aplicação VoD.

4.1.2 Categorias e Contrato de QoS

As restrições de qualidade para a aplicação VoD em questão são definidas pela valoração desejada das propriedades referentes aos recursos de processamento do cliente e transporte do canal de comunicação entre o cliente e o servidor. Portanto, para que a valoração dessas propriedades possa ser feita em um contrato de QoS para essa aplicação, é necessário que elas sejam descritas na forma de categorias de QoS (*QoSCategory*). As categorias de processamento e transporte e suas propriedades para o estabelecimento do contrato da aplicação VoD são apresentadas no Código 4.2.

```
01  QoSCategory Processing {
02      clockFrequence: increasing numeric MHz in;
03      utilization: decreasing numeric % in;
04  };
05
06  QoSCategory Transport {
07      delay: decreasing numeric ms in;
08      bandwidth: increasing numeric Mbps in;
09  };
```

Código 4.2 – Categorias de QoS para os recursos de transporte e processamento.

A categoria `Processing` (linhas 01 a 04) representa o recurso de processamento, onde a propriedade `clockFrequence` representa a freqüência de operação de processador e a propriedade `utilization` mede a porcentagem de utilização de CPU. Outras propriedades

poderiam ser descritas para esta categoria, tais como o número de instruções por ciclo de operação ou a prioridade para um determinado processo. Neste exemplo, é considerado que o cliente deve possuir uma CPU com freqüência de operação de no mínimo 700 MHz e com no máximo 50 % do tempo de CPU utilizado para a exibição de vídeos no formato MJPEG. A exibição de vídeos H.263 exigirá da CPU uma freqüência de operação de no mínimo 266 MHz e com no máximo 70 % do tempo de CPU utilizado.

A categoria `Transport` (linhas 06 a 09) representa o canal de comunicação entre o servidor e o cliente, onde a propriedade `bandwidth` representa a banda passante disponível para a conexão entre o cliente e o servidor, e a propriedade `delay` representa o atraso para a transmissão de dados neste canal de comunicação. O formato MJPEG requer uma `bandwidth` superior a 1.0 Mbps e um `delay` menor que 50 ms para uma exibição aceitável do vídeo, enquanto vídeos no formato H.263 requerem uma `bandwidth` de no mínimo 56 Kbps e podem experimentar um `delay` de até 200 ms. Outras propriedades do recurso de transporte poderiam ser levadas em consideração neste caso, tais como a variação do `delay` (*jitter*) ou a taxa de perdas de pacotes na rede (*data loss*), mas para efeito de simplificação do exemplo não serão consideradas.

Para especificar os serviços da aplicação VoD e seus respectivos níveis de qualidade requeridos, o próximo passo é a descrição do contrato de QoS em CBabel. O Código 4.3 descreve esse contrato, orientando o *framework* CR-RIO no gerenciamento dessa aplicação, de acordo com a qualidade requisitada para seus serviços. O contrato `Vod` (linhas 01 a 16) descreve dois serviços. O primeiro, `sMJPEG_video` (linhas 02 a 05), instancia um cliente (linha 03) e o conecta ao servidor de vídeo (linha 04), caso o recurso de processamento do cliente satisfaça as restrições especificadas no perfil `cpu_mjpeg` (linhas 18 a 21) e o canal de comunicação satisfaça as restrições declaradas no perfil `transport_mjpeg` (linhas 23 a 26). Se essas restrições forem satisfeitas o serviço será estabelecido e o cliente receberá o vídeo no formato preferencial (MJPEG). O serviço `sH263_video` (linhas 07 a 10) requer menos recursos, pois se estabelecido transmitirá o vídeo no formato inferior H.263. Para que o vídeo seja transmitido nesse formato, o cliente é conectado ao servidor pelo conector `H263` (linha 09), cuja função é transcodificar o vídeo do formato de maior para o de menor qualidade. As restrições referentes aos recursos de processamento e transporte para esse segundo serviço são descritas nos perfis `cpu_h263` e `transport_h263` (linhas 28 a 36).

Na cláusula de negociação de serviços do contrato `Vod` (linhas 12 a 15), o serviço `sMJPEG_video` é descrito como o de maior prioridade (linha 13), sendo que o outro serviço (`sH263_video`) só poderá ser estabelecido caso as restrições do primeiro não possam ser atendidas (*not*). Já na regra de negociação para o serviço `sH263_video` (linha 14), é definido que se ele estiver estabelecido e em um determinado momento o serviço `sMJPEG_video` puder ser provido, ele será, pois é o de maior preferência.

```
01  contract {
02    service {
03          instantiate player at HostCliente with cpu_mjpeg;
04          link player to servidor with transport_mjpeg;
05    } sMJPEG_video;
06
07    service {
08          instantiate player at HostCliente with cpu_h263;
09          link player to servidor by H263 with transport_h263;
10    } sH263_video;
11
12    negotiation {
13          not sMJPEG_video -> sH263_video;
14          sH263_video -> sMJPEG_video;
15    };
16  } Vod;
17
18  profile {
19    Processing.clockFrequence >= 700;
20    Processing.utilization <= 50;
21  } cpu_mjpeg;
22
23  profile {
24    Transport.delay <= 50;
25    Transport.bandwidth >= 1;
26  } transport_mjpeg;
27
28  profile {
29    Processing.clockFrequence >= 266;
30    Processing.utilization <= 70;
31  } cpu_h263;
32
33  profile {
```

```
34    Transport.delay <= 200;
35    Transport.bandwidth >= 0,056;  // 56 Kbps
36  } transport_h263;
```

Código 4.3 – Contrato de QoS para a aplicação VoD.

4.1.3 Gerenciamento do Contrato

Por fim, para que o contrato possa ser gerenciado pelo CR-RIO, é preciso implementar os agentes de recurso (*Resource Agents*) específicos aos recursos de processamento e transporte requeridos pela aplicação. É necessário também implementar o contratador de QoS (*Contractor*) que gerenciará o monitoramento feito pelos agentes de recurso, verificando se as propriedades monitoradas validam as restrições descritas no contrato, conforme discutido na seção 3.2. Além disso, é preciso um mecanismo que desempenhe o papel do *Configurator*, garantindo a efetivação das configurações para os serviços da aplicação. A Figura 4.2 ilustra a configuração dos componentes da infra-estrutura de suporte do CR-RIO para o gerenciamento do contrato para a aplicação em questão.

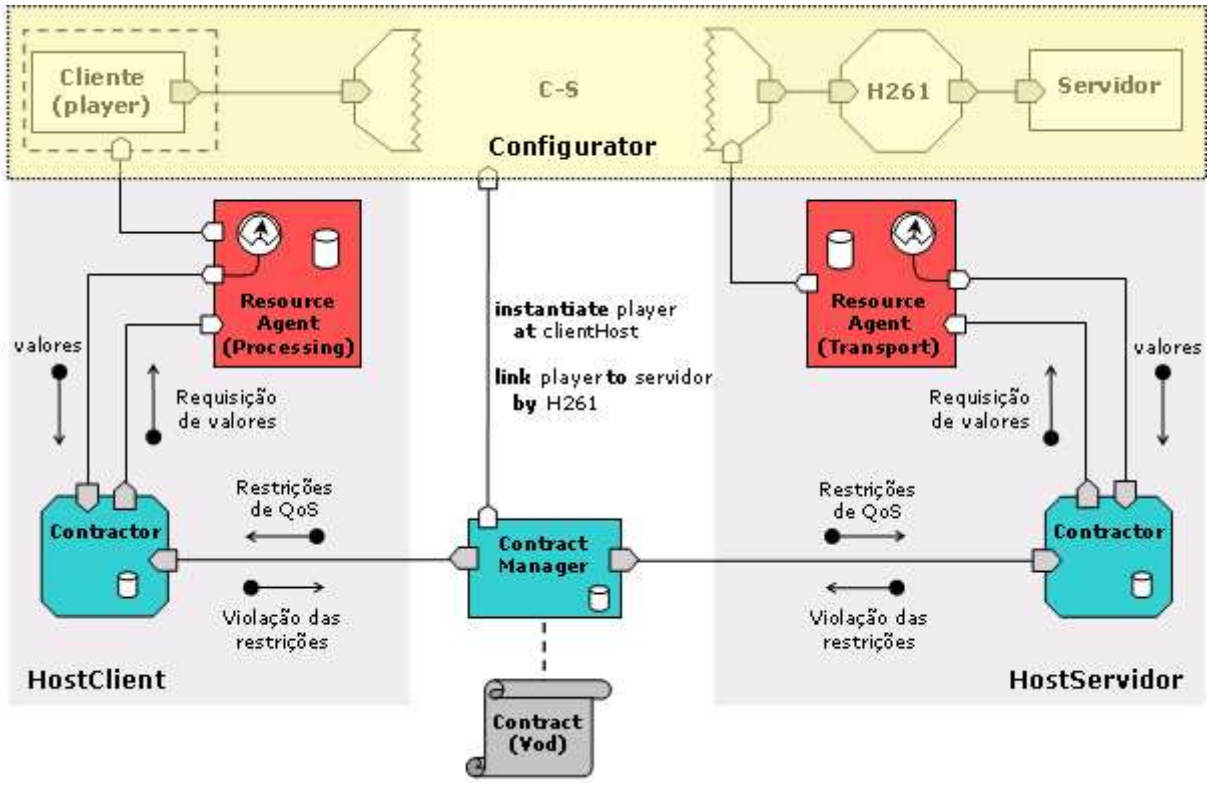

Figura 4.2 – Configuração dos componentes do CR-RIO para o gerenciamento do contrato da aplicação VoD.

No gerenciamento desse contrato (`Vod`), o primeiro passo é a interpretação de sua descrição pelo *Contract Manager* (CM), que poderia estar localizado em qualquer *host* da rede. Obtidas as informações do contrato, o CM identifica na cláusula de negociação de serviços que `sMJPEG_video` é o serviço de maior preferência. Para verificar se as restrições para esse serviço (perfis `cpu_mjpeg` e `transport_mjpeg` no Código 4.3) podem ser atendidas pelos níveis dos recursos de processamento do cliente e transporte do canal de comunicação, o CM envia essas restrições aos *Contractors* responsáveis pela verificação de cada um desses recursos. Para essa aplicação, haveria um *Contractor* na máquina do cliente, responsável pela verificação das propriedades do recurso processamento, e outro na máquina do servidor, para verificação das propriedades do recurso transporte. Esses *Contractors* recebem, então, essas restrições e solicitam aos *Resource Agents* (*Processing* e *Transport*, na Figura 4.2) os valores atuais das propriedades dos recursos por eles monitorados. Recebendo esses valores os *Contractors* verificam se eles satisfazem as restrições para o serviço `sMJPEG_video`. Caso as restrições sejam satisfeitas, o CM é notificado que esse serviço pode ser estabelecido. Para esse estabelecimento, o CM envia as configurações arquiteturais desse serviço (linhas 03 e 04 do Código 4.3) ao *Configurator*, que se encarrega de estabelecer a configuração exigida para o provimento do serviço `sMJPEG_video`.

Depois que `sMJPEG_video` estiver estabelecido, as propriedades dos recursos continuam sendo monitoradas pelo *framework* e caso alguma alteração em seus valores viole as restrições desse serviço, o *Contractor* notifica isso ao CM, que tenta o estabelecimento do próximo serviço (`sH263_video`). Por exemplo, se o serviço `sMJPEG_video` está sendo provido e, considerando que não é usado nenhum mecanismo de reserva, em um determinado momento a utilização da CPU do cliente atinja o valor de 60 %, violando assim a restrição de qualidade para esse serviço (linha 20 do Código 4.3), ele será finalizado. Nesse caso, o serviço `sH263_video` será estabelecido, já que esse valor não viola a sua restrição para essa mesma propriedade (linha 30 do Código 4.3). Caso essa propriedade volte a um valor aceitável pelo serviço preferencial, ele será restabelecido, desde que as outras restrições associadas a ele também sejam satisfeitas. Ou seja, se os níveis dos recursos voltarem a satisfazer as restrições associadas ao serviçco `sMJPEG_video`, ele será restabelecido.

4.2 Aplicação Cliente-Servidor Web

Este exemplo consiste de uma aplicação cliente-servidor Web, onde vários clientes enviam requisições de conteúdo a grupo de servidores (*cluster*), que processam essas requisições e enviam o conteúdo requisitado diretamente para o cliente. É desejado que o cliente tenha suas requisições atendidas dentro de um limite de tempo. Neste cenário, um fator importante que influencia nesse requisito de qualidade (tempo de resposta observado pelo cliente) é que o número de clientes pode variar, aumentando ou diminuindo a carga de processamento nos servidores. Além disso, a capacidade de processamento dos servidores de um grupo varia dinamicamente, por exemplo, uma outra aplicação que utiliza o mesmo grupo, pode sobrecarregá-lo. Outro fator importante é que a disponibilidade do canal de comunicação entre um cliente e um grupo de servidores também varia de forma dinâmica.

O ambiente desta aplicação ajuda a mostrar a utilidade do CR-RIO no gerenciamento de aplicações que disputam recursos concorrentes e cuja disponibilidade varia dinamicamente. Este cenário requer capacidade de adaptação dinâmica da infra-estrutura de suporte à aplicação, de forma a se adequar a mudanças no estado dos recursos do sistema.

4.2.1 Arquitetura da Aplicação

A Figura 4.3 ilustra a arquitetura dessa aplicação, onde servidores são associados a grupos (Grupo 1 e Grupo 2) distribuídos geograficamente que recebem as requisições oriundas dos clientes e as repassam aos seus servidores associados. Ao receber uma requisição, um servidor a processa e retorna o conteúdo solicitado diretamente para o cliente solicitante. Um grupo desempenha o papel de um *proxy* que, ciente do estado dos servidores a ele associados, pode decidir, baseado nos parâmetros de desempenho de cada um, a qual uma requisição será enviada [Cardellini et al. 2002]. O cliente estabelecerá conexão com o grupo cujo canal de comunicação tenha melhor disponibilidade. Isso pode ser resolvido de forma transparente para o cliente através de algum mecanismo que conheça os grupos existentes e que possa consultar a qualidade do enlace com cada um deles.

Neste contexto, um mecanismo de gerenciamento de réplicas de servidor pode ser utilizado para adicionar novos servidores a um grupo, quando for necessário aumentar a sua capacidade de processamento, e para removê-los, quando os servidores do grupo estiverem ociosos. Além disso, caso o canal de comunicação entre o cliente e um grupo de servidores

não tenha recursos suficientes para garantir a qualidade exigida, mecanismos que permitam a consulta dos parâmetros de qualidade dos enlaces com os outros grupos podem ser utilizados, possibilitando que o cliente seja conectado ao grupo cujo canal de comunicação atenda às suas restrições de qualidade.

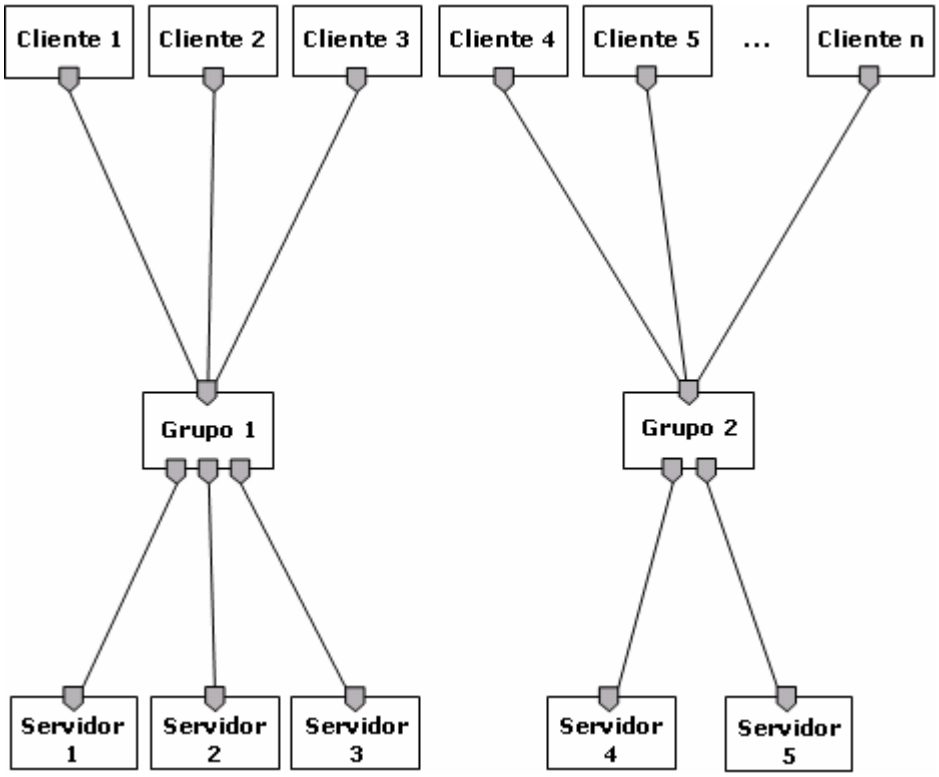

Figura 4.3 – Arquitetura da aplicação cliente-servidor Web.

4.2.2 Categorias de QoS

De acordo com os critérios de qualidade de serviço estabelecidos através de um contrato de QoS, esta aplicação pode ser gerenciada através do controle do mecanismo de replicação de servidores e do controle sobre a utilização de um canal de comunicação que satisfaça a qualidade requerida. Para definir esses critérios de QoS requeridos, é preciso especificar as propriedades de recurso exigidas pela aplicação através de categorias de QoS expressas na linguagem CBabel. O Código 4.4 descreve três novas categorias: uma especifica as propriedades de um grupo de servidores (`ServerGroup`), outra as propriedades referentes ao mecanismo de replicação (`Replication`) e a terceira as propriedades do cliente da aplicação (`Client`).

```
01  QoSCategory ServerGroup {
02      load: decreasing numeric % in;
03      loadDistributionPolicy: enum (roundRobin, random, optim) out;
04      serverGroupSet: enum (A, B, C, D, E) out;
05  };
06
07  QoSCategory Replication {
08      numberOfReplicas: increasing numeric in;
09      maxReplicas: numeric in;
10      replicaMaint: enum (add, release, maintain) out;
11      allocationPolicy: enum (bestMem, bestLoad, optim) out;
12  };
13
14  QoSCategory Client {
15      responseTime: decreasing numeric ms in;
16      linkPolicy: enum (maint, bestLink, bestGroup, optim) out;
17  };
```

Código 4.4 – Categorias de QoS dos recursos Grupo de Servidores, Replicação e Cliente.

A categoria `ServerGroup` (linhas 01 a 05) define as propriedades de um grupo de servidores. A propriedade `load` representa a carga de processamento de um determinado grupo e é a única propriedade de entrada (`in`) definida nesta categoria, ou seja, ela será utilizada para definir uma restrição que deverá ser válida para satisfazer o perfil ao qual ela pertencer. `loadDistributionPolicy` é a propriedade que orienta a política de distribuição de carga utilizada pelo grupo de servidores. Essas políticas são enumeradas na definição da propriedade (tipo `enum`), onde a primeira indica que a distribuição ocorrerá via *Round Robin*, a segunda de forma randômica, e a opção `optim` representa uma política "ótima", onde poderia ser definido, por exemplo, que uma requisição de processamento seria repassada ao servidor que possuir maior disponibilidade nos recursos de processamento. A propriedade `serverGroupSet` enumera os grupos de servidores existentes, que poderá ser utilizada no contrato de um cliente para determinar um servidor específico ao qual ele queira se conectar.

`Replication` (linhas 7 a 12) é a categoria referente ao mecanismo de replicação de servidores. Suas propriedades são: `numberOfReplicas`, que indica o número de réplicas (servidores) que um determinado grupo possui; `maxReplicas`, o número máximo de réplicas que um grupo pode possuir; `replicaMaint`, enumera a ação que será efetuada na manutenção da réplica, podendo ela ser removida (`release`), mantida (`maintain`) ou

adicionada uma nova (`add`); `allocationPollicy`, define as políticas que orientam a alocação de réplicas por um grupo. A opção `bestMem` define que será alocada a réplica que possuir mais recursos de memória disponíveis. Com a política `bestLoad` será alocado o servidor que contiver maior disponibilidade nos recursos de processamento. Já na opção `optim`, poderia ser definida uma política ótima que indique a réplica mais desejada.

A categoria `Client` (linhas 14 a 17) representa o cliente da aplicação e suas propriedades são: `responseTime`, o tempo de resposta observado pelo cliente e `linkPolicy`, as políticas que orientam a busca por um novo enlace. Nas opções de políticas definidas, `maint` indica que o cliente deve permanecer ligado àquele enlace; `bestLink` orienta a busca por um enlace com maior banda passante; `bestGroup` para buscar um enlace com o grupo de servidores que possuir maior carga de processamento disponível e `optim` uma política ótima para a definição do melhor enlace. Um exemplo de uma política ótima seria uma que orientasse a busca por um novo enlace levando em consideração as políticas `bestLink` e `bestGroup` em conjunto. Além dessas, a categoria `Transport` (Código 4.2) também será utilizada neste exemplo.

4.2.3 Contratos de QoS

Definidas as propriedades dos recursos a serem utilizados para o exemplo em questão, através das categorias acima, o próximo passo é a definição dos serviços da aplicação associados com seus respectivos níveis de qualidade desejados, através do contrato de QoS CBabel. Para o exemplo em questão, é descrito um contrato para o cliente e outro para o servidor, já que eles tratam de questões diferentes – o primeiro trata da conexão entre um cliente e o seu servidor e o outro o gerenciamento de réplicas por um grupo de servidores. Isso facilita a reutilização dos contratos e os torna mais claros e de mais fácil entendimento.

- Contrato para o cliente

O contrato para o cliente é descrito no Código 4.5. Ele descreve dois serviços associados com restrições de qualidades definidas para o tempo de resposta observado pelo cliente, que determinam a condição para o estabelecimento de cada um deles. O serviço `Smaintain` (linhas 02 a 04) conecta um cliente a um grupo de servidores, caso a restrição sobre o tempo de resposta definida no perfil `maintainClient` (linhas 16 a 19) seja satisfeita. Para especificar essa condição, a primeira restrição desse perfil (linha 17) é declarada através de uma propriedade de entrada (`responseTime`, na linha 15 do Código

4.4), determinando que o serviço `Smaintain` somente possa ser iniciado caso o tempo de resposta observado pelo cliente for menor ou igual a um tempo máximo estabelecido (`MáxResponseTime`). A outra restrição somente orienta que a conexão entre o cliente e o atual grupo de servidores deverá ser mantida, pois a restrição de tempo desejada pelo cliente está sendo satisfeita.

Caso o perfil `maintainClient` seja violado com o serviço `Smaintain` em funcionamento, uma tentativa de estabelecer o próximo serviço será efetuada, de acordo com a ordem estabelecida na cláusula de negociação de serviços do contrato (linhas 10 a 13). Neste caso, o próximo serviço é `SmoveClient` (linhas 6 a 8), que desconecta o cliente do atual grupo de servidores movendo-o para um novo, caso a banda passante disponível no canal de comunicação com o atual grupo seja menor que um mínimo estabelecido. Essa condição é descrita no perfil `moveClient` (linhas 21 a 24) pela propriedade de entrada `bandwidth` (descrita na categoria `Transport` do Código 4.2). Ou seja, ele verifica se o atraso no tempo de resposta ocorreu devido à escassez de banda passante na conexão entre o cliente e o servidor. Se isso for constatado, este serviço tentará mover o cliente para um novo grupo de servidores, cujo canal de comunicação tenha banda passante suficiente para o estabelecimento do serviço com o requisito de tempo de resposta desejado. A outra restrição deste mesmo perfil (linha 23) orienta a busca por um enlace com melhor banda passante possível.

Se nenhum dos serviços puder ser estabelecido, o estado *out-of-service* será alcançado, indicando a escassez dos recursos necessários ao atendimento dos serviços com a qualidade desejada. Contudo, um serviço cujo perfil reflita um estado de escassez de recursos poderia ser declarado, evitando que o gerenciamento do contrato fosse encerrado, perante a impossibilidade de estabelecimento dos serviços `Smaintain` e `SmoveClient`. Com esse novo serviço, o gerenciamento ficaria “estacionado” até a disponibilidade dos recursos voltarem a satisfazer as restrições descritas nos perfis associados aos serviços de maior preferência do contrato. Um exemplo dessa possibilidade é descrito através do serviço `SmaintainMinimum` do Código 4.6.

```
01  contract {
02    service {
03      link client to serverGroup with maintainClient;
04    } Smaintain;
05
```

```
06    service {
07      link client to serverGroup with moveClient;
08    } SmoveClient;
09
10    negotiation {
11      not Smaintain -> SmoveClient;
12      SmoveClient -> Smaintain;
13    }
14  } ClientWeb;
15
16  profile {
17    Client.responseTime <= MáxResponseTime;
18    Client.linkPolicy = maint;
19  } maintainClient;
20
21  profile {
22    Transport.bandwidth < MinBandwidth;
23    Client.linkPolicy = bestLink;
24  } moveClient;
```

Código 4.5 – Contrato de QoS para o cliente da aplicação cliente-servidor Web.

- Contrato para o grupo de servidores

O outro contrato refere-se aos serviços de gerenciamento de grupo de servidores, permitindo o controle do mecanismo de replicação de servidores de acordo com a demanda por recursos de processamento. Seu código está descrito no Código 4.6, sendo composto pelos seguintes serviços:

(a) `Smaintain` (linhas 02 a 04), que instancia um grupo de servidores responsável pelo processamento das requisições enviadas pelos clientes. O perfil `maintainServer` (linhas 26 a 30) associado a esse serviço define como condições de validação, através das restrições descritas nas linhas 27 e 28, que a carga de processamento do grupo de servidores deve ser menor ou igual a um valor máximo definido (`MaxServerLoad`) e maior ou igual a um valor mínimo (`MinServerLoad`). A última restrição desse perfil (linha 29) apenas orienta o mecanismo de gerenciamento de réplicas que, para este serviço, nenhuma réplica deve ser adicionada ou removida do grupo;

(b) `SaddServer` (linhas 06 a 08), que adiciona uma nova réplica ao grupo de servidores para aumentar a sua capacidade de processamento, se as restrições de entrada do

perfil `addServer` (linhas 32 a 37) associado a esse serviço puderem ser atendidas. As restrições de entrada para esse perfil (linhas 33 e 34), ou seja, as que o validarão, são descritas através das propriedades `load` e `numberOfReplicas`, que são especificadas como do tipo de entrada (`in`) no Código 4.4. Desta maneira, de acordo com essas restrições, um novo servidor só será associado a um grupo de servidores, se a carga de processamento desse grupo for maior que a máxima estabelecida (linha 33) e se o número máximo de réplicas permitido ainda não foi atingido (linha 34). As duas últimas restrições orientam, respectivamente, o mecanismo de replicação a adicionar uma nova réplica (linha 35) e na sua escolha levar em consideração a que tiver maior capacidade de carga de processamento (linha 36). Esse serviço se manterá ativo até que as condições de entradas sejam violadas, ou seja, quando a carga de processamento do grupo voltar ao nível desejado ou quando o número máximo de servidores for atingido, não podendo assim, ser adicionado mais réplicas ao grupo. Quando isso ocorrer, serão iniciadas tentativas de estabelecimento de um outro serviço, de acordo com a regra de negociação descrita para o serviço `SaddServer` (linha 20);

(c) `SremoveServer` (linhas 10 a 12), que é o serviço responsável pela remoção de réplicas de servidor de um grupo, quando a carga deste se tornar menor que uma mínima estabelecida. Dessa forma, servidores ociosos poderão ser disponibilizados para outras aplicações, prezando por um melhor aproveito dos recursos compartilhados. O perfil associado a esse serviço é `removeServer` (linhas 39 a 43), que define que a carga de processamento do grupo deve ser menor que uma mínima estabelecida (linha 40) e que o número de réplicas deve ser maior ou igual a um mínimo (linha 41) como condições de entrada. A última restrição (linha 42) apenas informa que a política de manutenção é a remoção de réplica, orientando assim, o mecanismo de gerenciamento de réplicas a remover um servidor do grupo;

(d) `SmaintainMinimum` (linhas 14 a 16), que será estabelecido quando a carga de processamento de um grupo estiver abaixo da mínima estipulada e o número de réplicas desse grupo tenha chegado ao mínimo permitido. Esse estado é declarado nas duas primeiras restrições do perfil `minimumNumberOfServers` (linhas 45 a 49) associado a este serviço. A outra restrição deste perfil (linha 48) orienta o mecanismo de gerenciamento de réplicas a manter a configuração atual. Ou seja, este serviço é responsável por manter a configuração da aplicação perante o estado de ociosidade de um grupo de servidores, evitando que *out-of-service* seja alcançado, o que implicaria no encerramento do gerenciamento do contrato.

A cláusula de negociação de serviços descrita para este contrato (linhas 18 a 23) define que a ordem de prioridade dos serviços, de acordo com a disposição em que eles são descritos, é: `Smaintain`, `SaddServer`, `SremoveServer` e `SmaintainMinimum`. Com isso, a tentativa de estabelecimento de serviço respeitará essa ordem. Em relação à transição desses serviços, como os perfis do contrato deste exemplo são mutuamente exclusivos, não é necessário o uso da palavra reservada `not` na descrição das regras de transição de serviço. Isso, uma vez que se um dos serviços declarados no lado esquerdo das regras estiver ativo e os perfis associados a um outro do lado direito se tornarem válidos, ocorrerá a transição para este último serviço. Por exemplo, se `Smaintain` estiver estabelecido e as restrições de entrada do perfil `addServer` tornarem-se válidas, o serviço ativo será finalizado e `SaddServer` será estabelecido. É importante notar que se o número máximo de réplicas for atingido (linha 34), o serviço `SaddServer` não será estabelecido e, nesse caso, nenhum outro poderá ser. Porém, assim como o serviço `SmaintainMinimum`, um novo serviço poderia ser declarado com o objetivo de capturar esse estado, ou seja, ele se tornaria ativo quando a carga de processamento de um grupo estiver acima da máxima tolerada (linha 33) e o número de réplicas for o máximo permitido, evitando assim que estado *out-of-service* seja alcançado. Dessa forma, esse novo serviço ficaria ativo até o momento em que a carga de processamento do grupo voltar ao valor tolerado, o que ativaria o serviço `Smaintain`.

```
01  contract {
02    service {
03      instantiate serverGroup with maintainServer;
04    } Smaintain;
05
06    service {
07      instantiate server with addServer;
08    } SaddServer;
09
10    service {
11      remove server with removeServer;
12    } SremoveServer;
13
14    service {
15      instantiate server with minimumNumberOfServers;
16    } SmaintainMinimum;
17
18    negotiation {
```

```
19      Smaintain -> ( SaddServer || SremoveServer || SmaintainMinimum );
20      SaddServer -> ( Smaintain || SremoveServer );
21      SremoveServer -> ( Smaintain || SaddServer || SmaintainMinimum );
22      SmaintainMinimum -> ( Smaintain || SaddServer );
23    }
24  } ReplicationManager;
25
26  profile {
27    ServerGroup.load <= MaxServerLoad;
28    ServerGroup.load >= MinServerLoad;
29    Replication.replicaMaint = maintain;
30  } maintainServer;
31
32  profile {
33    ServerGroup.load > MaxServerLoad;
34    Replication.numberOfReplicas < maxReplicas;
35    Replication.replicaMaint = add;
36    Replication.allocationPolicy = bestLoad;
37  } addServer;
38
39  profile {
40    ServerGroup.load < MinServerLoad;
41    Replication.numberOfReplicas >= minReplicas;
42    Replication.replicaMaint = release;
43  } removeServer;
44
45  profile {
46    ServerGroup.load < MinServerLoad;
47    Replication.numberOfReplicas == minReplicas;
48    Replication.replicaMaint = maintain;
49  } minimumNumberOfServers;
```

Código 4.6 – Contrato de QoS para o gerenciamento de grupos de servidores da aplicação cliente-servidor Web.

4.2.4 Gerenciamento dos Contratos

Para este exemplo, no que diz respeito à configuração dos componentes da infra-estrutura de suporte do CR-RIO (ver seção 3.2), seria necessária a criação dos *Resource Agents* para o monitoramento das propriedades de entrada das categorias `ServerGroup`,

`Replication`, `Client` e `Transport`. Depois disso, seria preciso criar um *Contractor* específico às restrições de qualidade descritas no contrato para o cliente (Código 4.5) e outro para o contrato de gerenciamento de réplicas de servidor (Código 4.6). Como é o *Contractor* que gerencia o monitoramento feito pelos componentes *Resource Agents*, é preciso implantá-lo em cada *host* que possuir alguma propriedade de recurso que deva ser monitorada. Por fim, seria necessário implementar o *Configurator*, também específico, neste caso, a cada um dos contratos. No caso do contrato do cliente, essa implementação seria responsável por um mecanismo de descoberta de recursos que permitiria buscar um enlace com melhor qualidade e pelo estabelecimento da conexão entre o cliente e um grupo de servidores. Já para o contrato de grupo de servidores, o *Configurator* seria responsável por implementar ou acessar mecanismos que permitam iniciar um grupo de servidores e o gerenciamento de suas réplicas.

O gerenciamento dos contratos envolvidos na aplicação seria realizado de forma independente. Assim sendo, no gerenciamento do contrato para um cliente da aplicação, considerando que cada cliente estaria em um *host* distinto, seria necessário implantar um *Contractor*, responsável por coordenar a monitoração da propriedade tempo de resposta observado pelo cliente, e um *Contract Manager*, responsável pelo gerenciamento do contrato, em cada *host* cliente. Desta maneira, cada cliente terá um contrato, conforme o definido no Código 4.5, e os componentes da infra-estrutura de suporte necessários para gerenciá-lo de acordo com as restrições de qualidade por ele desejadas.

No gerenciamento do contrato para os grupos de servidores da aplicação, considerando que os grupos de servidores estariam localizados em *hosts* distintos, uma possível configuração seria implantar um *Contractor* e um *Contract Manager* em cada um dos grupos de servidores existentes. O *Contractor* seria responsável por coordenar o processo de monitoramento das propriedades carga de processamento e número de réplicas de um grupo, e o *Contract Manager*, responsável pelo gerenciamento do contrato. Com essa configuração, cada grupo de servidores teria seus recursos (as réplicas de servidor) gerenciados por um contrato (Código 4.6), conforme a sua demanda por carga de processamento. Deste modo, cada um dos grupos poderá declarar através do seu contrato qual a circunstância desejada para adicionar ou remover uma réplica de servidor.

4.3 Aplicação Videoconferência

Esta aplicação consiste em um sistema de videoconferência onde usuários conectados a uma rede de computadores transmitem e recebem dados de áudio e vídeo capturados de dispositivos multimídia conectados em suas máquinas. Nela, um usuário pode criar uma nova sessão de conferência ou se associar a uma já em andamento. O cenário da aplicação é composto por um conjunto de clientes, que representam terminais de sessões de videoconferência, e de elementos de suporte necessários ao estabelecimento da aplicação. A aplicação aqui apresentada possui o objetivo de exemplificar o uso de contratos de QoS CBabel e os seus gerenciamentos pelo *framework* CR-RIO, ela está sendo desenvolvida em paralelo a este trabalho [Loques et al. 2005]. A Figura 4.4 ilustra a disposição dos componentes da arquitetura da aplicação, onde:

Gerenciador de usuários: é responsável pelo controle dos usuários participantes da aplicação videoconferência. Para que um usuário possa criar ou participar de uma sessão de conferência já existente, é necessário que ele seja cadastrado no gerenciador. Depois disso, ele deve solicitar um pedido de criação ou participação em alguma sessão que ele esteja interessado ao gerenciador, que pode aceitar ou negar essa solicitação, dependendo das restrições por ele definidas (o número máximo de usuários por sessão e o número máximo de sessões que podem ser criadas);

Terminal: é o terminal de videoconferência utilizado pelo usuário final da aplicação. Através dele o usuário captura e exibe áudio e vídeo com os parâmetros desejados (formato, resolução, tamanho, taxa de *frame*). Com esse intuito, terminais disponíveis, como VIC, RAT, JMF *Studio*, *OpenPhone*, *NetMeeting*, Sip *Communicator*, poderiam ser utilizados. Na Figura 4.4, os terminais representados estão utilizando o VIC. O SDR (*Session Directory Tool*), uma ferramenta que efetua o gerenciamento de sessões de conferência, permitindo que usuários criem novas e associem-se às existentes, poderia ser utilizado. O protocolo de iniciação de sessão SIP (*Session Initiation Protocol*) [Handley et al. 1999], um protocolo responsável por iniciar sessões de comunicação interativa entre usuários, incluindo voz, vídeo, *chat*, jogos interativos e realidade virtual, também seria utilizado;

Refletor: refletores são *proxies* para a rede virtual (ou *overlay*) *multicast* que interliga os usuários participantes de uma conferência multimídia localizados em redes locais distintas. Um exemplo é a proposta Narada, que suporta o estabelecimento de uma rede virtual *multicast* provendo as funcionalidades necessárias para esse tipo de comunicação [Chu et al.

2000]. A Figura 4.4 ilustra três refletores que são responsáveis por “refletirem” os dados transmitidos pelos terminais associados aos outros refletores conectados à *overlay*, que por sua vez transmitem esses dados aos terminais a eles associados. Por exemplo, para que o terminal 1 transmita um vídeo para os terminais 3 e 5, os dados do vídeo são passados para o refletor 1, associado ao primeiro terminal, que os envia a todos os outros refletores (2 e 3) conectados à *overlay*. Os refletores 2 e 3 repassam então o vídeo para os terminais destino a eles associados;

Proxy: é utilizado para efetuar a negociação dos protocolos de conferência em favor de um dispositivo que não possua essa capacidade. No exemplo em questão, o *proxy* efetua essa negociação em favor de um PDA (*Personal Digital Assistant*), que possui recursos limitados para esse fim. O *proxy* também converte o vídeo para um formato suportado por esse tipo de dispositivo;

Gateway: permite a comunicação entre usuários que utilizam diferentes protocolos para estabelecimento de conferência. No exemplo, os terminais 5 e 6 usam o terminal NetMeeting, que utiliza o protocolo H.323 [Toga e Ott 1999] para o estabelecimento de conferências. Assim, o *Gateway* é encarregado da conversão desse protocolo para o SIP e SDR utilizados pelos outros usuários da aplicação.

No contexto desta aplicação, o gerenciamento de requisitos de qualidade requeridos pelos serviços envolvidos na videoconferência é desejado. Em um terminal, por exemplo, a disponibilidade do recurso de processamento da máquina que o executa deve ser tal que não comprometa o processamento dos dados multimídias (envio ou recebimento desses dados). Outra restrição diz respeito à banda passante entre um terminal e o seu refletor, já que se esse recurso for muito escasso, poderá comprometer o recebimento ou envio dos dados. Para que esse gerenciamento seja possível, cada usuário deve definir os parâmetros de qualidade desejados para os serviços da aplicação (por exemplo, qualidade do vídeo e do áudio) e as restrições que especificam os níveis requeridos dos recursos que serão utilizados, de forma a atender a qualidade de serviço desejada.

Utilizando o *framework* CR-RIO para prover esse gerenciamento, contratos de QoS podem ser descritos para cada usuário da aplicação, definindo os parâmetros de qualidade desejados de acordo com os níveis de recursos que eles possuem. Desta maneira, um usuário poderá definir, por exemplo, os níveis de qualidade para um vídeo de maneira compatível com os recursos que ele possui e as variações aceitáveis na disponibilidade desses recursos. Por exemplo, um usuário que possui recursos de processamento limitados poderá definir o

H.263 como o seu formato de vídeo preferencial. Caso a disponibilidade desses recursos degrade, ele poderá definir através do contrato que neste caso o vídeo pode ser transcodificado para um formato ainda inferior (H.261, por exemplo).

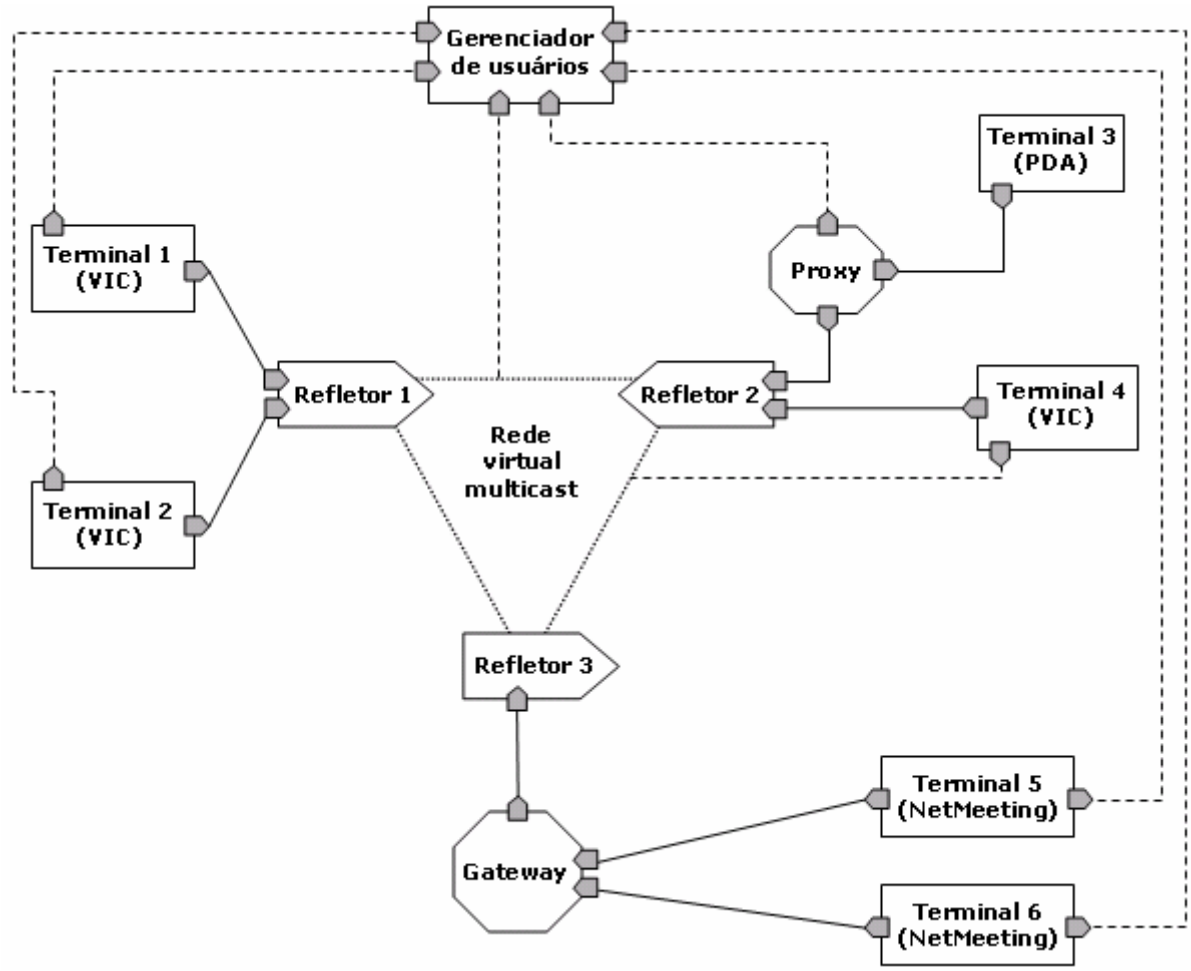

Figura 4.4 – Arquitetura da aplicação videoconferência.

4.3.1 Arquitetura da Aplicação

Para a utilização do CR-RIO, o primeiro passo é a descrição da configuração arquitetural da aplicação através da ADL CBabel. O Código 4.7 descreve essa configuração, onde são declaradas classes de módulos para o gerenciador de usuários (linhas 03 a 05), para os terminais (linhas 06 a 10) e para o PDA (linhas 11 a 13). Além disso, são especificadas as classes de conectores `Refletor` (linhas 14 a 17), `Proxy` (linhas 18 a 21) e `Gateway` (linhas 22 a 26), que representam respectivamente, os refletores, o *proxy* e o *gateway* da aplicação. Em cada uma dessas classes são declaradas instâncias dos tipos de porta `associar`, que

define o ponto de interação com o módulo `Gerenciador`, e dos tipos `transmitir-fluxo` e `receber-fluxo`, que representam os pontos de interação entre os elementos da configuração para a transmissão e recebimento de fluxo multimídia. Por fim, os comandos de configuração nas linhas 27 a 36 orientam a instanciação do gerenciador de usuários e dos refletores, assim como a topologia de ligação entre esses componentes. Neste código, não é descrita nenhuma configuração para a instanciação e ligação dos componentes terminais. A configuração para os terminais será descrita nos serviços dos contratos de QoS, associada aos requisitos não-funcionais requeridos por cada usuário da aplicação.

```
01  module Videoconferencia {
02    port associar, transmitir-fluxo, receber-fluxo;
03    module Gerenciador {
04       out port associar;
05    } gerenciador;
06    module Terminal {
07       out port associar;
08       out port transmitir_fluxo;
09       in port receber-fluxo;
10    } terminal;
11    module PDA {
12       in port receber-fluxo;
13    } pda;
14    connector Refletor {
15       in port receber-fluxo;
16       out port transmitir-fluxo;
17    } r1, r2, r3;
18    connector Proxy {
19       in port receber-fluxo;
20       out port transmitir-fluxo;
21    } proxy;
22    connector Gateway {
23       out port associar;
24       in port receber-fluxo;
25       out port transmitir-fluxo;
26    } gateway;
27    instantiate gerenciador at HostA;
28    instantiate r1 at HostB;
29    instantiate r2, proxy at HostC;
30    instantiate r3, gateway at HostD;
31    link r1.transmitir-fluxo to r2.receber-fluxo;
```

```
32    link r1.transmitir-fluxo to r3.receber-fluxo;
33    link r2.transmitir-fluxo to r1.receber-fluxo;
34    link r2.transmitir-fluxo to r3.receber-fluxo;
35    link r3.transmitir-fluxo to r1.receber-fluxo;
36    link r3.transmitir-fluxo to r2.receber-fluxo;
37 } videoconferencia;
```

Código 4.7 – Descrição arquitetural da aplicação videoconferência.

4.3.2 Categorias de QoS

Com a configuração arquitetural da aplicação descrita, o próximo passo é a descrição dos contratos de QoS para os usuários da videoconferência através da linguagem CBabel. Porém, antes de descrever os contratos, é preciso especificar as propriedades dos recursos a serem utilizados na forma de categoria de QoS (*QoSCategory*). O Código 4.8 descreve as categorias que serão utilizadas nos contratos para a aplicação em questão.

```
01  QoSCategory Conference {
02      quality: enum (Low, Medium, High) out;
03      maxUsers: numeric out;
04      numberOfUsers: numeric in;
05  };
06
07  QoSCategory VideoMedia {
08      codec: enum (H261, H263, MPEG2) out;
09      resolution: enum (Low, Medium, High) out;
10  };
11
12  QoSCategory AudioMedia {
13      codec: enum (G711, G723, GSM) out;
14      quality: enum (8BITS, 16BITS) out;
15  };
16
17  QoSCategory Processing {
18      clockFrequence: increasing numeric MHz in;
19      utilization: decreasing numeric % in;
20      memReq: increasing numeric Mbytes in;
21  };
```

Código 4.8 – Categorias de QoS para os recursos gerenciador de usuários, mídia de vídeo e áudio e processamento.

No Código 4.8, a categoria `Conference` (linhas 01 a 05) descreve propriedades relacionadas com uma conferência multimídia, onde `quality` enumera três possíveis qualidades para a conferência, `maxUsers` o número máximo de usuários permitido na conferência, e `numberOfUsers` o número de usuários corrente participando da conferência. Em `VideoMedia` (linhas 07 a 10) são descritas as propriedades relacionados com qualidade de vídeo, onde: `codec`, enumera as opções de formatos de codificação de vídeo; `resolution`, fornece três opções para a resolução do vídeo. A categoria `AudioMedia` (linhas 12 a 15) especifica as seguintes propriedades relacionadas com a qualidade de áudio: `codec`, descreve três formatos de codificação de áudio; `quality`, enumera duas opções de qualidade de áudio. Por último, na categoria `Processing` (linhas 17 a 21) são declaradas as propriedades relacionadas com os recursos de processamento de cada usuário: `clockFrequence`, a freqüência de operação do processador; `utilization`, a porcentagem de utilização da CPU; `memReq`, a quantidade de memória principal requerida. Além dessas, a propriedade `delay` – atraso na transmissão de dados no canal de comunicação – declarada na categoria `Transport`, definida no Código 4.2, e a propriedade `linkPolicy` – defini as políticas que orientam a escolha do enlace – da categoria `Client` (Código 4.4) serão utilizadas.

4.3.3 Contratos de QoS dos Usuários

Cada um dos usuários da videoconferência poderá definir o seu contrato especificando as qualidades suportadas e desejadas para a captura e exibição dos dados multimídia, assim como os níveis dos recursos locais requeridos. Porém, como forma de simplificar a exemplificação dessa descrição, serão apresentados três contratos distintos: um para os usuários com terminais comuns (VIC), outro para os usuários com terminais *NetMeeting* e outro para o usuário com o dispositivo PDA.

- Usuário VIC

O Código 4.9 contém o contrato para um usuário que utiliza terminal comum (VIC), por exemplo, o terminal 1 da Figura 4.4. Esse contrato descreve os seguintes serviços:

(1) `sHighQuality` (linhas 02 a 05) é o serviço pelo qual o usuário irá criar ou associar-se a um conferência multimídia. É o serviço preferencial do contrato, definindo como qualidade desejada para o vídeo e o áudio a mais alta. A descrição do serviço orienta a

iniciação de um terminal em um determinado *host* (linha 03), caso os recursos de processamento do usuário satisfaçam as restrições definidas no perfil `highProc` (linhas 26 a 29). Os níveis dos recursos especificados nessas restrições de processamento estão relacionados com a qualidade exigida por este serviço na conferência. A outra configuração arquitetural para este serviço (linha 04) orienta a ligação do terminal iniciado com o refletor `r1`, caso as restrições de validação dos perfis `highQuality`, `maxUsers`, `transport` sejam satisfeitas. O perfil `hiqhQuality` (linhas 36 a 42) especifica a qualidade desejada para o vídeo e o áudio que serão recebidos e transmitidos pelo terminal do usuário. Para o vídeo é desejado formato MJPEG com alta resolução (linhas 37 e 38), e para o áudio a qualidade GSM com 16 BITS (linhas 39 e 40). A última restrição deste perfil (linha 41) indica a qualidade desejada para uma conferência criada pelo usuário. O perfil `maxUsers` (linhas 52 a 55) define como condição de validação que, ao se associar a uma conferência, o número máximo de usuários permitidos na conferência não pode ter sido atingido (linha 53). A outra restrição apenas orienta, no momento de criação de uma conferência pelo usuário, qual é o número máximo de usuários permitidos (linha 54). O perfil `transport` (linhas 22 a 24) define o atraso permitido no canal de comunicação entre o usuário e o refletor. De acordo com os tipos das propriedades (*in* ou *out*) definidas nas categorias utilizadas nos perfis associados a este serviço, ele somente será estabelecido caso todas as restrições do perfil `highProc`, a do perfil `transport` e a primeira de `maxUsers` possam ser atendidas. As demais restrições apenas especificam os parâmetros de qualidade desejados;

(2) `sLowQuality` (linhas 07 a 10) possui o mesmo intuito funcional que o primeiro serviço, porém com restrições de qualidade inferiores. Sua descrição contém as mesmas orientações de configuração arquitetural que `sHighQuality`, mas associadas com alguns perfis diferentes. O perfil `lowProc` (linhas 31 a 34) define restrições para os recursos de processamento menos exigentes, já que a qualidade inferior desejada por este serviço requer menos desses recursos. `lowQuality` (linhas 44 a 50) define qualidades inferiores para vídeo e áudio, onde para o vídeo é especificado o formato H.261 com baixa resolução (linhas 45 e 46) e para o áudio o formato G711 com 8BITS (linhas 47 e 48). A última restrição indica que a qualidade de uma conferência criada pelo usuário será baixa (linha 49). Os demais perfis são os mesmos utilizados no serviço anterior;

(3) `sChangeReflector` (linhas 12 a 14) é o serviço responsável por conectar um usuário a outro refletor, caso o nível do canal de comunicação com o atual refletor degrade ao ponto de violar as restrições impostas a esse recurso. Esse nível de degradação é especificado

na primeira restrição de validação do perfil `changeReflector` (linhas 57 a 60) associado. A outra restrição desse perfil (linha 59) apenas orienta a busca por um outro refletor com melhor qualidade no enlace.

A cláusula de negociação de serviços do contrato (linhas 16 a 19) define `sHighQuality` como o serviço de maior prioridade (linha 17). Se ele estiver em vigor e suas restrições não puderem mais ser satisfeitas, ocorrerá a tentativa de estabelecimento dos próximos serviços, `sChangeReflector` ou `sLowQuality`, respeitando essa ordem. Se a violação ocorrer devido a uma degradação nos níveis dos recursos de processamento do usuário que não permita satisfazer as restrições do perfil `highProc`, o serviço `sLowQuality`, que exige menos desses recursos, será estabelecido, caso esses níveis sejam adequados com suas restrições, caso contrário, nenhum serviço poderá ser estabelecido e o estado *out-of-service* será alcançado. Esse estado também será alcançado caso a condição contida na primeira restrição do perfil `maxUsers` seja violada, já que se isso ocorrer mais nenhum outro serviço poderá ser estabelecido. Porém, se a violação se referir à restrição do perfil `transport` (atraso no canal de comunicação), o serviço `sChangeReflector` será estabelecido e tentará buscar outro refletor cujo enlace possua o nível de qualidade requerido.

```
01  contract {
02    service {
03      instantiate terminal at HostCliente with highProc;
04      link terminal to r1 with highQuality, maxUsers, transport;
05    } sHighQuality;
06
07    service {
08      instantiate terminal at HostCliente with lowProc;
09      link terminal to r1 with lowQuality, maxUsers, transport;
10    } sLowQuality;
11
12    service {
13      link terminal to reflector with changeReflector;
14    } sChangeReflector;
15
16    negotiation {
17      not sHighQuality -> ( sChangeReflector || sLowQuality );
18      sLowQuality -> ( sChangeReflector || sHighQuality );
19    }
20  } videoconferencia;
```

```
21
22  profile {
23    Transport.delay < 50;
24 } transport;
25
26  profile {
27    Processing.utilization <= 50;
28    Processing.memReq >= 40;
29  } highProc;
30
31  profile {
32    Processing.utilization <= 70;
33    Processing.memReq >= 20;
34  } lowProc;
35
36  profile {
37    VideoMedia.codec = MPEG2;
38    VideoMedia.resolution = High;
39    AudioMedia.codec = GSM;
40    AudioMedia.quality = 16BITS;
41    Conference.quality = High;
42  } highQuality;
43
44  profile {
45    VideoMedia.codec = H261;
46    VideoMedia.resolution = Low;
47    AudioMedia.codec = G711;
48    AudioMedia.quality = 8BITS;
49    Conference.quality = Low;
50  } lowQuality;
51
52  profile {
53    Conference.numbeOfUsers < Conference.MaxUsers;
54    Conference.maxUsers = 10;
55  } maxUsers;
56
57  profile {
58    Transport.delay >= 50;
59    Client.linkPolicy = bestLink;
60  } changeReflector;
```

Código 4.9 – Contrato de QoS para um usuário com terminal VIC na videoconferência.

Na regra de negociação para o serviço de menor prioridade `sLowQuality` (linha 18) é definido que se ele estiver em execução e qualquer um dos outros serviços puderem ser estabelecidos, eles serão. Isso porque se o perfil `changeReflector`, associado ao primeiro serviço do lado direito dessa regra (`sChangeReflector`), se tornar válido, implicará na violação do perfil `transport` associado a `sLowQuality`. A outra opção é o serviço `sHighQuality`, e caso seus perfis se tornem válidos, ele deverá ser estabelecido, pois é o serviço de maior preferência.

- Usuário NetMeeting

O próximo contrato, apresentado no Código 4.10, destina-se ao usuário que utiliza o terminal *NetMeeting* para participar da videoconferência, como os ilustrados nos terminais 5 e 6 da Figura 4.4. Esse contrato é descrito de forma muito similar ao definido anteriormente, porém com algumas diferenças necessárias para considerar o uso de um *gateway* encarregado da conversão do protocolo H.323, utilizado pelo *NetMeeting*, para o SIP e o SDR, utilizados pelos outros usuários da aplicação. O uso desse *gateway* é especificado nas configurações dos serviços `sHighQuality` e `sLowQuality`, onde o `terminal` é conectado ao refletor `r3` através do conector `gateway`. A outra diferença é o serviço `sChangeGateway`, que neste contrato busca um outro `gateway` onde a restrição para o canal de comunicação entre ele e o usuário possa ser satisfeita.

```
01 contract {
02  service {
03    instantiate terminal at HostCliente with highProc;
04    link terminal to r3 by gateway with highQuality, maxUsers, transport;
05  } sHighQuality;
06
07  service {
08    instantiate terminal at HostCliente with lowProc;
09    link terminal to r3 by gateway with lowQuality, maxUsers, transport;
10  } sLowQuality;
11
12  service {
13    link terminal to reflector with changeGateway;
14  } sChangeGateway;
15
16  negotiation {
17    not sHighQuality -> ( sChangeGateway || sLowQuality );
```

```
18    sLowQuality -> ( sChangeGateway || sHighQuality );
19  }
20 } videoconferencia;
21
22  profile {
23    Transport.delay < 50;
24 } transport;
25
26  profile {
27    Processing.utilization <= 50;
28    Processing.memReq >= 40;
29  } highProc;
30
31  profile {
32    Processing.utilization <= 70;
33    Processing.memReq >= 20;
34  } lowProc;
35
36  profile {
37    VideoMedia.codec = MPEG2;
38    VideoMedia.resolution = High;
39    AudioMedia.codec = GSM;
40    AudioMedia.quality = 16BITS;
41    Conference.quality = High;
42  } highQuality;
43
44  profile {
45    VideoMedia.codec = H261;
46    VideoMedia.resolution = Low;
47    AudioMedia.codec = G711;
48    AudioMedia.quality = 8BITS;
49    Conference.quality = Low;
50  } lowQuality;
51
52  profile {
53    Conference.numbeOfUsers < Conference.MaxUsers;
54    Conference.maxUsers = 10;
55  } maxUsers;
56
57  profile {
58    Transport.delay >= 50;
```

```
59    Client.linkPolicy = bestLink;
60  } changeGateway;
```

Código 4.10 – Contrato de QoS para um usuário com terminal NetMeeting na videoconferência.

- Usuário PDA

O último contrato é para o usuário que utiliza um dispositivo PDA para participar da videoconferência, apenas recebendo os dados multimídia transmitidos pelos outros usuários. Ele é descrito no Código 4.11 de forma similar ao contrato definido anteriormente, porém com algumas diferenças necessárias para considerar os limites de recursos inerentes a este tipo de dispositivo.

Neste contrato são definidos apenas dois serviços. O primeiro, `sConference` descrito nas linhas 02 a 05 do Código 4.11, é responsável por iniciar o usuário PDA em uma conferência multimídia. As configurações arquiteturais para esse serviço se encarregam de iniciar o módulo que representa esse usuário e de ligá-lo ao refletor `r2` através do conector `proxy`. Esse conector se encarrega de fazer a negociação requerida, através dos protocolos utilizados na aplicação, para que este usuário possa participar de alguma conferência. Além disso, ele se encarrega de transcodificar os formatos de áudio e vídeo para os suportados por esse tipo de dispositivo.

```
01  contract {
02    service {
03      instantiate pda at HostCliente with proc;
04      link pda to r2 by proxy with maxUsers, transport;
05    } sConference;
06
07    service {
08      link pda to reflector by proxy with changeProxy;
09    } sChangeProxy;
10
11    negotiation {
12      sConference -> sChangeProxy;
13    }
14  } videoconferencia;
15
16  profile {
17    Transport.delay < 70;
18 } transport;
```

```
19
20  profile {
21    Processing.utilization <= 60;
22    Processing.memReq >= 10;
23  } proc;
24
25  profile {
26    Conference.numbeOfUsers < Conference.MaxUsers;
27  } maxUsers;
28
29  profile {
30    Transport.delay >= 70;
31    Client.linkPolicy = bestLink;
32  } changeProxy;
```

Código 4.11 – Contrato de QoS para um usuário com dispositivo PDA na videoconferência.

Os perfis `proc` (linhas 20 a 23), `maxUsers` (linhas 25 a 27) e `transport` (linhas 16 a 18) definem as restrições para o serviço `sConference`. Em `proc` são definidas as restrições que dizem respeito aos níveis requeridos para o recurso de processamento, neste caso menos exigentes que as dos contratos anteriores, já que o formato dos dados multimídia enviados para o PDA será de qualidade inferior, exigindo assim menos recursos para processá-los. O perfil `maxUsers` define a condição de entrada em uma conferência, de acordo com o número máximo de usuários permitido. Por fim, em `transport` é definida a restrição de atraso no canal de comunicação entre o usuário e o *proxy*.

O outro serviço definido neste contrato, `sChangeProxy` (linhas 07 a 09 do Código 4.11), é encarregado de procurar um outro `proxy` para o PDA, caso a restrição do canal de comunicação com o `proxy` atual seja violada. Essa violação é descrita na primeira restrição do perfil `changeProxy` (linhas 29 a 32). A outra restrição (linha 31) apenas orienta a busca por um enlace entre o PDA e o `proxy` que atenda a restrição de qualidade requerida.

A cláusula de negociação de serviços para este contrato (linhas 11 a 13 do Código 4.11) define que o serviço de maior preferência é `sConference`. Caso ele esteja em andamento e o perfil do serviço `sChangeProxy` torne-se válido, este serviço será estabelecido. Se a violação nos perfis do serviço `sConference` não se referir à restrição do canal de comunicação, nenhum outro serviço poderá ser estabelecido e o estado *out-of-service* será alcançado.

4.3.4 Gerenciamento dos Contratos

Neste ponto, para que um usuário da aplicação videoconferência possa ter seus serviços gerenciados de acordo com o contrato de QoS CBabel por ele descrito, é preciso configurar os componentes do *framework* CR-RIO no ambiente desta aplicação. Primeiro é preciso definir os *Resource Agents* que irão monitorar as propriedades de entrada das categorias de QoS utilizadas no contrato (`Conference`, `VideoMedia`, `AudioMedia`, `Processing`, `Transport` e `Client`). Esses agentes implementarão ou utilizarão algum mecanismo já existente que permita o monitoramento dessas propriedades. Depois é preciso especializar o *Contractor* ao contrato do usuário para tratar as verificações das restrições declaradas nos perfis associados a esse contrato, de acordo com os valores monitorados pelos agentes. É preciso então implantar um *Contractor* em cada *host* que contém propriedades a serem monitoradas. Uma possível configuração seria implantar um no *host* do usuário, responsável pelas restrições referentes às propriedades dos recursos de processamento e transporte, e outro no *host* do Gerenciador de Usuários, responsável pelas propriedades relacionadas com o número de usuários em uma conferência.

Além disso, o *Configurator* para esse contrato deve ser capaz de mapear as configurações arquiteturais descritas nos serviços em ações do nível de sistema que efetivem o estabelecimento do serviço de videoconferência para o usuário. É necessário, por exemplo, que os parâmetros de qualidade de áudio e vídeo especificados nos perfis associados ao serviço sejam configurados no terminal de videoconferência do usuário. Também é necessário neste contexto, algum mecanismo de descoberta de recursos que permita a busca por enlaces que atendam os requisitos de qualidade desejados.

Como no cenário desta aplicação cada usuário define o seu contrato de QoS conforme a qualidade desejada e os recursos que ele possui, o componente *Contract Manager*, responsável pelo gerenciamento de contrato, pode ser implantado em cada um dos *hosts* dos usuários. Desta maneira, o CR-RIO gerenciará os serviços de cada usuário, conforme as restrições de qualidade definidas em seu contrato, de forma independente. Essa solução distribuída é mais adequada, pois a falha de um *Contract Manager* não interfere no gerenciamento dos contratos dos outros usuários da aplicação.

4.4 Conclusão do Capítulo

Neste capítulo foram apresentados exemplos de aplicações que possuem requisitos de QoS em diferentes contextos de serviço. Esses exemplos permitiram demonstrar a funcionalidade da proposta CR-RIO no gerenciamento de aplicações com esse tipo de requisito. Com isso, foi possível validar através de exemplos a linguagem de descrição de contratos e o modelo de infra-estrutura de suporte a gerenciamento de contratos de QoS apresentados no Capítulo 3.

O primeiro exemplo de aplicação, uma aplicação de vídeo sob demanda constituída de um servidor multimídia que transmite vídeos a clientes, permitiu exemplificar o uso do contrato de QoS na descrição de serviços que definem a qualidade do vídeo transmitido. Esses serviços foram associados a restrições, dos recursos de processamento do cliente e da disponibilidade de banda passante no canal de comunicação entre o cliente e o servidor, que especificam o que é necessário, em termos de recursos, para a transmissão do vídeo em uma determinada qualidade. Foi mostrado como o CR-RIO é utilizado para monitorar os níveis desses recursos, permitindo identificar a adequabilidade de estabelecimento de um serviço ou de mantê-lo em execução. Através desse gerenciamento, um vídeo sendo transmitido a um cliente pode ter a sua qualidade degradada de forma automática, contando com um mecanismo de transcodificação de formato, para se adequar à escassez de recursos, em um determinado momento da transmissão.

O segundo exemplo é uma aplicação cliente-servidor Web, composta por clientes que enviam requisições de conteúdo a grupos de servidores. Trata-se de um cenário mais complexo que o primeiro, onde foi possível exemplificar o uso do *framework* CR-RIO para gerenciar, de forma distribuída, os segmentos desta aplicação. Para isso, foram descritos dois contratos de QoS, um para o cliente e outro para os grupos de servidores. Através do seu contrato, cada cliente poderia especificar a qualidade requerida, no caso o tempo de resposta observado para o atendimento de uma requisição por ele enviada. Com as informações do contrato e os componentes da infra-estrutura de suporte do *framework* implantados em cada cliente, o CR-RIO poderia gerenciar o atendimento do serviço ao cliente, de acordo com as restrições de QoS por ele desejadas. Foi mostrado que o CR-RIO consegue, através de um serviço não-funcional definido no contrato, fazer com que o cliente se conecte a um grupo de servidores que possua um enlace de melhor qualidade, para garantir a continuidade do serviço, quando a disponibilidade do enlace com o grupo atual degradar. Para isso, seria

necessário utilizar um mecanismo de descoberta de recursos, que possibilite a busca por enlaces que atendam às restrições de qualidade exigidas para esse recurso. Esse gerenciamento dinâmico e distribuído também foi mostrado para o contrato de gerenciamento dos grupos de servidores, porém, neste caso, o gerenciamento se deu nos serviços de controle de réplicas de servidor, em cada um dos grupos presentes na aplicação, de acordo com suas necessidades por recursos de processamento.

O terceiro exemplo descreve o cenário de uma aplicação de videoconferência composta por usuários que possuem características diferentes, o que exige configurações e qualidade distintas para os serviços por eles definidos através dos contratos de QoS. Levando em consideração este contexto, são apresentados três contratos de QoS, sendo um para cada tipo de usuário do cenário dessa aplicação. Foi mostrado como os usuários dessa aplicação poderiam ter os seus contratos gerenciados pelos componentes do CR-RIO de forma distribuída e independente. O cenário distribuído e complexo dessa aplicação permitiu mostrar a viabilidade do *framework* CR-RIO no gerenciamento independente dos contratos para cada usuário, conforme as restrições de qualidade por eles requeridas.

O próximo capítulo descreve a implementação do *framework* CR-RIO, de acordo com o modelo proposto e validado através dos exemplos deste capítulo. A descrição da implementação fornece maiores detalhes sobre o funcionamento do *framework*, permitindo compreender de forma mais ampla cada um de seus componentes e o processo de gerenciamento de contrato de QoS.

Capítulo 5

Implementação da Proposta

A implementação de suporte a contratos de QoS do *framework* CR-RIO foi feita na linguagem Java, por ela oferecer portabilidade e utilizar mecanismos que facilitam a implementação de aplicações com requisitos de extensibilidade e comunicação remota. Seu mecanismo de reflexão estrutural, por exemplo, foi utilizado para permitir que novas classes específicas aos contratos de QoS que serão gerenciados pudessem ser implantadas no *framework* sem a necessidade de alterá-lo. O mecanismo de comunicação remota Java RMI (Java *Remote Method Invocation*) também foi utilizado para o estabelecimento de comunicação entre os componentes distribuídos do CR-RIO.

Este capítulo expõe alguns aspectos da implementação da proposta CR-RIO. Inicialmente é descrito como a implementação foi estruturada de acordo com o modelo apresentado no Capítulo 3. Através da modelagem proposta para a implementação é mostrado, então, o seu funcionamento no gerenciamento de aplicações com requisitos de QoS. Depois são descritos os passos necessários para que a implementação desenvolvida gerencie um contrato para uma aplicação. Por fim, é apresentado um exemplo de aplicação de vídeo sob demanda que foi desenvolvido para testar e validar a implementação do *framework.*

5.1 Modelagem

Tomando como base o modelo conceitual apresentado para o *framework* CR-RIO no Capítulo 3, um modelo de classes foi proposto para sua implementação. Esse modelo pode ser dividido em duas partes principais: uma que modela a estrutura de dados para os elementos

compostos nos contrato de QoS CBabel e outra modelando os componentes da infra-estrutura de suporte a gerenciamento desses contratos.

5.1.1 Elementos do Contrato de QoS CBabel

A Figura 5.1 ilustra as classes que representam os elementos que constituem um contrato de QoS e como elas se relacionam. Para tornar o diagrama mais claro, os métodos para definir e buscar os valores de cada atributo (métodos *set* e *get*) foram omitidos. O diagrama é então constituído pelas seguintes classes:

- `QoSContract`: é a classe que representa o contrato de QoS CBabel. Ela é composta pelos atributos nome do contrato (`name`), um ou mais serviços (`services`) e uma ou mais regras de transição de serviço (`transitions`). Os atributos `services` e `transitions` armazenam uma ou mais instâncias das classes `Service` e `Transition`, respectivamente;
- `Service`: representa um serviço declarado no contrato. Esta classe é composta pelo nome do serviço (`name`) e uma ou mais tarefas de configuração arquitetural (`tasks`). O atributo `tasks` é composto por um ou mais elementos do tipo `Task`;
- `Task`: esta classe representa uma tarefa de configuração arquitetural declarada em um serviço do contrato, sendo composta pelos seguintes atributos: `adlTag`, a primitiva arquitetural CBabel utilizada na configuração, como `link` e `instantiate`; `component1` e `component2`, os nomes das instâncias de módulos utilizados na configuração, como `cliente` e `servidor` em `link cliente to servidor`; `host1` e `host2`, os nomes dos *hosts* onde foram instanciados os módulos; `profiles`, os perfis de qualidade associados à tarefa de configuração, que é constituído por um ou mais elementos da classe `Profile`;
- `Profile`: é composta pelos atributos de um perfil de qualidade declarado no contrato – nome do perfil (`name`) e o conjunto de restrições nele declaradas (`restrictions`). Esse conjunto de restrições é formado por um ou mais elementos da classe `Restriction`;
- `Restriction`: contém as informações referentes a uma restrição declarada em um perfil, organizadas nos atributos: `qosCategory`, o nome da categoria de QoS

ao qual a propriedade utilizada na restrição pertence; `property`, nome da propriedade utilizada; `equalIntValue`, `equalStrValue`, `minValue` e `maxValue`, onde será armazenado o valor declarado para a propriedade na restrição, dependendo do seu tipo (numérico ou *string*) e do operador lógico utilizado (<, >, = ou ==);

- `QoSCategory`: guarda os dados de uma categoria de QoS – nome da categoria (`name`) e as propriedades que ela contém (`properties`, conjunto de elementos do tipo `Property`);
- `Property`: representa uma propriedade declarada em uma categoria de QoS. Os seguintes atributos a descrevem: `name`, nome da propriedade; `entryProperty`, se ela é uma propriedade de entrada ou não (saída); `type`, o seu tipo – `numeric` ou `enum`; `unit`, sua unidade de medida no caso do tipo `numeric`; `preference`, se a preferência por valores é crescente ou decrescente; `enumElements`, os valores declarados para a enumeração, no caso do tipo `enum`;
- `Transition`: contém as informações referentes a uma regra de transição declarada na cláusula de negociação de serviço do contrato. Essas informações são organizadas em: `fromService`, o nome do serviço no lado esquerdo da regra; `not`, se a palavra reservada `not` foi utilizada ou não; `toServices`, os nomes dos serviços declarados no lado direito da regra.

Esse modelo é utilizado pelo *framework* CR-RIO para armazenar as informações obtidas de um contrato de QoS no processo de sua interpretação. Portanto, para cada contrato que é carregado no *framework*, uma instância da classe `QoSContract` é criada contendo as informações descritas no contrato, necessárias no seu gerenciamento. Nesse processo, o arquivo contendo a descrição arquitetural da aplicação, descrito na ADL CBabel, pode ser consultado com a finalidade de resolver os nomes dos *hosts* onde foram instanciados os módulos referenciados nas configurações dos serviços do contrato. Isso é necessário, pois nem sempre essa informação está contida na descrição do contrato, por exemplo, como as configurações dos serviços descritos no contrato para a aplicação de vídeo sob demanda (Código 4.3), que não possui a informação do nome do *host* onde foi instanciado o módulo `servidor`. Essa informação está contida na descrição da arquitetura dessa aplicação (Código 4.1) e será buscada quando esse contrato for interpretado pelo CR-RIO.

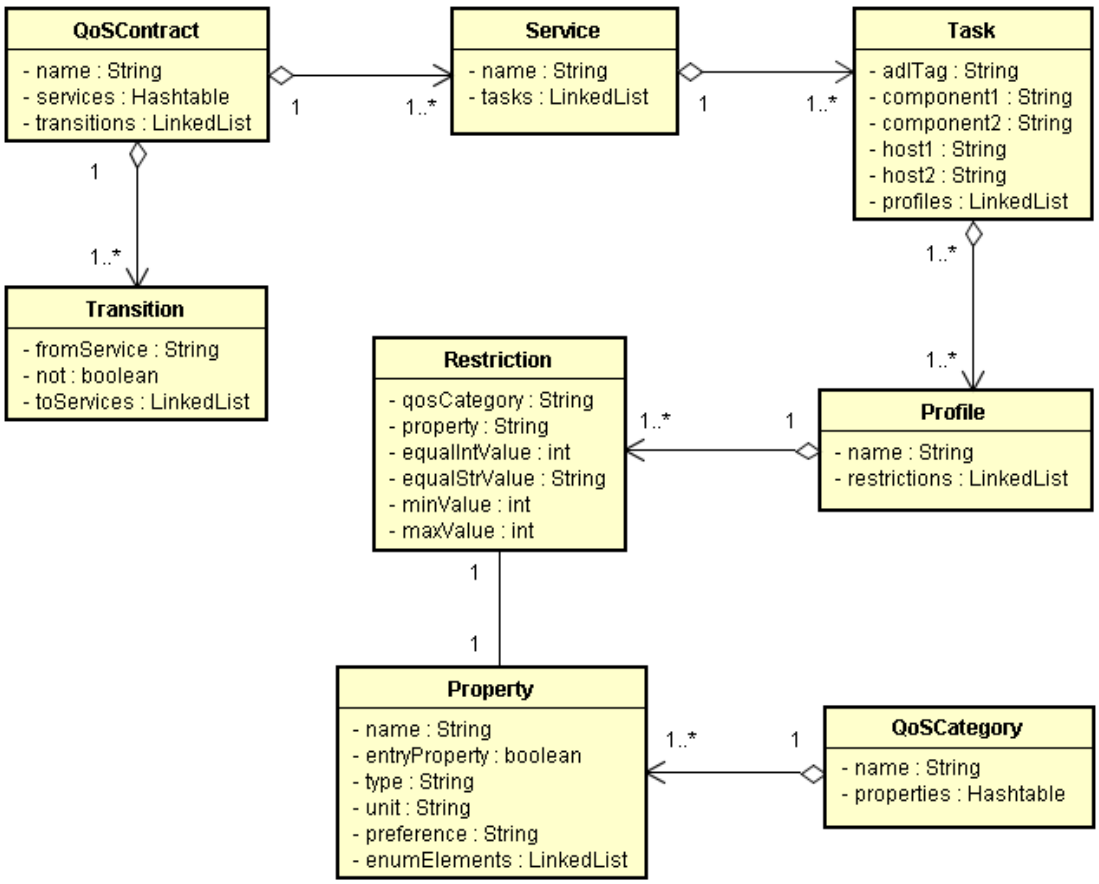

Figura 5.1 – Diagrama de classes dos elementos que constituem um contrato de QoS.

5.1.2 Elementos da Infra-Estrutura de Suporte

A outra parte da modelagem refere-se ao modelo de classes para os elementos da infra-estrutura de suporte a gerenciamento de contrato, ou seja, os elementos que implementam o mecanismo de gerenciamento de contrato no CR-RIO. Esse modelo é ilustrado no diagrama da Figura 5.2, onde somente os atributos e métodos mais importantes foram mostrados. As seguintes classes compõem esse diagrama:

- `CrrioGUI`

A classe `CrrioGUI` implementa uma interface gráfica que facilita a utilização do CR-RIO. Através dela o usuário pode selecionar contratos de QoS e disparar o processo de gerenciamento pelo *framework* de cada um deles. Além disso, ela mantém o usuário informado sobre o andamento do processo de gerenciamento dos contratos, disponibilizando informações como o serviço corrente e sobre a negociação dos serviços em cada um deles.

Pela interface, o usuário pode também interromper o gerenciamento de um contrato em andamento.

- `ContractData`

Esta classe armazena as informações necessárias no gerenciamento de um contrato carregado no *framework*. Ela foi criada com intuito de encapsular essas informações, permitindo o gerenciamento simultâneo de vários contratos. Dentre as informações que ela mantém, há o nome do serviço que está estabelecido, os dados contidos na descrição do contrato (instância da classe `QoSContract`), as referências remotas para os *Contractors* envolvidos na monitoração das propriedades de categorias de QoS utilizadas nos perfis associados aos serviços do contrato, e a referência para o elemento *Configurator*.

- `ContractParser` e `CategoryParser`

São as classes responsáveis, respectivamente, pela interpretação sintática de categorias e contratos de QoS. As categorias de QoS são carregadas de um repositório e interpretadas quando o *framework* é carregado. Já um contrato de QoS para uma aplicação, é carregado e interpretado quando o usuário solicita o seu gerenciamento pela interface `CrrioGUI`.

- `QoSContract`

Contém as informações obtidas pelo processo de interpretação da descrição de um contrato de QoS para uma aplicação. É a mesma classe apresentada anteriormente e ilustrada na Figura 5.1.

- `ContractManager`

É a classe principal do *framework*, implementando as funcionalidades do Gerente de Contrato do modelo de componentes da infra-estrutura de suporte a gerenciamento de contrato do CR-RIO (*Contract Manager*, apresentado na seção 3.2). Sendo assim, ela é responsável pela coordenação do processo de gerenciamento de contrato, que envolve os seguintes passos:

- Interpretação da descrição de categorias e contratos de QoS;
- Tentativa de estabelecimento de serviço e para isso verificação das suas restrições de qualidade associadas;
- Estabelecimento de serviço e acesso a mecanismos de configuração que efetuam configurações na infra-estrutura de suporte a uma aplicação ou nela própria.

- `RemoteContractManagerImpl`

A função desta classe é disponibilizar alguns serviços oferecidos por `ContractManager` via rede (RMI). É através desses serviços que o gerente de contrato é informado quando variações na disponibilidade dos recursos utilizados por uma aplicação violam as restrições definidas no contrato para o serviço ativo ou satisfazem as restrições de algum serviço de maior preferência. Esses serviços disponibilizados de forma remota são declarados na interface `RemoteContractManager`, conforme o modelo de implementação para utilização do mecanismo RMI.

- `Contractor`

Classe que implementa as funcionalidades do Contratador de QoS do modelo de componentes da infra-estrutura de suporte do CR-RIO (*Contractor*, apresentado na seção 3.2). Ela é responsável por verificar se alterações nos valores dos níveis dos recursos utilizados pela aplicação violam as restrições de qualidade definidas para o serviço corrente ou satisfazem as restrições de um possível serviço de maior prioridade. Se algum desses casos for constatado, o Gerente de Contrato (`ContractManager`) é notificado, via RMI, iniciando então a tentativa de estabelecimento de um outro serviço – o próximo serviço na regra de transição para o serviço violado ou o serviço de maior preferência, descrito nessa mesma regra, cujas restrições foram validadas pelo novo estado do sistema. Para obtenção dos níveis dos recursos utilizados, a classe `Contractor` gerencia o processo de monitoramento realizado pelos Agentes de Recurso (`ResourceAgent`).

Uma especialização da classe `Contractor` deve ser criada para cada contrato de QoS específico a uma aplicação. Essa especialização é feita por herança, onde a classe especializada deve obrigatoriamente implementar os métodos abstratos definidos em `Contractor`. Um desses métodos é `checkRestriction`, que se encarrega de verificar se os valores monitorados satisfazem as restrições declaradas nos perfis de qualidade associados aos serviços do contrato. O outro método é `loadResourceAgents`, que determina quais agentes de recurso serão carregados, de acordo com a necessidade por monitoramento das propriedades de categorias de recurso utilizadas no contrato. Desta forma, a classe especializada consiste apenas em implementar, através desses dois métodos, as funcionalidades do `Contractor` que são específicas a cada contrato. Esse mecanismo é conhecido como o padrão de projeto *Template Method* [Gamma et al. 1995].

- `RemoteContractorImpl`

Disponibiliza alguns serviços oferecidos pela classe `Contractor` via rede (RMI). Esses serviços são declarados na interface `RemoteContractor`, conforme exigido pelo modelo de implementação do mecanismo RMI. É através desses serviços que o Gerente de Contrato solicita ao *Contractor*, via RMI, verificações das restrições associadas aos serviços e o gerenciamento do processo de monitoração dos recursos necessários a uma aplicação.

- `ResourceAgent`

Implementa a funcionalidade do Agente de Recurso do modelo de componentes da infra-estrutura de suporte do CR-RIO (*Resource Agent*, apresentado na seção 3.2), ou seja, efetua o monitoramento dos níveis das propriedades dos recursos utilizados por uma aplicação. Para cada tipo de recurso (por exemplo, processamento e transporte) uma classe especializando `ResourceAgent`, através do mecanismo de herança, deve ser criada. Ao herdar `ResourceAgent` a classe especializada deve implementar o método abstrato `getResourceValues`, seguindo também o modelo definido no padrão de projeto *Template Method*. Essa implementação é responsável por efetuar a leitura dos valores das propriedades do recurso ao qual a especialização se refere. O processo de monitoramento implementado em `ResourceAgent` consiste em obter esses valores e no envio deles ao *Contractor*, quando mudanças na disponibilidade de um recurso são detectadas.

Para determinar a ocorrência de mudanças, podem ser implementadas nas classes especializadas de `ResourceAgent` funções que permitam, por exemplo, definir uma faixa de variação para o valor de uma propriedade de recurso monitorada, que será considerada para decidir sobre a ocorrência de mudanças significativas. Desta maneira, o envio dos valores monitorados ao *Contractor* ao qual um agente está associado somente acontecerá se a alteração no valor de uma propriedade monitorada ultrapassar a faixa de valores definida na função que determina se uma mudança é significativa.

- `ResourceState`

`ResourceState` é a superclasse das especializações que definirão os atributos referentes às propriedades de uma categoria de recurso, que são descritas na forma de categoria de QoS CBabel (*QoSCategory*). Assim, para cada tipo de recurso é preciso criar uma especialização de `ResourceState`, que conterá os atributos correspondentes às propriedades declaradas em uma categoria de QoS. Cada especialização de `ResourceAgent`

para um determinado recurso possui um `ResourceState` associado, que guardará em seus atributos os valores das propriedades do recurso obtidos no processo de monitoração.

- `ProbeTask`

Implementa o mecanismo responsável pelo monitoramento periódico das propriedades de um recurso, ou seja, dispara de tempo em tempo o processo de monitoramento implementado em `ResourceAgent`. O período em que esse processo deve ocorrer é definido em cada especialização de `ResourceAgent`.

- `Configurator`

Esta interface desempenha o papel do Configurador presente no modelo de componentes da infra-estrutura de suporte do CR-RIO (*Configurator*, apresentado na seção 3.2). Ela deve ser implementada para cada contrato específico, com o objetivo de oferecer o mecanismo de efetivação das configurações descritas nos serviços do contrato. Sendo assim, será sua implementação que receberá do Gerente de Contrato (`ContractManager`) as configurações referentes a um serviço a ser estabelecido.

- `ContractorFactory` e `ConfiguratorFactory`

São responsáveis pela instanciação de objetos das classes especializadas de `Contractor` e `Configurator` específicas a cada contrato. Elas utilizam o mecanismo de reflexão estrutural oferecido pela linguagem Java e uma convenção de nomenclatura para definição dos nomes das classes especializadas (*v.* seção 5.2), de forma a permitir que o *framework* consiga localizar as classes especializadas para cada contrato sem precisar ser modificado.

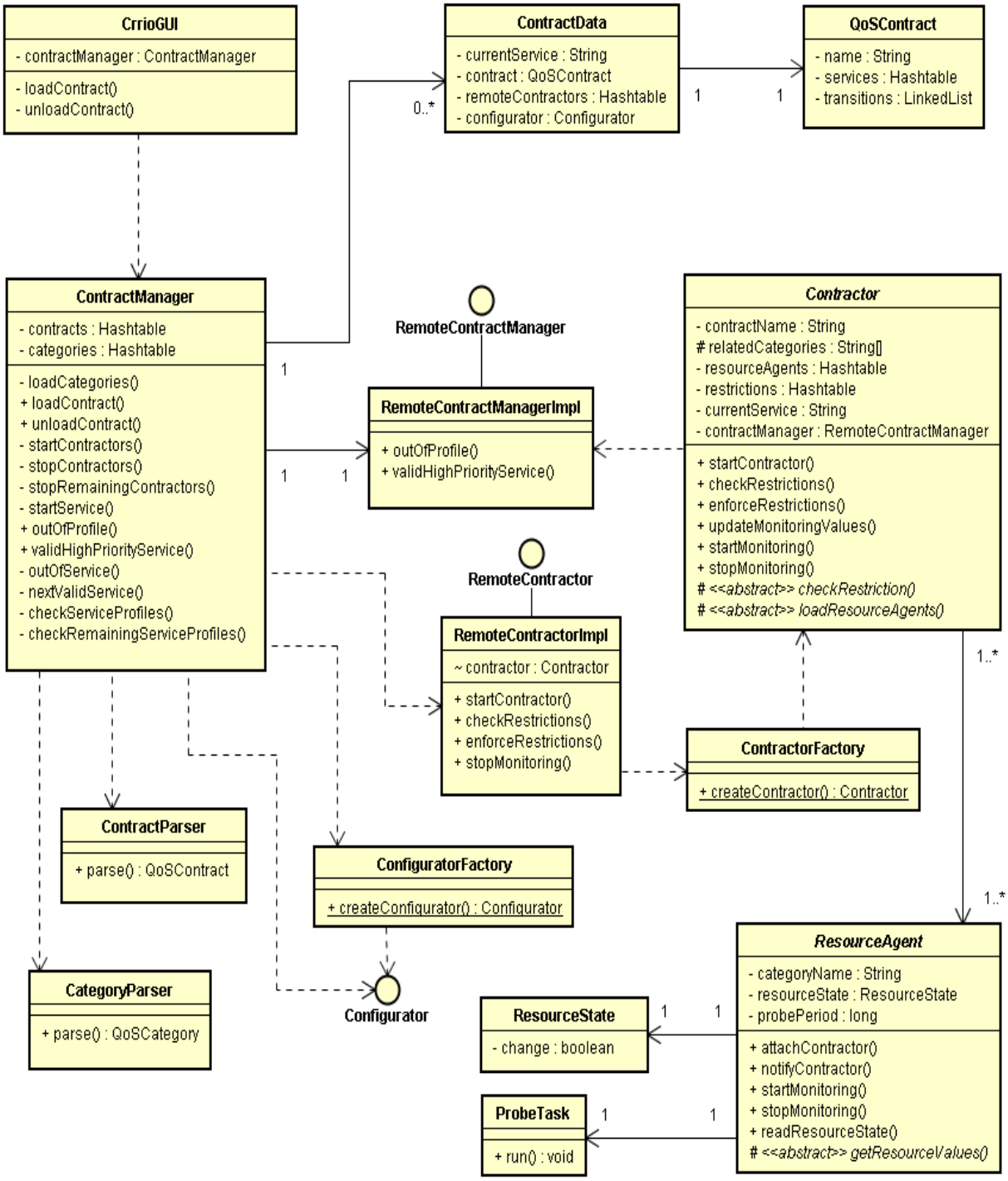

Figura 5.2 – Diagrama de classes dos elementos da infra-estrutura de suporte do CR-RIO.

5.1.3 Funcionamento do Modelo

O diagrama de interação ilustrado na Figura 5.3 mostra como os elementos do modelo de classes apresentado na Figura 5.2 interagem no processo de carregamento de um contrato

pelo CR-RIO. Para simplificar o entendimento do diagrama, somente os passos mais importantes são mostrados (a comunicação via RMI, por exemplo, é omitida).

Para iniciar o processo de gerenciamento de um contrato para uma aplicação no CR-RIO, o usuário do *framework* seleciona o contrato desejado e solicita o seu gerenciamento através da interface gráfica `CrrioGUI`. É a partir desse momento que o diagrama da Figura 5.3 mostra a interação entre os elementos de suporte do *framework* no processo de carregamento de um contrato a ser gerenciado.

No primeiro passo, `CrrioGUI` solicita à classe principal (`ContractManager`) o carregamento do contrato selecionado pelo usuário (`1:loadContract`). `ContractManager` inicia então o processo de carregamento do contrato, começando pela interpretação sintática da sua descrição (`1.1:parse`). É nessa interpretação que um objeto da classe `QoSContract` é criado para armazenar os elementos contidos na descrição do contrato. Depois disso, a classe principal inicia cada `Contractor` remoto responsável pela coordenação do monitoramento dos recursos requeridos pela aplicação (`1.2:startContractors` e `1.2.1:startContractor`). Nessa iniciação são passadas para cada `Contractor` as seguintes informações: o nome do *host* onde está sendo executada a classe principal (para que o `Contractor` possa se comunicar via RMI com ela); as restrições associadas aos serviços do contrato que serão utilizadas na validação dos perfis (somente as restrições envolvendo propriedades de entrada, pois são as que definem as condições de validação dos perfis); e as regras de transição de serviços para que o `Contractor` possa verificar a validação de perfis associados a possíveis serviços de maior prioridade que poderiam ser impostos.

No próximo passo, `1.3:nextValidService`, as tentativas de estabelecimento de serviço serão iniciadas, de acordo com a ordem de prioridade definida na cláusula de negociação de serviços descrita no contrato. Para cada tentativa, ou seja, para saber se um serviço pode ser estabelecido, `ContractManager` solicita a cada `Contractor` responsável pelo gerenciamento da monitoração das propriedades envolvidas nas restrições associadas ao serviço candidato, que verifique se os valores dessas propriedades satisfazem as restrições associadas a esse serviço (`1.3.1:checkRestrictions`). Para efetuar essa verificação, um `Contractor` requisita a cada `ResourceAgent` que efetue a leitura dos valores correntes das propriedades de recurso envolvidas nas restrições associadas ao serviço (`1.3.1.1:readResourceState`). O `Contractor` notifica então o resultado dessa

verificação à classe `ContractManager`, uma resposta positiva (`ok`) no diagrama da Figura 5.3.

Depois de ser notificado por cada `Contractor` que as restrições para um serviço podem ser atendidas, `ContractManager` inicia o processo de estabelecimento do serviço – `1.4:startService(nextService)`. A primeira ação desse processo é requisitar aos mesmos *Contractors* que iniciem o processo de monitoração das propriedades dos recursos a serem utilizados, com intuito de verificar a observância das restrições associadas ao serviço a ser estabelecido (`1.4.1:enforceRestrictions`). Para isso, cada `Contractor` solicita aos `ResourceAgents` por ele gerenciados, o inicio do monitoramento dessas propriedades (`1.4.1.1:startMonitoring`). Por fim, `ContractManager` repassa para o `Configurator` as configurações arquiteturais descritas no contrato para o serviço a ser estabelecido, juntamente com os perfis de qualidade a ele associados. As informações contidas nesses perfis podem ser utilizadas na orientação do estabelecimento do serviço, como por exemplo, uma restrição em um perfil que indique a qualidade desejada para um vídeo a ser transmitido (`1.4.2:effectAdaptation`).

É importante ressaltar que a comunicação dos *Contractors* com os *Resource Agents* a eles associados é feita de forma local, e que a comunicação do *Contract Manager* com os *Contractors* é feita via RMI. Embora os *ResourceAgents* pudessem ser implementados como componentes independentes que proveriam, via rede, informações sobre os valores das propriedades monitoradas, a comunicação local justifica-se pelo fato do processo de verificação de restrições efetuado nos *Contractors* requerer a busca periódica pelos valores monitorados. Portanto, a comunicação remota neste caso não seria adequada, já que o processo de verificação de restrições exigiria um grande número de comunicações remotas, sobrecarregando a rede e aumentando o tempo de execução desse processo.

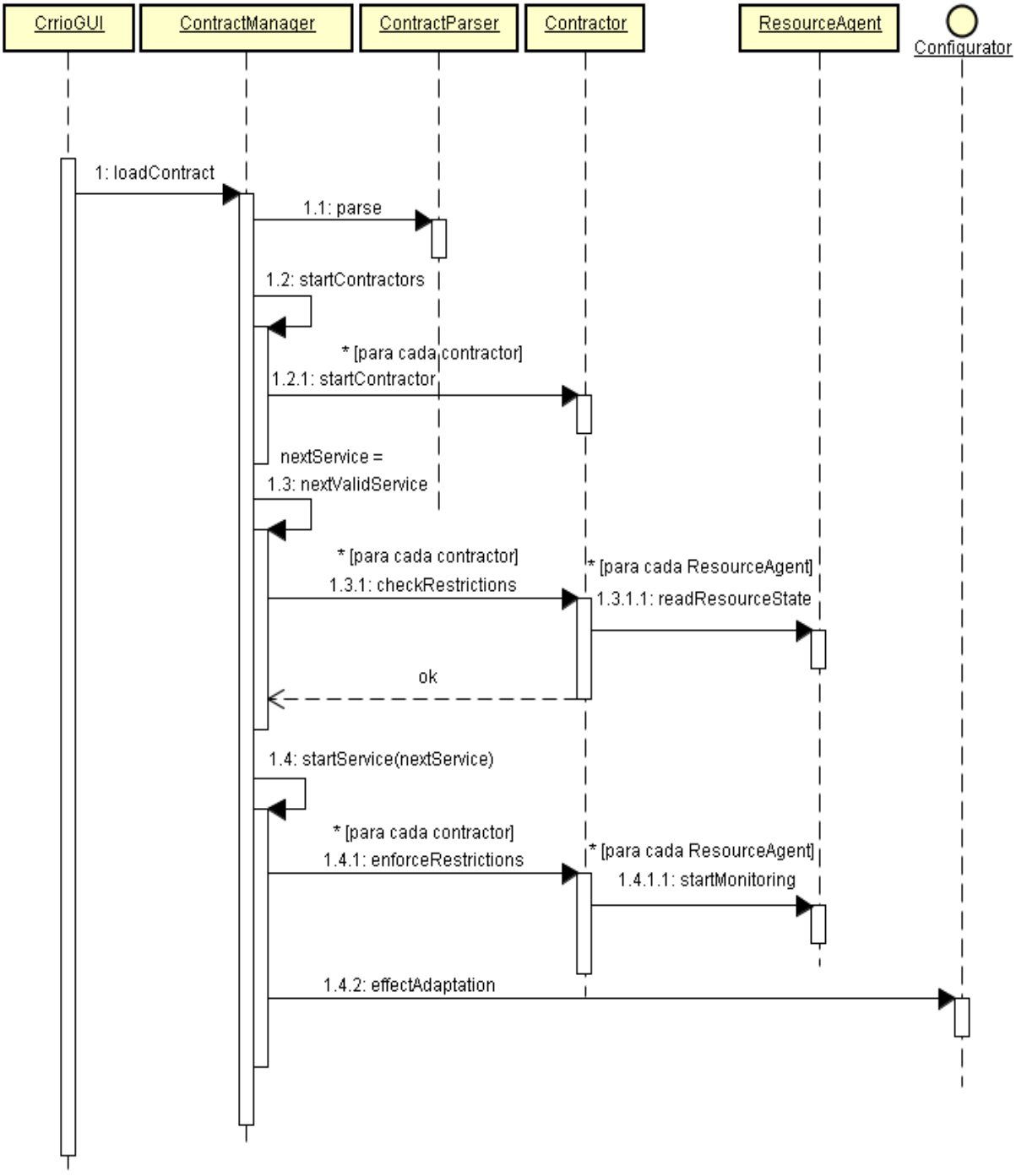

Figura 5.3 – Diagrama de interação do processo de carregamento de contrato no CR-RIO.

Com o serviço estabelecido, as propriedades de recurso envolvidas em suas restrições continuam a ser verificadas. Caso em um determinado momento um `Contractor` detecte que alguma restrição não possa mais ser satisfeita, ele notifica essa violação ao Gerente de Contrato (`ContractManager`). O diagrama da Figura 5.4 ilustra o comportamento dos elementos do modelo de suporte do CR-RIO nessa situação.

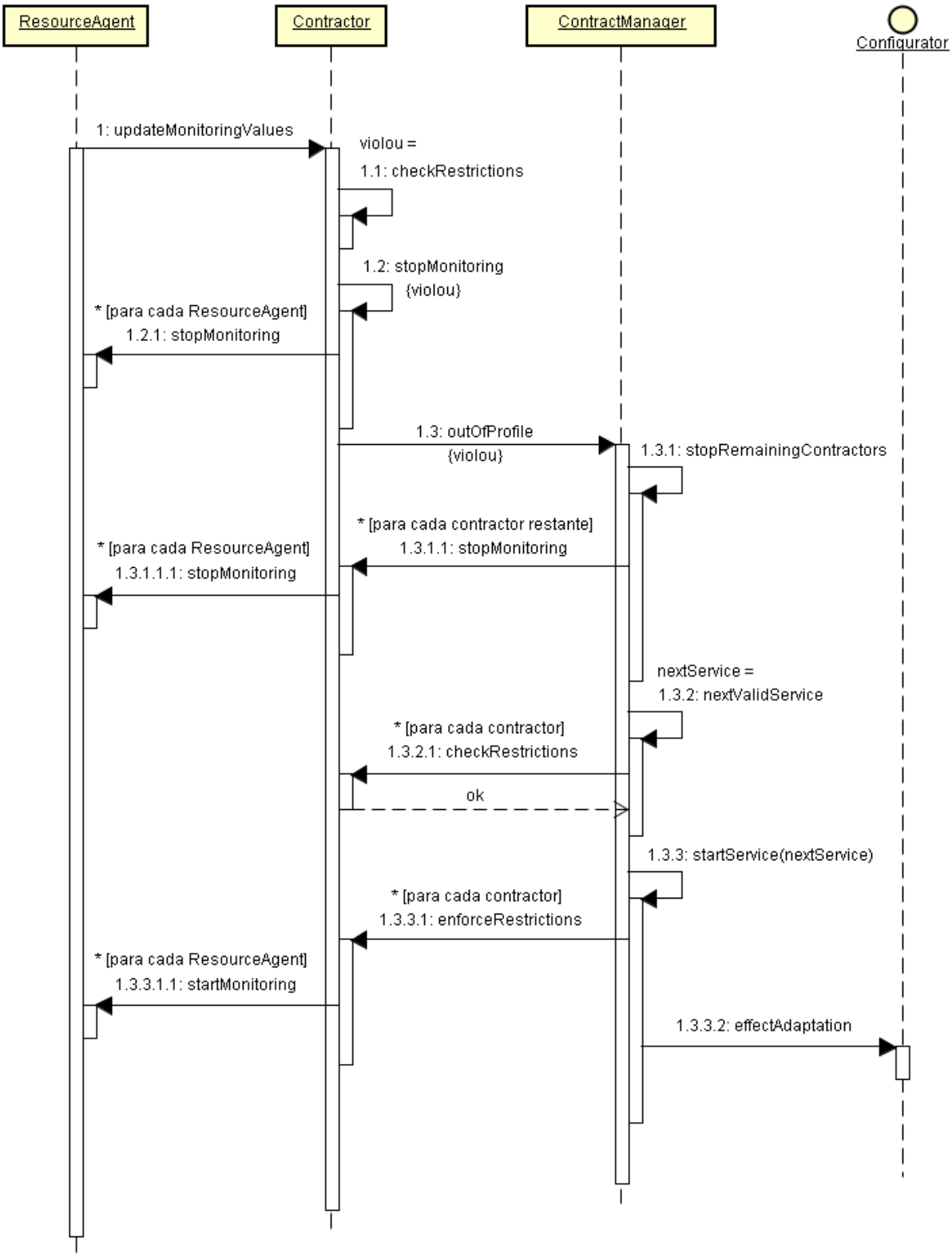

Figura 5.4 – Diagrama de interação do processo de violação de contrato no CR-RIO.

Quando um `ResourceAgent` detecta uma variação nos valores monitorados, ele envia ao `Contractor` que o gerencia os novos valores das propriedades de recurso por ele

monitoradas (`1:updateMonitoringValues`, Figura 5.4). O `Contractor` verifica então se esses novos valores satisfazem as restrições, referentes às propriedades cujos valores foram alterados, associadas ao serviço estabelecido (`1.1:checkRestrictions`). Se ele comprovar a violação de alguma dessas restrições, primeiramente, ele interrompe o processo de monitoramento nos agentes de recurso por ele gerenciados (`1.2:stopMonitoring` e `1.2.1:stopMonitoring`). Depois disso, o `Contractor` notifica o Gerente de Contrato que uma violação de perfil ocorreu (`1.3:outOfProfile`). Ao receber essa notificação, o Gerente requisita a cada `Contractor` restante, caso exista algum, a interrupção do processo de monitoração por ele gerenciado, já que o serviço corrente será encerrado (`1.3.1:stopRemainingContractors` e `1.3.1.1:stopMonitoring`). Então, cada *Contractor* restante interrompe o monitoramento efetuado pelos agentes a ele associados (`1.3.1.1.1:stopMonitoring`). A partir daí, o Gerente tentará estabelecer um possível próximo serviço, seguindo os mesmos passos iniciados em `1.3:nextValidService` da Figura 5.3.

É importante ressalvar que quando o processo de monitoração é interrompido, nem o *Contractor* nem os *ResourceAgents* são descarregados, eles apenas param de efetuar a verificação das restrições associadas ao serviço e as leituras dos valores das propriedades dos recursos. Desta forma, eles ficam prontos e aguardando solicitações de verificação de restrições, quando o Gerente verifica se um serviço pode ser estabelecido, e solicitações de inicio do processo de monitoramento, quando um serviço é estabelecido e suas restrições precisam ser observadas para garantir a qualidade desejada (*v.* Figura 5.3).

5.1.4 Especialização do Modelo

O modelo de classes do CR-RIO (Figura 5.2) deve ser especializado em alguns pontos para cada tipo de recurso e para cada contrato específico a uma aplicação. Esses pontos são os *hotspots* – pontos de ajustes específicos – do *framework* e estão localizados nos seguintes elementos da infra-estrutura de suporte: *Resource Agent*, *Contractor* e *Configurator*.

No elemento *Resource Agent* os ajustes consistem em especializar as classes `ResourceAgent` e `ResourceState` para cada categoria de recurso. Na prática, para cada categoria de QoS definida, especializações dessas duas classes devem ser criadas contemplando a particularidade da categoria de QoS a que elas se referem. Como essas classes são específicas a uma determinada categoria de QoS, elas podem ser reutilizadas no

gerenciamento de diferentes contratos que tenham alguma restrição relacionada com a categoria a que elas dizem respeito. O diagrama de classes da Figura 5.5 ilustra a especialização para as categorias de QoS `Processing` e `Transport` utilizadas no exemplo da aplicação de vídeo sob demanda (Código 4.2). Com o intuito de facilitar o entendimento do diagrama, somente os métodos e atributos relevantes para o contexto da especialização são mostrados.

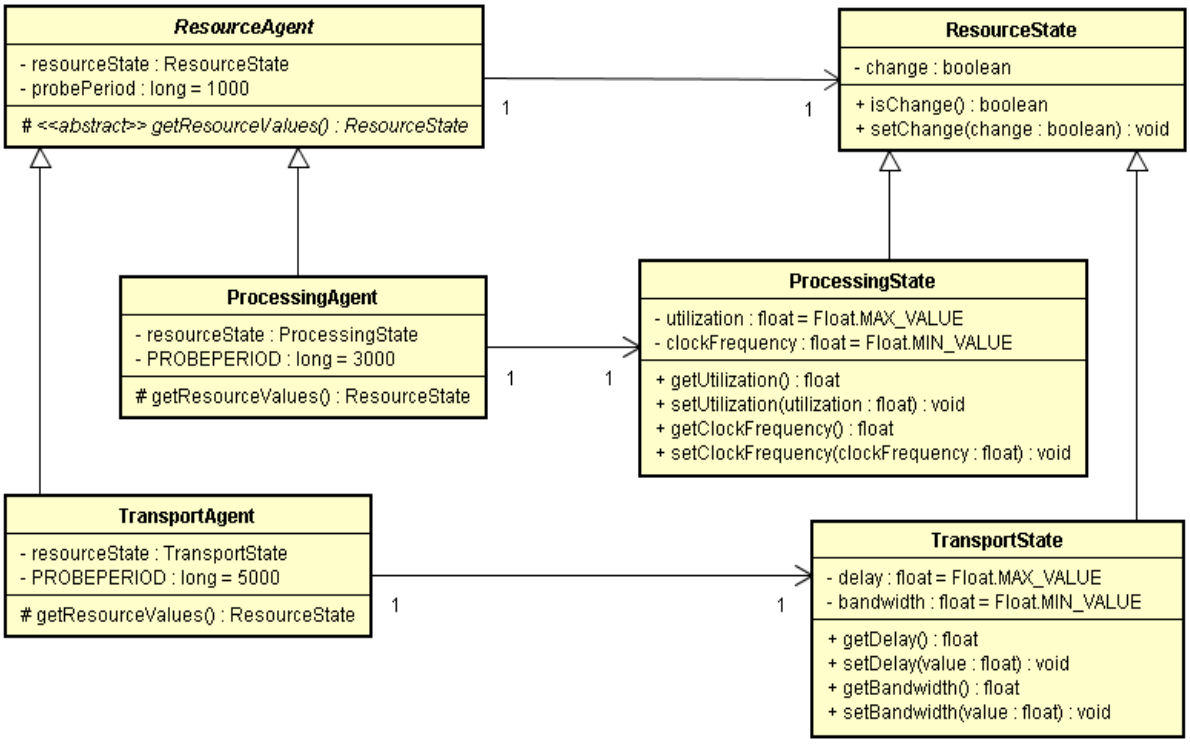

Figura 5.5 – Diagrama de classes exemplificando a especialização do CR-RIO para as categorias de QoS *Transport* e *Processing*.

Na Figura 5.5 as classes `ProcessingAgent` e `TransportAgent` herdam `ResourceAgent` e implementam o método abstrato `getResourceValues` nela definido. A implementação desse método é responsável por efetuar a leitura das propriedades do recurso ao qual a classe especializada se refere. A classe filha também pode redefinir o período em que a monitoração do recurso ocorrerá, por exemplo, em `ProcessingAgent` esse período é redefinido para 3 segundos (`PROBEPERIOD:long = 3000`). As classes `ProcessingState` e `TransportState` herdam `ResourceState`, definindo, respectivamente, os atributos `utilization` e `clockFrequency` para o recurso processamento e `delay` e `bandwitdth` para o recurso transporte. Esses atributos serão valorados no processo de monitoramento e utilizados pelos *Contractors* para validação das restrições associadas a um determinado

serviço. A superclasse `ResourceState` apenas define o atributo `change`, que indica se os valores dos atributos definidos em uma classe especializada foram alterados. Essa informação é utilizada pelo `ResourceAgent`, que quando detecta que os valores desses atributos foram alterados, dispara a atividade de verificação de restrições implementada no `Contractor`. Através do uso do atributo `change`, na criação de uma classe especializada de `ResourceAgent`, pode ser definida uma função de variação que identifique quando uma mudança significativa ocorre. Isso evita que os *Contractors* sejam notificados sobre alterações insignificantes nos valores monitorados.

Já os elementos *Contractor* e *Configurator* devem ser especializados para cada contrato específico a uma aplicação. Isso é feito pela criação de uma classe especializada de `Contractor` e pela criação de outra que implementa a interface `Configurator`. A Figura 5.6 ilustra as classes criadas através desse modelo para o exemplo da aplicação de vídeo sob demanda (Vod) apresentado na seção 4.1.

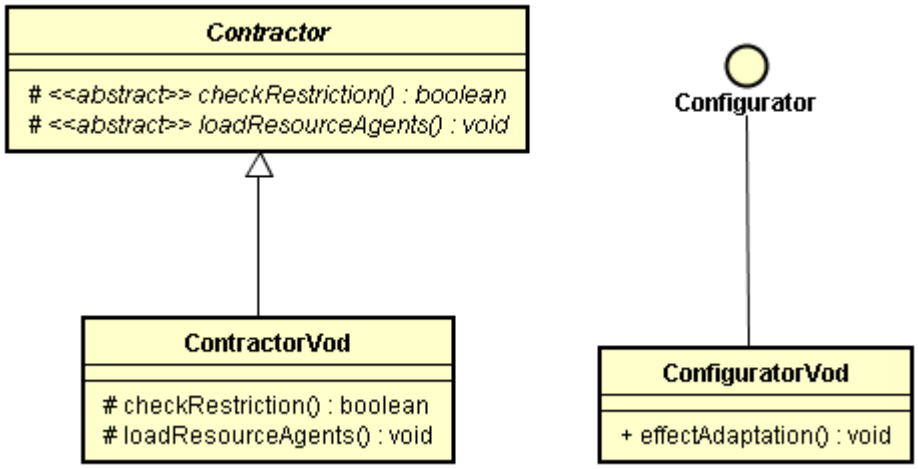

Figura 5.6 – Diagrama de classes da especialização do *Contractor* e do *Configurator* para o exemplo de aplicação vídeo sob demanda.

Na especialização ilustrada na Figura 5.6, a classe `ContractorVod` herda `Contractor`, obrigando-a assim a implementar os métodos abstratos `checkRestriction` e `loadResourceAgents` definidos na superclasse. No primeiro método é implementada a funcionalidade que se encarrega de verificar se os valores monitorados satisfazem as restrições de qualidade associadas aos serviços declarados no contrato da aplicação Vod (Código 4.3). No segundo método é implementado o carregamento das classes especializadas de `ResourceAgent`, que são responsáveis pelo monitoramento das propriedades de recurso

utilizadas nas restrições deste contrato. A especialização do elemento *Configurator* é feita pela criação da classe `ConfiguratorVod`, que implementa a interface `Configurator`. `ConfiguratorVod` implementa o método `effectAdaptation` definido nessa interface. A função deste método é traduzir as configurações arquiteturais descritas em um serviço para ações do nível de sistema que garantam o estabelecimento do serviço com a qualidade desejada. Essas ações do nível de sistema são específicas a cada serviço, por isso é necessário criar uma classe *Configurator* para cada contrato de QoS. Ou seja, a classe criada será responsável por garantir o estabelecimento dos serviços descritos em um contrato.

5.2 Funcionamento

Como foi mostrado na seção anterior, para que uma aplicação possa ser gerenciada pelo *framework* CR-RIO, de acordo com os requisitos de qualidade por ela desejados e descritos na forma de um contrato de QoS, o modelo de classes do *framework* deve ser especializado para contemplar as particularidades inerentes a cada aplicação. Além disso, é preciso implantar os componentes de suporte a gerenciamento e monitoração no ambiente da aplicação. Os seguintes passos envolvem o estabelecimento de um contrato para uma aplicação no CR-RIO:

(1) O arquivo contendo a descrição arquitetural da aplicação, os arquivos descrevendo as categorias de QoS relacionadas com os recursos utilizados pela aplicação e o arquivo contendo o contrato de QoS desejado devem ser carregados em repositórios específicos do *framework*. Isso é necessário para que o CR-RIO possa reconhecer esses arquivos no momento em que ele for carregado.

(2) O modelo de classes do CR-RIO deve ser particularizado para a aplicação através da implementação das classes especializadas de `ResourceAgent`, `ResourceState`, `Contractor` e `Configurator`. As classes especializadas de `Contractor` e `Configurator` devem seguir um padrão de nomenclatura para os seus nomes. Esse padrão consiste em concatenar os nomes “Contractor” e “Configurator” ao nome do contrato descrito para a aplicação, por exemplo, `ContractorVod` e `ConfiguratorVod` para um contrato de nome `Vod`. Esse padrão permite ao *framework*, juntamente com o mecanismo de reflexão suportado pela linguagem Java, reconhecer novas classes especializadas para aplicações específicas a partir do nome do contrato. Seguindo esse padrão, para que o CR-RIO reconheça essas classes basta carregá-las em locais apropriados da estrutura de pacotes do *framework*.

(3) Os componentes *Contractors* devem ser carregados nos *hosts* onde eles gerenciarão o monitoramento das propriedades de recurso utilizadas nas restrições de qualidade associadas aos serviços do contrato. Antes disso, para que cada um desses componentes possa ser localizado e acessado remotamente pelo componente *Contract Manager*, através do mecanismo RMI, é preciso carregar o serviço de nome utilizado por esse mecanismo, o *Java Remote Object Registry* (rmiregistry), em cada um desses *hosts*. Depois do serviço de nome carregado, os *Contractors* podem ser iniciados. Esse processo é feito pela execução da classe `RemoteContractorImpl`, que carregará a especialização da classe `Contractor` para o contrato da aplicação e a registrará no serviço de nome para seu uso remoto. Para que o *framework* possa saber qual `Contractor` especializado deve ser carregado e quais recursos deverão ser monitorados, as seguintes informações são passadas como parâmetros na iniciação da classe `RemoteContractorImpl`: o nome do contrato que será gerenciado e os nomes das categorias cujas propriedades serão monitoradas.

(4) O último passo é o carregamento da interface `CrrioGUI`, que ao entrar em execução carrega o módulo *Contract Manager* (classe `ContractManager`), juntamente com as categorias e contratos de QoS localizados nos repositórios do *framework*. No *host* onde ela for carregada também é preciso que o serviço de nome utilizado pelo RMI esteja em execução, já que os *Contractors* se comunicam com o *Contract Manager* via RMI. A partir daí, um usuário do CR-RIO poderá selecionar o contrato para uma aplicação na interface e ordenar o início do seu gerenciamento.

5.3 Exemplo de Utilização

Para exemplificar e validar a implementação do *framework* CR-RIO, uma aplicação baseada no exemplo de vídeo sob demanda apresentado na seção 4.1 foi desenvolvida. A razão pela escolha dessa aplicação foi o fato de sua implementação ser rápida, adequando-se ao intuito de teste do *framework*. Porém, aplicações mais complexas estão sendo desenvolvidas – uma aplicação cliente-servidor com requisitos de tempo real [Freitas et al. 2005] e uma aplicação de videoconferência [Loques et al. 2005] –, permitindo que o CR-RIO seja testado no gerenciamento de aplicações em cenários mais complexos.

A aplicação de vídeo sob demanda implementada consiste de um servidor que transmite vídeos a clientes remotos interessados em reproduzi-los em suas máquinas. Um vídeo pode estar armazenado no servidor em um arquivo multimídia ou pode ser capturado de

uma câmera conectada ao servidor. Os clientes solicitam ao servidor a transmissão de um vídeo, informando a qualidade desejada para ele, no caso deste exemplo, MJPEG ou H.263.

Na implementação do servidor foi utilizado o *Java Media Framework (JMF)*, uma API concebida para a captura, decodificação, codificação, transmissão e recepção de dados multimídia via *streaming* em aplicações e *applets* Java [JMF 2005]. O uso dessa API permitiu ao servidor capturar e transmitir um vídeo aos clientes remotos interessados, com a qualidade por eles desejada. A transmissão do vídeo foi feita utilizando o protocolo RTP (*Real-Time Transport Protocol*) suportado pela JMF. Esse protocolo fornece serviços de transporte fim-a-fim para transmissão de dados pela rede em tempo-real.

Para reproduzir o vídeo recebido nos clientes foi utilizada a ferramenta VIC, uma aplicação multimídia de tempo-real, baseada no protocolo RTP, para videoconferência sobre a Internet. Ela foi escolhida pela sua simplicidade de uso e por se adequar ao propósito da aplicação – reproduzir vídeos transmitidos pelo servidor através do protocolo RTP e suportar alteração do formato de um vídeo sem parar sua reprodução.

5.3.1 Implementação do Modelo definido no CR-RIO

De forma a permitir que aplicação possa ser gerenciada com base em um contrato de QoS descrito para ela, é preciso implementar os passos do modelo do *framework* CR-RIO: descrição da configuração arquitetural e do contrato de QoS para a aplicação através da ADL CBabel e a implementação dos *hotspots* do *framework* conforme descrito na seção 5.1.4.

A descrição arquitetural, as categorias e o contrato de QoS para essa aplicação são os mesmos apresentados no exemplo de aplicação de vídeo sob demanda (seção 4.1). Para que essas descrições possam ser interpretadas pelo CR-RIO, a descrição arquitetural da aplicação deve estar contida em um arquivo, o contrato de QoS em outro e, juntamente com os arquivos contendo a descrição das categorias de processamento e transporte, eles devem ser carregados nos repositórios apropriados do *framework*.

Na implementação dos *hotspots* do *framework*, em relação aos *Resource Agents*, foram criadas as classes `ProcessingAgent`, `ProcessingState`, `TransportAgent` e `TransportState` responsáveis pela leitura e armazenamento dos valores das propriedades das categorias `Processing` e `Transport` (Código 4.2) utilizadas nas restrições descritas no contrato para essa aplicação (Código 4.3). Com intuito de facilitar os testes do *framework* através do gerenciamento desse contrato, a leitura dessas propriedades foi implementada de

forma simulada. Isso foi feito através da criação de interfaces gráficas que simulam os recursos a serem monitorados. Desta forma, o valor de cada uma das propriedades de interesse pode ter o seu valor alterado através da interface, o que facilitou a criação das diversas circunstâncias compreendidas no gerenciamento de um contrato de QoS para uma aplicação.

No que diz respeito à especialização do *Contractor*, foi implementada a classe `ContractorVod`, já que o nome do contrato é `Vod` (linha 16 do Código 4.3), responsável por implementar a verificação das restrições para este contrato e por carregar as classes `ProcessingAgent` e `TransportAgent` responsáveis pela leitura das propriedades dos recursos de processamento e transporte, respectivamente. Para a implementação do elemento *Configurator* foi criada a classe `ConfiguratorVod`, que é encarregada de estabelecer os serviços descritos no contrato para essa aplicação. Para isso, quando o *Contract Manager* passa para o `ConfiguratorVod` as configurações referentes a um serviço a ser estabelecido e seus perfis qualidade associados, `ConfiguratorVod` extrai as informações necessárias para o estabelecimento do serviço e requisita ao servidor, via *socket*, o inicio da transmissão de um vídeo ou a troca do formato de um já em transmissão. Essas informações extraídas das configurações do serviço e de seus perfis associados indicam para o servidor o formato desejado para o vídeo e o cliente remoto que o receberá. Por exemplo, o serviço `sMJPEG_video`, descrito nas linhas 02 a 05 do Código 4.3, indica que o formato do vídeo será MJPEG e o nome do *host* do cliente que receberá esse vídeo (linha 03 desse mesmo código).

5.3.2 Implantação da Aplicação e do framework CR-RIO

Em um primeiro cenário de teste para essa aplicação exemplo, a implementação do servidor e um cliente VIC foram carregados em *hosts* distintos. Neste ponto o servidor está pronto para receber a requisição de transmissão do vídeo capturado de uma câmera conectada à máquina onde ele está em execução, e o cliente pronto para o recebimento e reprodução desse vídeo.

O próximo passo foi a implantação do componente especializado de *Contractor* (`ContractorVod`) no *host* do cliente, encarregado de gerenciar o monitoramento das propriedades do recurso de processamento de sua máquina (freqüência de operação e utilização do processador), e no *host* do servidor, responsável pelo monitoramento das

propriedades do recurso de transporte (canal de comunicação entre o cliente e o servidor). A implantação do *Contractor* especializado foi feita pelo carregamento da classe `RemoteContractorImpl` na linha de comando. Para o *Contractor* no *host* servidor, por exemplo, o comando para o carregamento foi: `java RemoteContractorImpl Vod Transport`. O parâmetro `Vod` indica que o contrato a ser gerenciado será o de nome “Vod” e, portanto, que o nome da classe que especializa o *Contractor* para o contrato em questão é `ContractorVod`. O parâmetro `Transport` é nome da categoria de QoS que terá suas propriedades monitoradas, ou seja, o `ContractorVod` carregado no servidor gerenciará o monitoramento das propriedades do recurso de transporte.

Com os *Contractors* especializados implantados nos *hosts* do cliente e do servidor, a interface gráfica do CR-RIO (`CrrioGUI`) foi carregada em um outro *host* da rede. Ao ser iniciado, o *framework* carrega as categorias `Processing` e `Transport` contidas em seu repositório e permite que o gerenciamento do contrato `Vod` seja solicitado pelo usuário.

Quando o gerenciamento do contrato é iniciado, o *framework* solicita aos dois *Contractors* carregados a verificação das restrições associadas ao serviço de maior prioridade, no caso, aquele em que o vídeo será transmitido com melhor qualidade (MJPEG). Os *Contractors* requisitam os valores das propriedades envolvidas aos *Resource Agents* por ele gerenciados (`ProcessingAgent` e `TransportAgent`). Esses valores são lidos nas interfaces gráficas que simulam os recursos envolvidos, e, caso satisfaçam as restrições, o serviço será estabelecido e o vídeo capturado pela câmera conectada ao servidor será transmitido ao cliente com a qualidade MJPEG. Se os valores das propriedades de recurso forem alterados na interface gráfica, o *Contractor* reavaliará as restrições e, caso elas sejam violadas, o serviço com menor qualidade de vídeo (H.263) tentará ser estabelecido. O estabelecimento deste serviço trocará o formato do vídeo de MJPEG para H.263, o que é efetivado pela inclusão de um conector, descrito na configuração arquitetural do serviço, responsável por essa transcodificação de formatos. Com o serviço de menor qualidade ativo, se os valores das propriedades forem alterados de forma a satisfazer as restrições do serviço de melhor qualidade, este será estabelecido e o vídeo voltará a ser transmitido no formato MJPEG.

O gerenciamento do contrato `Vod` para esta aplicação exemplo permitiu validar a implementação do *framework* e mensurar o desempenho de suas operações no processo de gerenciamento de contrato. No cenário deste exemplo foram utilizadas três maquinas com

processadores Pentium 4 de 2.80 GHz e com 512 MB de memória RAM. Essas máquinas foram conectadas através de um *switch* 10/100 Mbps, onde cada uma representa um *host* na rede. Nos testes de desempenho executados o tempo médio gasto pelo *framework* entre o recebimento de uma notificação de violação de perfil (*out-of-profile*) e o estabelecimento de um próximo serviço foi de 2,3 ms. Conforme ilustrado na Figura 5.4, esse processo é composto pelos seguintes passos: encerrar o monitoramento dos *Contractors* restantes (`stopRemainingContractors`), onde foi gasto o tempo médio de 0,5 ms; a busca por um próximo serviço válido (`nextValidService`), tempo médio de 0,9 ms; estabelecimento do próximo serviço válido (`startService`), tempo médio de 0,95 ms; e a efetivação das configurações do serviço para o exemplo em questão (`effectAdaptation`), o envio (via *socket*) de requisições de transmissão de vídeo ao servidor, tempo médio de 0,050 ms.

O *framework* CR-RIO foi testado também em um cenário onde foi preciso gerenciar três contratos de forma simultânea. Esse cenário mais complexo foi composto por uma máquina servidora responsável pelo envio de vídeos capturados de três arquivos diferentes a três clientes remotos. Foi definido um contrato para cada cliente, de forma que cada um pudesse expressar a qualidade desejada para o vídeo transmitido de acordo com os limites de suas máquinas. Assim, o CR-RIO pode gerenciar de forma simultânea e independente cada um desses contratos, permitindo que a qualidade do vídeo transmitido a cada um dos clientes pudesse ser definida, levando em consideração as opções descritas em seus contratos e a disponibilidade do recurso de processamento de suas máquinas.

5.4 Conclusão do Capítulo

Neste capítulo foram expostos alguns aspectos de implementação do *framework* CR-RIO, apresentando o modelo de classes da sua implementação e como os elementos deste modelo interagem no gerenciamento de contratos de QoS para aplicações com requisitos não-funcionais. Foi descrita uma forma padrão para especialização do modelo a cada contrato específico a uma aplicação e às categorias de QoS que descrevem as propriedades dos recursos utilizados pela aplicação.

Foi apresentada também a aplicação de vídeo sob demanda desenvolvida para testar e validar a implementação do *framework*. Através dessa aplicação foram exemplificados os passos necessários para implementar os *hotspots* do CR-RIO e como implantar os componentes da sua infra-estrutura de suporte no ambiente distribuído da aplicação. Os testes

executados no gerenciamento do contrato para essa aplicação pelo CR-RIO mostraram a sua aplicabilidade no gerenciamento de aplicações com requisitos de QoS. Além disso, foi possível mensurar o tempo gasto pelas operações do *framework*, mostrando que o processo de gerenciamento efetuado pelo CR-RIO possui um desempenho aceitável por muitas aplicações que necessitam desse tipo de gerenciamento.

Capítulo 6

Trabalhos Correlatos

Este capítulo descreve alguns trabalhos correlacionados com o *framework* CR-RIO. Dentre as propostas consideradas, QuO, *Rainbow* e HQML são as que possuem mais pontos em comum com o CR-RIO, e suas descrições e comparações constituem a maior parte deste capítulo. Além desses, alguns outros trabalhos são descritos de forma mais sucinta.

6.1 QuO

O *Quality Objects* (QuO) [Loyall et al. 1998, Pal et al. 2000, Schantz et al. 2002] é um *framework* que permite incluir QoS em aplicações distribuídas para a plataforma CORBA. Ele suporta a especificação de contratos de QoS entre clientes e provedores de serviços, inclui o suporte para monitoramento em tempo de execução dos contratos e também provê mecanismos para a adaptação da aplicação.

6.1.1 Visão Geral

Em uma aplicação CORBA padrão, um cliente faz uma chamada de método a um objeto remoto através de sua interface funcional. A chamada é processada por um ORB (*Object Request Broker*) na máquina cliente, entregue ao ORB localizado na máquina do objeto remoto, e então processada por esse objeto. Como pode ser visto na Figura 6.1, o *framework* QuO adiciona alguns passos a esse processo. Esses passos são executados pelos seguintes componentes implementados pelo desenvolvedor de acordo com a proposta QuO:

- ***Contract***: especifica um contrato definindo o nível de serviço desejado pelo cliente, o nível de serviço que um objeto remoto espera prover, regiões de operação indicando possíveis estados de QoS e ações que são tomadas quando mudanças nos estados de QoS ocorrem;
- ***Delegate***: atua como um *proxy* local para o objeto remoto. O *delegate* provê uma interface similar à do objeto remoto, porém adiciona adaptações de comportamento levando em consideração o atual estado de QoS do sistema. Assim, é possível escolher alternativas diferentes de acordo com o estado de QoS determinado pelo *Contract*;
- ***System condition objects (SysCond)*:** provê interfaces para recursos, gerenciadores de mecanismos ou propriedades, objetos e ORBs do sistema que precisam ser medidos e controlados pelos contratos QuO.

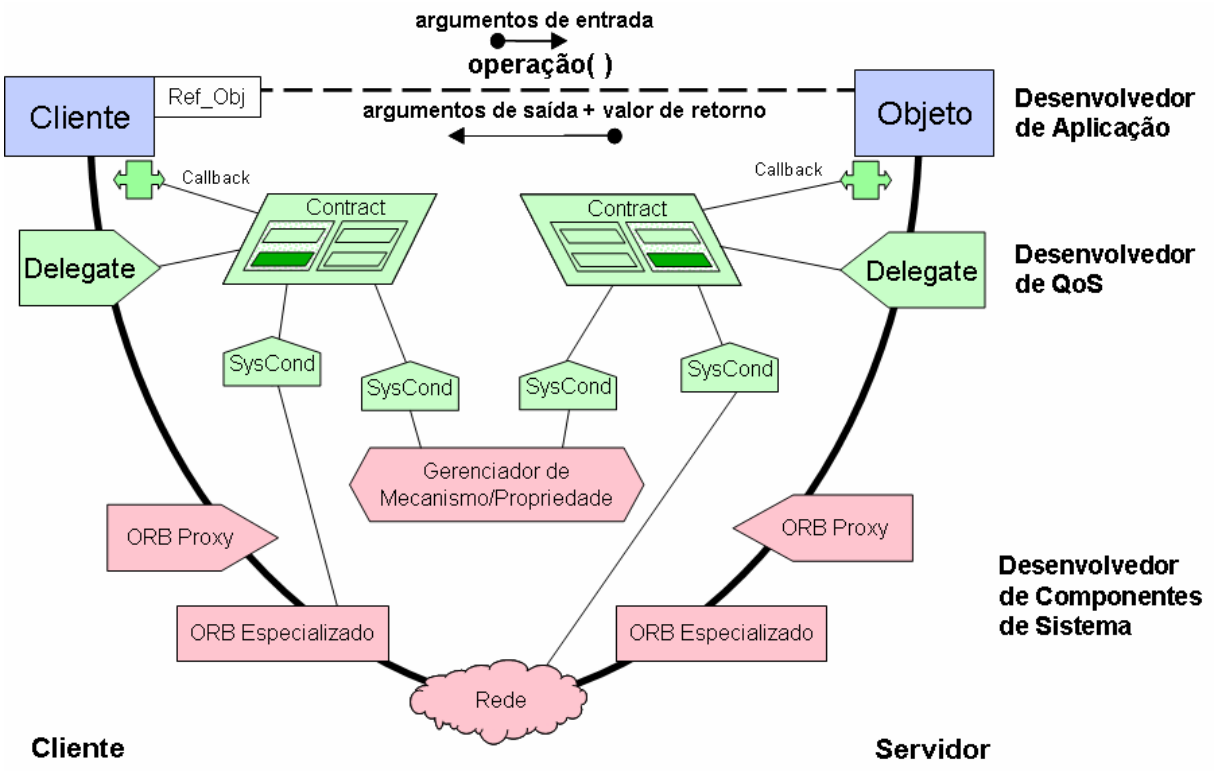

Figura 6.1 – Visão geral do *framework* QuO.

6.1.2 Funcionamento

A Figura 6.2 ilustra os passos do processo de gerenciamento de contrato durante uma chamada a um método remoto em uma aplicação QuO.

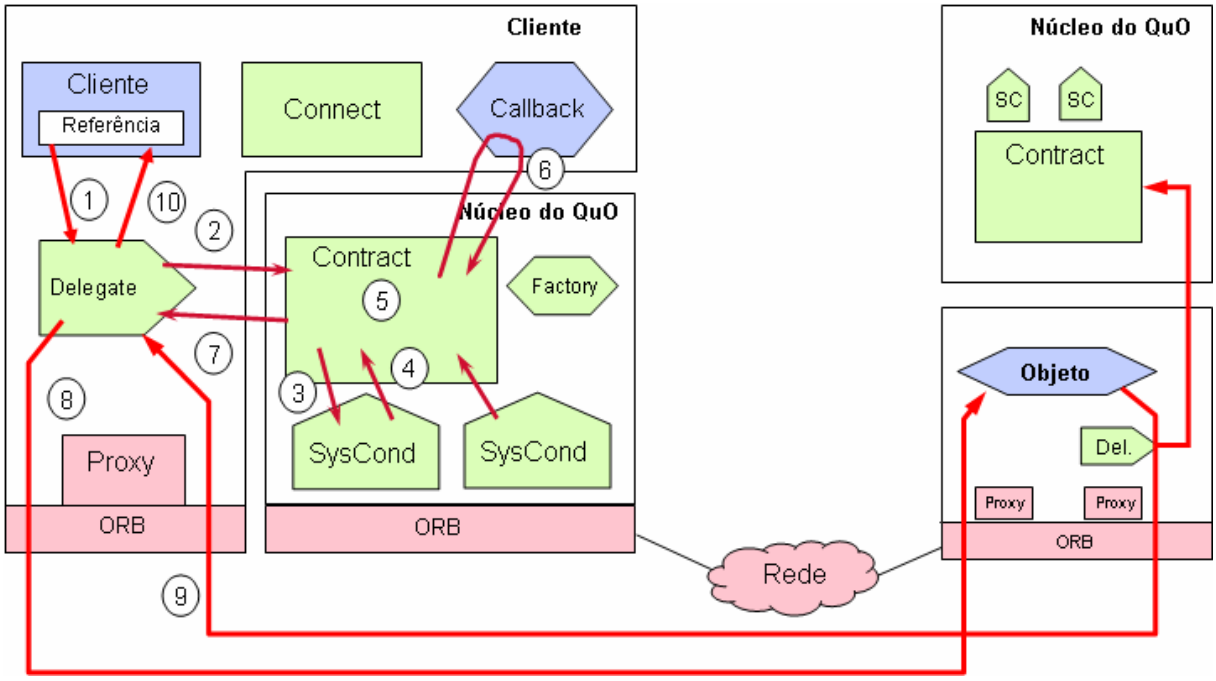

Figura 6.2 – Chamada a um método remoto em uma aplicação QuO.

Em uma aplicação QuO, Quando um cliente faz uma chamada a um método remoto, ela é interceptada pelo *Delegate* (passo 1 na Figura 6.2). O *Delegate* dispara a avaliação do contrato (passo 2), onde são coletados os valores das condições do sistema medidos pelos objetos *SysCond* (passos 3 e 4). O contrato é composto por um conjunto de regiões aninhadas que descrevem os possíveis estados de QoS do sistema. Cada região é definida por um predicado composto por valores dos objetos de condição do sistema (*SysCond*). O *Contract* avalia esses predicados para definir quais regiões estão ativas (passo 5). Se uma transição de estado ocorre (quando ocorre mudança nos valores dos predicados das regiões, de forma que uma região que estava inativa torna-se ativa), *Callbacks* – chamadas a métodos da aplicação cliente – ou chamadas a métodos dos objetos *SysCond* podem ser disparadas (passo 6). O *Contract* passa a lista das regiões ativas para o *Delegate* (passo 7), que de acordo com essa informação escolhe uma alternativa para processar a chamada ao método remoto. Ele pode, por exemplo, bloquear a chamada ao método e efetuar alguma adaptação no sistema, caso o nível de QoS tenha degradado, ou simplesmente passá-la ao objeto remoto (passo 8), caso

contrário. O objeto remoto recebe e processa a chamada ao seu método e retorna o valor resultante (passo 9). O *Contract* é então novamente avaliado, da mesma forma descrita anteriormente, e, de acordo com a informação de quais regiões estão ativas, o *Delegate* escolhe um comportamento a ser tomado, por exemplo, podendo repassar o valor de retorno para o cliente (passo 10).

6.1.3 Componentes do *Framework* QuO

No desenvolvimento de uma aplicação QuO o desenvolvedor é responsável pela implementação dos contratos QuO, dos objetos de condição do sistema, dos mecanismos de *Callback* e do comportamento do objeto *Delegate*. O *framework* QuO é composto pelos seguintes componentes, que dão suporte a esse desenvolvimento:

- **Linguagens de descrição de qualidade (QDL – *Quality Description Languages*)**: um conjunto de linguagens usadas para descrever os aspectos de QoS das aplicações QuO, como contratos de QoS (especificado pela linguagem de descrição de contratos CDL – *Contract Description Language*) e o comportamento adaptativo dos objetos e *Delegates* (especificados pela linguagem de descrição de estrutura SDL – *Structure Description Language*);
- **Núcleo de tempo de execução QuO**: que é responsável por coordenar a avaliação dos contratos e monitorar os objetos de condição de sistema;
- **Geradores de código**: responsáveis por reunir as descrições das linguagens de aspecto de QDL, o código do núcleo QuO e o do cliente para produzir um único programa.

6.1.4 Exemplo Controle de Replicação de Objeto Remoto

O Código 6.1 mostra um exemplo de descrição de contrato QuO, através da linguagem CDL, que especifica e controla a replicação de objetos remotos em uma aplicação QuO [Loyall et al. 1998]. O cliente possui dois modos de operação, um onde é desejado um alto grau de disponibilidade dos objetos remotos e outro onde ele deseja usar poucos recursos, não precisando assim, de um alto grau de disponibilidade.

São passados como parâmetros para o contrato três objetos de condição de sistema (`ClientExpectedReplicas`, `MeasuredNumberReplicas` e `ReplMgr`) e um *Callback*

(`ClientCallback`). `ClientExpectedReplicas` contém o número de réplicas de objeto remoto desejado pelo cliente, `MeasuredNumberReplicas` observa o atual número de réplicas no sistema, e `ReplMgr` é o gerenciador de política de replicação. O *Callback* `ClientCallback` permite notificar ao cliente da aplicação o atual estado de disponibilidade dos objetos remotos.

Este contrato possui duas regiões principais, `Low_Cost` (linhas 07 a 19) e `Available` (linhas 21 a 31), que representam diferentes estados de QoS, de acordo com o número de réplicas desejado pelo cliente. A região `Low_Cost`, representa o estado onde o cliente deseja apenas uma réplica, conforme descrito em seu predicado (linha 07), e a região `Available` representa o interesse do cliente em um número de réplicas maior ou igual a dois (linha 21). Nas transições de estado definidas para essas duas regiões (linhas 33 a 38), é feita uma chamada a um método do objeto `ReplMgr` (linhas 35 e 37) para que o mecanismo gerenciador de réplicas possa ajustar o grau de replicação dos objetos remotos, de acordo com o desejado pelo cliente.

Aninhadas a essas duas regiões, existem outras regiões (`Low`, `Normal` e `High`) cujos predicados utilizam o objeto `MeasuredNumberReplicas` para comparar o atual número de réplicas com o número desejado pelo cliente. Por exemplo, a região `Available` possui duas regiões internas, `Low` e `Normal` (linhas 23 a 25), e para que a primeira esteja ativa, é necessário que os predicados de `Available` e de `Low` sejam verdadeiros. Isso representaria o estado onde o cliente deseja o número de réplicas maior ou igual a dois (predicado de `Available` na linha 21), porém o número atual é menor que isso (predicado de `Low` na linha 23). Nas transições de estado dessas regiões internas (linhas 13 a 18 e 26 a 30), o cliente é informado, através do *Callback* `ClientCallback`, que houve mudança na disponibilidade do serviço. Por exemplo, quando a região `Available` está ativa e o predicado da região interna `Low` passa a ser verdadeiro, o cliente é informado que a disponibilidade do serviço degradou, já que o número de réplicas atual é menor que o desejado.

```
01 contract repl_contract(
02   syscond ValueSC ValueSCImpl ClientExpectedReplicas,
03   callback AvailCB ClientCallback, syscond ValueSC ValueSCImpl
04   MeasuredNumberReplicas, syscond ReplSC ReplSCImpl ReplMgr) is
05
06   negotiated regions are
07    region Low_Cost : when ClientExpectedReplicas == 1 =>
08      reality regions are
09        region Low    : when MeasuredNumberReplicas < 1 =>
10        region Normal : when MeasuredNumberReplicas == 1 =>
11        region High   : when MeasuredNumberReplicas > 1 =>
12
13        transitions are
14          transition any -> Low: ClientCallback.availability_degraded();
15          transition any -> Normal:
16                              ClientCallback.availability_back_to_normal();
17          transition any -> High:ClientCallback.resources_being_wasted()
18        end transitions;
19      end reality regions;
20
21    region Available : when ClientExpectedReplicas >= 2 =>
22      reality regions are
23        region Low: when MeasuredNumberReplicas<ClientExpectedReplicas =>
24        region Normal : when MeasuredNumberReplicas >=
25                             ClientExpectedReplicas =>
26        transitions are
27          transition any -> Low: ClientCallback.availability_degraded();
28          transition any -> Normal:
29                              ClientCallback.availability_back_to_normal();
30        end transitions;
31      end reality regions;
32
33      transitions are
34        transition Low_Cost -> Available:
35          ReplMgr.adjust_degree_of_replication(ClientExpectedReplicas);
36        transition Available -> Low_Cost:
37          ReplMgr.adjust_degree_of_replication(ClientExpectedReplicas);
38      end transitions;
39    end negotiated regions;
40  end repl_contract;
```

Código 6.1 – Contrato QuO em CDL para controle de réplicas de objetos remotos.

6.1.5 Comparação

A proposta QuO é a que tem maior grau de relação com a CR-RIO. As duas propostas possuem o mesmo objetivo e é possível estabelecer um mapeamento entre elas. Primeiramente, comparando suas arquiteturas (QuO na seção 6.1.1 e CR-RIO na seção 3.2), seus componentes podem ser relacionados conforme mostrado na Tabela 6.1.

	QuO	**CR-RIO**
Medição de propriedades de recurso	SysCond	QoSAgent
Interpretação dos valores medidos do sistema	Contract	Contractor
Escolha do comportamento a ser tomando	Delegate	Contract Manager

Tabela 6.1 – Comparação entre os componentes da arquitetura dos *frameworks* QuO e CR-RIO.

Diferentemente do CR-RIO, onde as aplicações desenvolvidas não estão amarradas a nenhuma tecnologia de implementação, aplicações QuO são desenvolvidas na plataforma CORBA. Outra diferença é que o QuO não possui uma ADL para descrever a arquitetura da aplicação. O uso da ADL CBabel no CR-RIO, permite ter uma visão do sistema em um nível maior de abstração, o que facilita o seu entendimento, a utilização de técnicas formais de validação [Rademaker et al. 2004, Rademaker 2005] e a realização de alterações na configuração da arquitetura.

No que diz respeito à descrição de contratos de QoS, a linguagem utilizada pelo CR-RIO (CBabel, apresentada na seção 3.1) permite descrever requisitos de QoS e associá-los a componentes da arquitetura do sistema através do conceito de serviços com qualidade diferenciada, o que garante um nível de abstração arquitetural. Já na linguagem utilizada pelo QuO (CDL), os requisitos de QoS são descritos através do conceito de regiões, que desce ao nível de implementação ao chamar diretamente métodos dos objetos de condição do sistema ou *Callbacks*, quando ocorre uma transição de região. Dessa maneira, o contrato QuO mistura a definição de contrato de QoS com a implementação do sistema, o que torna a definição do contrato menos legível e não garante uma separação de interesses nas aplicações QuO. A Tabela 6.2 resume o mapeamento entre os conceitos utilizados na descrição de contratos das duas propostas.

QuO	CR-RIO
Conceito de regiões (Estado de QoS)	Conceito de serviços
Predicados das regiões	*Profiles* do contrato
Transição de estado ocorre por regiões	Transição de estado ocorre por serviços
Callback (chamadas aos componentes da aplicação)	Pode ser definido no nível de implementação (*hotspot*)
Nível de abstração mais baixo (arquitetural e implementação)	Nível de abstração mais alto (arquitetural)

Tabela 6.2 – Comparação entre os conceitos utilizados nos contratos QuO e CR-RIO.

6.2 Rainbow

O *Rainbow* [Cheng et al. 2004, Garlan et al. 2004] é um *framework* que inclui mecanismos que permitem ao projetista de uma aplicação especificar quais aspectos devem ser monitorados e em quais condições uma adaptação no sistema deve ocorrer para garantir os requisitos de qualidade da aplicação. Isso é feito com base em um modelo arquitetural. Durante a execução da aplicação, uma infra-estrutura de suporte monitora as propriedades definidas no modelo, avalia a ocorrência de violações nas restrições especificadas no mesmo e, se for necessário, efetua adaptações no sistema [Schmerl e Garlan 2002].

6.2.1 Visão Geral

O *Rainbow* é centrado em uma abordagem baseada em arquitetura e provê uma infra-estrutura de suporte que oferece mecanismos para se especializar a necessidades de sistemas específicos. O modelo arquitetural é usado para prover uma perspectiva global do sistema e expor suas propriedades e comportamentos. Dessa forma, é possível determinar restrições topológicas e comportamentais para uma aplicação, estabelecer um conjunto de adaptações permitidas e ajudar a garantir a validade dessas adaptações.

Para capturar características comuns a uma determinada família de sistemas, o *Rainbow* utiliza o conceito de estilo arquitetural, que caracteriza uma categoria de sistemas relacionados por propriedades semânticas e estruturais comuns. Ele estende esse conceito com a inclusão do estilo de adaptação (*adaptation style*), onde são definidos operadores e estratégias de adaptação responsáveis por capturar atributos dinâmicos do sistema e adaptá-lo

através de operações primitivas definidas nesse estilo [Cheng et al. 2002]. Assim, um estilo arquitetural no *Rainbow* é definido pelas seguintes entidades:

- **Tipos de componentes e conectores** definem tipos de componentes (Cliente, Servidor, Banco de dados) e conectores (HTTP, RPC), determinando o vocabulário dos elementos que constituem um tipo e sua interface;
- **Restrições** que determinam a composição permitida dos elementos instanciados dos tipos de componentes e conectores. Em um estilo cliente-servidor, por exemplo, pode ser definida uma restrição indicando que um servidor deve estabelecer conexão com um único cliente;
- **Propriedades** são atributos dos tipos de componentes e conectores que provêem informações analíticas, comportamentais ou semânticas. Por exemplo, as propriedades carga de processamento e tempo de resposta de um servidor em um estilo cliente-servidor interessado em desempenho;
- **Análises** que podem ser executadas em sistemas construídos em um determinado estilo arquitetural. Por exemplo, a teoria de fila pode ser utilizada para analisar o desempenho de um sistema construído em estilo cliente-servidor;
- **Operadores de adaptação** especificam um conjunto de ações específicas a um estilo que podem ser executadas para alterar a configuração dos elementos do sistema ou para consultar suas propriedades. Em um sistema cliente-servidor, por exemplo, pode ser definido um operador responsável por adicionar um novo servidor a um grupo de servidor, para aumentar a capacidade de carga de processamento daquele grupo, quando for observado que a propriedade carga do grupo está acima de um valor máximo estabelecido;
- **Estratégias de adaptação** especificam as adaptações que podem ser aplicadas quando o sistema entra em uma condição não desejada. As propriedades dos tipos de componentes são utilizadas em uma estratégia para avaliar se seus valores estão de acordo com as condições desejadas. Os operadores de adaptação são utilizados para executar ações de adaptação quando uma restrição do sistema é violada. Uma estratégia pode ser composta por várias táticas, que determinam possíveis soluções para a resolução de um problema. Por exemplo, em um sistema cliente-servidor, uma condição pode ser o tempo de resposta abaixo de um máximo estabelecido. Assim, se essa condição for violada, uma estratégia de adaptação é chamada. Ela pode ser

composta por uma tática que verifica se o problema está na carga do servidor e por outra que verifica se está no enlace de conexão do cliente com o servidor. Identificado o problema, ações de adaptações são efetuadas para corrigi-lo.

Um estilo arquitetural pode ser desenvolvido através da ferramenta gráfica AcmeStudio [Schmerl e Garlan 2004], que suporta a definição de um sistema através da descrição dos tipos de seus componentes e conectores e as regras sobre sua composição. Através dela é possível empacotar toda essa definição em um estilo arquitetural que pode ser especializado em diferentes domínios de aplicação.

6.2.2 Componentes do Framework Rainbow

O *framework Rainbow* propõe uma infra-estrutura de suporte a adaptação dinâmica que não está amarrada a nenhum sistema específico, podendo assim, ser reutilizada nos sistemas desenvolvidos. Conforme pode ser visto na Figura 6.3, essa infra-estrutura é dividida em três camadas com seus respectivos componentes:

- ***System layer:*** camada que determina a interface de acesso ao sistema, sendo composta pelos seguintes componentes:
  - ***Probes***, que permitem consultar informações de mecanismos de medição que observan e medem o estado do sistema;
  - ***Resource discovery***, que oferece suporte a consultas por novos recursos;
  - ***Effectors,*** que efetuam modificações no sistema em execução.
- ***Architecture layer:*** esta camada representa o núcleo do *Rainbow*. Nela é implementado o mecanismo de suporte a adaptação no nível arquitetural, de acordo com o modelo estabelecido. Os seguintes componentes são responsáveis por isso:
  - ***Gauges***, agregam informações oriundas dos *Probes* e atualizam as propriedades correspondentes no modelo arquitetural [Garlan et al. 2001];
  - ***Model manager***, manipula e provê acesso ao modelo arquitetural do sistema;
  - ***Constraint evaluator,*** verifica o modelo periodicamente e dispara uma adaptação caso uma restrição seja violada;

- ***Adaptation engine,*** determina o curso da ação e executa adaptações necessárias;
- ***Types and properties*** (tipos de componentes e suas propriedades), ***Rules*** (regras de restrição), ***Strategies and tactics*** (estratégias e táticas de adaptação) e ***Operators*** (operadores de adaptação) fazem parte do conhecimento específico descrito em um estilo arquitetural.

- ***Translation infrastructure:*** camada responsável por mediar o mapeamento de informações entre o nível arquitetural e o nível de sistema. Ela contém um repositório para armazenar os mapeamentos. Por exemplo, o mapeamento de um identificador no nível arquitetural para um endereço IP no nível de sistema.

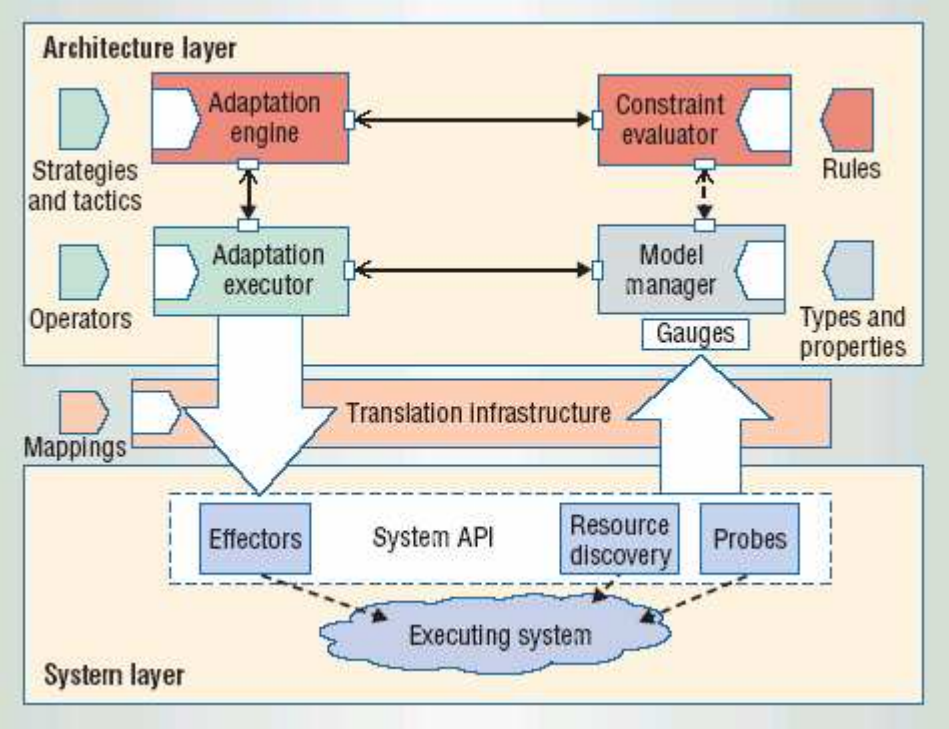

Figura 6.3 – Arquitetura do *framework Rainbow* [Garlan et al. 2004].

6.2.3 Exemplo Cliente-Servidor Web

O cenário deste exemplo consiste de um conjunto de clientes Web que remetem requisições de conteúdo a grupos de servidores. Um cliente conecta-se a um grupo de servidor e envia uma requisição que é armazenada em uma fila de requisições compartilhada pelo grupo. Os servidores do grupo então, vão retirando as requisições da fila e processando-as.

A restrição de qualidade para esta aplicação é o tempo de resposta observado pelos clientes. Duas propriedades que influenciam esta restrição são: a banda passante do enlace entre o cliente e um grupo de servidores e a carga de processamento desse grupo. Para manter a restrição de qualidade, essas propriedades são monitoradas e verificadas se estão de acordo com os valores desejados. Caso seja verificado o contrário, uma estratégia de adaptação é efetuada para tentar adequar o sistema aos parâmetros de qualidade requeridos.

Primeiramente, é preciso definir o estilo para o sistema desta aplicação, conforme pode ser visto no Código 6.2. No inicio do código são definidos os tipos dos componentes que fazem parte do sistema, onde: `ClientT` representa o cliente da aplicação, `ServerT` representa o servidor, `ServerGroupT` representa o grupo de servidores e `LinkT` representa o enlace de comunicação entre o cliente e o grupo de servidores. Depois, são definidas as propriedades dos tipos dos componentes que interferem nos requisitos de qualidade e, portanto, precisam ser monitoradas. São elas: `ClientT.responseTime` o tempo de resposta observado pelo cliente, `ServerT.load` a carga de processamento de um determinado servidor, `ServerGroupT.load` a carga de processamento de um determinado grupo de servidores; `LinkT.bandwidth` a banda passante entre um cliente e um grupo de servidores. Essas propriedades serão monitoradas e utilizadas na verificação de violação das restrições de qualidades definidas. Por último, são definidos os operadores de adaptação: `ServerGroupT.addServer` que procura e adiciona um servidor disponível a um grupo de servidor para aumentar a sua capacidade de processamento e `ClientT.move(ServerGroupT toGroup)` que desconecta um cliente do seu atual grupo de servidor e o conecta a um outro grupo.

```
01  Types: ClientT, ServerT, ServerGroupT, LinkT.
02  Properties: ClientT.responseTime, ServerT.load, ServerGroupT.load,
03              LinkT.bandwidth.
04  Operators: ServerGroupT.addServer, ClientT.move(ServerGroupT toGroup).
```

Código 6.2 – Estilo arquitetural Rainbow para o exemplo cliente-servidor Web.

Para a especificação dos níveis de qualidade desejados é utilizado o conceito de invariante, que inclui um predicado lógico envolvendo as propriedades monitoradas e os valores desejados. Quando esse predicado é negado, uma chamada a uma estratégia de adaptação é disparada. O Código 6.3 mostra a declaração do invariante para o exemplo em questão (*invariant*, na linha 01), onde o predicado `self.responseTime <`

`maxResponseTime` compara se o tempo de resposta observado pelo cliente é menor que um tempo máximo estabelecido. Caso essa condição não seja satisfeita, a estratégia de adaptação `responseTimeStrategy` é chamada (linha 02). Essa estratégia (linhas 04 a 17) é dividida em duas táticas que tentam diminuir o tempo de resposta de acordo com o máximo permitido. A primeira tática (linhas 05 a 09) verifica se a carga do grupo de servidores ao qual o cliente está conectado está acima do máximo permitido (linha 06), adicionando um novo servidor àquele grupo (linha 07), caso o predicado seja verdadeiro. A segunda tática (linhas 10 a 15) verifica se a banda passante do enlace de comunicação entre o cliente e o grupo de servidores é menor que um mínimo estabelecido (linha 11). Caso essa condição seja verificada, a tática é procurar um novo grupo de servidores, cujo enlace com o cliente satisfaça a condição de qualidade desejada (linha 12), e então, conectar o cliente a esse novo grupo (linha 13). Caso o problema não esteja em nenhuma das duas táticas, a estratégia retorna "falso" (linha 16), indicando que a adaptação não foi realizada. Esse código é associado com cada cliente da aplicação.

```
01  Invariant C : responseTime  <=  maxResponseTime
02  ! → responseTimeStrategy(C);
03
04  strategy responseTimeStrategy (C : ClientT) {
05    let G : ServerGroupT = findConnectedServerGroup(C);
06    if ( query("load", G) > maxServerLoad )
07        G.addServer();
08        return true;
09    }
10    let conn : LinkT = findConnector( C , G );
11    if ( query("bandwidth", conn) < minBandwidth ) {
12        let G : ServerGroupT = findBestServerGroup( C );
13        C.move( G );
14        return true;
15    }
16    return false;
17  }
```

Código 6.3 – Estratégia de adaptação Rainbow para o exemplo cliente-servidor Web.

6.2.4 Comparação

A proposta *Rainbow* se concentra em definir adaptações dinâmicas no sistema, acionadas quando condições não desejadas por uma aplicação em execução são detectadas através do monitoramento das propriedades dos componentes. Diferente do CR-RIO, ela não trabalha com o conceito de serviços com qualidade diferenciada, definindo apenas uma linguagem de alto nível onde as ações de adaptação do sistema são orientadas. Em CR-RIO, um contrato de QoS para uma aplicação é descrito através da definição dos seus serviços com restrições de QoS associadas, o que garante que um serviço só seja iniciado, caso os requisitos de QoS possam ser atendidos. Em *Rainbow* não existe preocupação direta com inicialização de serviço, pois é admitido que a aplicação já esteja em execução. Desta forma, um sistema poderia ser iniciado sem que as restrições de QoS pudessem ser satisfeitas.

A representação da arquitetura de um sistema no *Rainbow* é feita através da ADL Acme [Garlan et al. 1997] e é utilizada a ferramenta gráfica AcmeStudio para descrevê-la [Schmerl e Garlan 2002]. Essa ferramenta também permite que os *Gauges* sejam associados a propriedades de componentes da arquitetura. Em contraste, o CR-RIO utiliza a ADL CBabel, que permite que os requisitos não-funcionais sejam descritos, valorados e associados aos serviços da aplicação. O conceito de serviço com qualidade diferenciada utilizado em CBabel identifica melhor o tipo dos recursos a serem utilizados e provê um maior grau de abstração no nível de arquitetura. Correlacionando com a ferramenta AcmeStudio, está sendo desenvolvido um *plugin* para o Eclipse que permite elaborar a arquitetura de uma aplicação através de elementos gráficos [Sztajnberg e Braga 2003].

Embora o *Rainbow* não utilize explicitamente o conceito de contrato, os invariantes podem ser considerados contratos de QoS, já que definem níveis de aceitação por uma aplicação para determinadas propriedades do sistema. Por exemplo, o tempo de resposta observado por um cliente, em um cenário cliente-servidor, deve estar abaixo de um valor máximo estabelecido.

A descrição de uma categoria de QoS (*QoSCategory*) é utilizada no CR-RIO para definir as propriedades relacionadas aos recursos de um sistema, como a banda passante em um canal de comunicação. Além disso, é possível definir propriedades que estabelecem alguma política associada a um recurso. Por exemplo, pode ser definida uma propriedade para caracterizar as possíveis políticas de distribuição de carga em um servidor, como a propriedade `loadDistributionPolicy` descrita no Código 4.4. Na proposta *Rainbow*

essas propriedades são definidas nos tipos dos componentes, como pode ser visto na linha 4 do Código 6.2, que descreve a propriedade `bandwidth` pertencente ao tipo de componente `LinkT`. A diferença é que a linguagem utilizada na definição de uma *QoSCategory* possui um poder de expressão maior para definir as propriedades dos recursos a serem utilizados por uma aplicação. Por exemplo, é possível estabelecer quais valores de uma propriedade são preferidos, através das palavras reservadas *increasing* e *decreasing* (ver seção 3.1.2). É possível também especificar um conjunto valores para uma propriedade, como por exemplo, na propriedade `allocationPolicy` descrita no Código 4.4, que declara os valores `bestMem`, `bestCPU` e `optim` representando as possíveis políticas de alocação de réplicas de servidor.

Também é possível estabelecer uma relação entre os componentes da arquitetura do *framework Rainbow* com os do CR-RIO (respectivamente seção 3.2 e 6.2.2). No nível de sistema, o *Rainbow* utiliza os *Probes* para efetuar medições das propriedades de sistema, enquanto o CR-RIO utiliza os *QoSAgents*. A análise dos dados medidos é feita pelo *Contractor* no CR-RIO, que verifica se os valores estão violando as restrições de um serviço. Correlacionando, o *Rainbow* utiliza os *Gauges* para agregar esses valores no modelo arquitetural e o *Constraint evaluator* para validar se eles estão de acordo com o esperado. Caso não esteja, os componentes *Adaptation engine*, *Adaptation executor* e *Effectors* se encarregam de efetivar as estratégias de adaptação, que são orientadas pelos operadores de adaptação. Na detecção de violação dos requisitos de QoS de um serviço no CR-RIO, ocorre a tentativa de negociação de um novo serviço, que é gerenciada pelo componente *Contract Manager*. Um novo serviço pode requerer alterações no sistema, que são orientadas pelas primitivas de configuração arquitetural (*link*, *start* e *instantiate*) e efetuadas pelo componente *Configurator*. A Tabela 6.3 resume as comparações feitas entre as duas propostas.

	Rainbow	**CR-RIO**
ADL	Acme	CBabel
Descrição de propriedades de recursos	Propriedades de tipos de componente	*QoSCategory*
Conceito de serviço	Não	Sim
Conceito de contrato	Não *	Sim

Monitoração de recurso	*Probes*	*QoSAgent*
Validação de requisitos de QoS	*Constraint evaluator*	*Contractor*
Descrição de ações de adaptação	Operadores de adaptação	Primitivas de configuração arquitetural
Efetivação de adaptação	*Adaptation engine*, *Adaptation executor* e *Effectors*	*Configurator*

* não define explicitamente, porém os invariantes podem ser considerados contratos.

Tabela 6.3 – Comparação entre as propostas *Rainbow* e CR-RIO.

6.3 HQML

HQML (*Hierarchical QoS Markup Language*) é uma linguagem projetada para suportar a especificação de políticas e requisitos de QoS para aplicações distribuídas na Web, particularmente aplicações multimídia [Gu et al. 2002]. Ela propõe um modelo básico, descrito na linguagem XML (*Extensible Markup Language*), que define uma estrutura de dados onde serão descritas essas políticas e requisitos de QoS. Essa estrutura de dados consiste de um conjunto de campos XML (*tags*) que pode se estendido para adicionar informações necessárias a uma aplicação que não estejam contempladas no modelo básico.

6.3.1 Visão Geral

O projeto desta linguagem se baseou em um modelo de componente de aplicação genérico para caracterizar a estrutura de aplicações multimídia distribuídas. Nesse modelo, os componentes de uma aplicação são construídos como tarefas que executam operações sobre dados multimídia, tais como transformação, agregação, *prefetching* e filtragem. Cada componente aceita entrada e gera saída com níveis de QoS (Q^{in} e Q^{out}). Os dados podem ser objetos multimídia básicos (texto, imagem, fluxo de áudio ou vídeo) ou objetos compostos por vários tipos de dado multimídia. As tarefas podem ser conectadas formando um grafo acíclico direcionado, chamado de configuração da aplicação, que conecta provedores de serviço a usuários finais.

Além dessas características, HQML permite que um mesmo serviço seja adaptado para ser oferecido a diferentes tipos de clientes. Essa característica é adequada a aplicações multimídia como a ilustrada na Figura 6.4. Nela, um cliente em uma estação de trabalho conectada a uma rede de alta velocidade pode receber vídeos em formato MPEG (configuração A) transmitidos por um servidor, enquanto que um cliente em um PDA conectado a uma rede sem fio recebe imagens *bitmap* de menor qualidade, previamente adaptadas por um transcodificador MPEG-bitmap (configuração B).

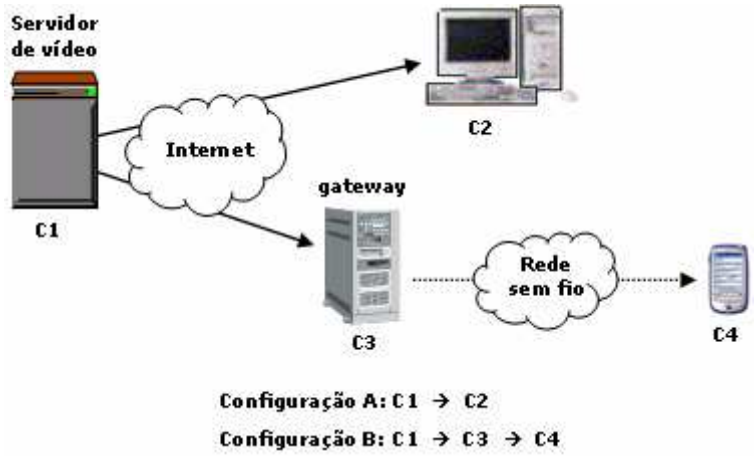

Figura 6.4 – Exemplo de aplicação multimídia utilizada em HQML.

Em HQML a configuração de uma aplicação é descrita através de uma estrutura hierárquica que apresenta na sua base componentes atômicos encarregados de funções básicas (um decodificador MPEG-2 ou transcodificador MPEG-bitmap, por exemplo). Uma coleção de componentes atômicos conectados em um único *host* para estabelecimento de um serviço é chamada de componente composto. Um grupo de *hosts* interconectados e de mesmo tipo forma um *cluster*. A conexão entre esses componentes é feita por *links*, que podem ser de três tipos: *link* fixo, que define um canal de comunicação que não pode ser movido durante o tempo de execução; *link* de *host* móvel, canal sem fio com mobilidade de máquina; e *link* de usuário móvel, mobilidade de usuário de uma máquina para outra.

6.3.2 Especificação de QoS

A especificação de QoS em HQML para aplicações multimídia é dividida em três níveis: *Nível Usuário*, que leva em consideração critérios qualitativos de QoS sob o ponto de vista do usuário (e.g., qualidade alta, média ou baixa para um vídeo); *Nível Aplicação*, especificação dos parâmetros (e.g., taxa de frame e resolução para a transmissão de um vídeo)

e políticas de qualidade (e.g., regras de adaptação) para uma aplicação; e *Nível Recurso de Sistema*, representando os requisitos de QoS para os recursos de sistema utilizados pela aplicação (e.g., memória, processamento e banda passante).

A especificação no *Nível Usuário* é utilizada, em tempo de execução, como parâmetro para determinar a melhor combinação entre a opção de serviço mais econômica para o usuário, o nível de QoS por ele desejado e os níveis de QoS providos pelos diferentes serviços oferecidos por um provedor. Essa especificação apresenta três partes principais: (1) descrições gerais para uma aplicação (e.g., nome e o provedor de serviço), (2) várias configurações para a aplicação associadas com seus respectivos critérios de qualidade (serviços com qualidade diferenciada), (3) o modelo de preço a ser utilizado pelo provedor de serviço (e.g., pagamento por minuto ou por *bytes* transmitidos).

A aplicação multimídia distribuída ilustrada na Figura 6.5 (*Live Media Streaming*) será utilizada para exemplificar o uso da especificação de QoS através da linguagem HQML. Conforme ilustrado nesta figura, a aplicação pode ser disposta em duas configurações: (a) onde o cliente (PC) receberá dados multimídia (vídeo e áudio) com alta qualidade, a um custo alto e caso o nível de disponibilidade dos seus recursos seja suficiente; (b) o cliente (PDA) possui recursos limitados e, por isso, receberá os dados em formatos inferiores de qualidade a um custo mais baixo. O *Servidor 2* ilustrado nas duas configurações é utilizado como "espelho" do primeiro (*Servidor 1*), podendo ser utilizado caso este fique sobrecarregado ou indisponível. O elemento *Gateway* presente na segunda configuração é responsável por transcodificar os dados para os formatos suportados pelo cliente PDA.

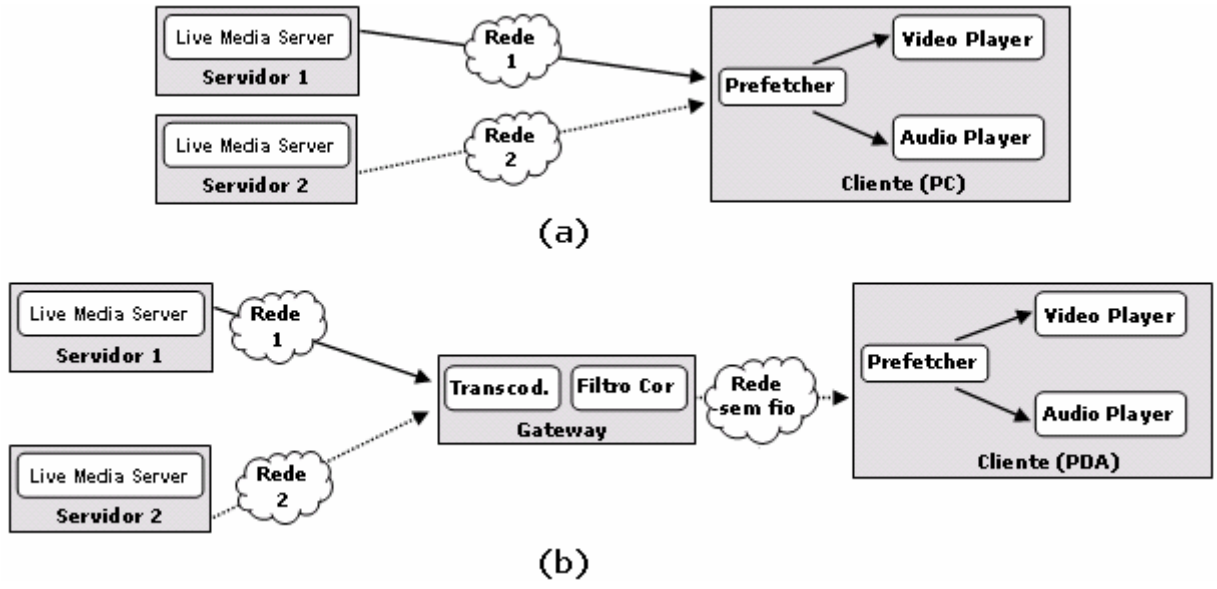

Figura 6.5 – Aplicação multimídia com cenários de diferentes qualidades em HQML.

O Código 6.4 exemplifica a especificação de QoS referente ao *Nível Usuário* para a aplicação *Live Media Streaming*. Na linha 01, os nomes da aplicação e do provedor de serviço são descritos. No resto do código são descritas três opções de configuração com seus respectivos níveis de qualidade. A configuração “100” (linhas 02 a 07) corresponde ao caso (a) da Figura 6.5, a “101” (linhas 08 a 13) ao caso (b) dessa mesma figura e a configuração “200” (linhas 14 a 19) pode representar, por exemplo, uma aplicação de cirurgia remota, onde é desejada a preferência do vídeo transmitido ter imagem nítida (critério de QoS `clarity` na linha 16). Nas duas primeiras configurações a preferência é por transmissão sem interrupções (critério `smoothness` nas linhas 04 e 10), porém na configuração “101”, o cliente receberá o vídeo com qualidade inferior (critério `average` na linha 09). Nos outros dois casos é desejada alta qualidade para o vídeo (critério `high` nas linhas 03 e 15). Por fim, nas duas últimas linhas de cada configuração é descrito o valor inicial para o serviço e a forma que ele será cobrado.

```
01  <App name = “Live Media Streaming” ServiceProvider = “Company X”>
02     <Configuration id = “100”>
03        <UserLevelQoS> high </UserLevelQoS>
04        <QoSPreference> smoothness </QoSPreference>
05        <Price unit = “$”> 5 </Price>
06        <PriceModel> flat rate </PriceModel>
07     </Configuration>
08     <Configuration id = “101”>
09        <UserLevelQoS> average </UserLevelQoS>
10        <QoSPreference> smoothness </QoSPreference>
11        <Price unit = “$”> 1 </Price>
12        <PriceModel> flat rate </PriceModel>
13     </Configuration>
14     <Configuration id = “200”>
15        <UserLevelQoS> high </UserLevelQoS>
16        <QoSPreference> clarity </QoSPreference>
17        <Price unit = “$”> 2 </Price>
18        <PriceModel> per hour increase </PriceModel>
19     </Configuration>
20  </App>
```

Código 6.4 – Especificação HQML no *Nível Usuário* para a aplicação *Live Media Streaming*.

A especificação no *Nível Aplicação* orienta um *QoS Proxy* – sistema de *middleware* genérico com suporte a gerenciamento de QoS (negociação, adaptação, alocação e reserva de

recursos, monitoramento e descoberta de recursos) – a efetuar as configurações necessárias no sistema para que a aplicação possa funcionar conforme os parâmetros de qualidade declarados no *Nível Usuário*. Dessa forma, para cada configuração declarada neste último nível (Código 6.4) é criada uma especificação no *Nível Aplicação* que descreve as configurações da aplicação para o nível de qualidade desejado.

Para a configuração (a) da aplicação *Live Media Streaming* (Figura 6.5), um exemplo de especificação no *Nível Aplicação* é o descrito no Código 6.5, onde são especificados os parâmetros e políticas de QoS para a configuração “100” descrita no Código 6.4. Essa associação está descrita na linha 01 do Código 6.5. No campo `CriticalQoS` (linhas 02 a 07) são especificados os parâmetros críticos de QoS (limites para a taxa de frame por segundo do vídeo reproduzido no cliente), de forma a garantir a qualidade exigida pela configuração associada (configuração “100”). Da linha 08 a 25 são descritos os componentes que fazem parte da arquitetura da aplicação. O servidor é declarado como do tipo `replacable` (linha 09), indicando que esse componente deve ser substituído por um outro do mesmo tipo, caso a restrição de QoS seja violada. Os campos `HostAddr` (linhas 10 a 12 e 13 a 15) indicam os endereços dos *hosts* onde o componente será baixado e instanciado. O cliente é do tipo `required` (linha 20), o que significa que ele deve ser localizado e instanciado por um *QoS Proxy* no *host* que atenda as restrições definidas para o seu hardware e software (linhas 21 e 22). A interligação desses componentes é descrita nas linhas 26 a 32, indicando o tipo de link para efetuar essa ligação (linha 27).

Por fim, é descrita no Código 6.5 uma regra para adaptar a aplicação no caso de degradação do recurso banda passante (linhas 33 a 43). A adaptação descrita (linhas 38 a 40) consiste em alterar a configuração de aplicação da “100” para a “101”, que seria a especificação no *Nível Aplicação* para a configuração “101” do Código 6.4, ou seja, diminuir a qualidade do vídeo. O campo `Notification` especifica uma notificação que será enviada ao usuário quando a adaptação ocorrer (linha 42).

```
01  <AppConfig id = “100”>
02     <CriticalQoS type = “frame rate”>
03        <Range unit = “fps”>
04           <UpperBound> 40 </UpperBound>
05           <LowerBound> 30 </LowerBound>
06        </Range>
07     </CriticalQoS>
08     <ServerCluster>
```

```
09        <Server type = “replacable”>
10           <HostAddr type = “primary”>
11              paris.cs.uiuc.edu
12           </HostAddr>
13           <HostAddr type = “alternative”>
14              boston.cs.uiuc.edu
15           </HostAddr>
16           ...
17        </Server>
18     </ServerCluster>
19     <ClientCluster>
20        <Client type = “required”>
21           <Hardware> Pentium PC 500 </Hardware>
22           <Software> Windows 2000 </Software >
23           ...
24        </Client >
25     </ClientCluster>
26     <LinkList>
27        <Link type = “FixedLink”>
28           <Start> Server </Start>
29           <End> Client </End>
30           ...
31        </Link>
32     </LinkList>
33     <ReconfigRuleList>
34        <ReconfigRule>
35           <Condition type = “Bandwith”>
36              very low
37           </Condition>
38           <ReconfigAction type = “switch to”>
39              101
40           </ReconfigAction>
41        </ReconfigRule>
42        <Notification> Reconfigured </Notification>
43     </ReconfigRuleList>
44  </AppConfig>
```

Código 6.5 – Exemplo de especificação HQML no *Nível Aplicação* para a aplicação *Live Media Streaming*.

O campo `AdaptationRuleList`, exemplificado no Código 6.6, também é utilizado para descrever políticas de adaptação para aplicação. Esse código descreve as regras de

adaptação para o cliente PDA da aplicação *Live Media Streaming* (Figura 6.5), onde valores abstratos (e.g., *high*, *low*, *very low*) qualificam os recursos sendo gerenciados, sendo usados como condição para iniciar uma adaptação (linhas 06 a 08). Em seguida é definida a ação a ser realizada no momento da adaptação (linhas 09 a 12), consistindo na chamada do método `ChageColorDepth`, do componente `ColorFilter`, responsável por alterar a resolução de cores de uma imagem. Esse código ainda inclui a característica interativa de HQML (como as *tags* <*Notification*> e <*Feedback*>, linhas 13 e 14), permitindo interações entre usuários e *QoS Proxies*.

```
01  <Client type = “required”>
02     <Hardware> PDA </Hardware>
03     ...
04     <AdaptationRuleList>
05        <AdaptationRule>
06           <Condition type = “Bandwith”>
07              low
08           </Condition>
09           <Action>
10              <Component> ColorFilter </Component>
11              <Method> ChangeColorDepth </Method>
12           </Action>
13           <Notification> color degrade ! </Notification>
14           <Feedback> early or late </Feedback>
15        </AdaptationRule>
16     </AdaptationRuleList>
17  </Client>
```

Código 6.6 – Exemplo de política de adaptação em HQML para o cliente PDA da aplicação *Live Media Streaming*.

No último nível de especificação HQML, o *Nível Recurso de Sistema*, são descritos os requisitos para os recursos de sistema que serão utilizados por uma aplicação. Se a infra-estrutura de suporte a aplicação oferecer serviços de gerenciamento de QoS, essa descrição pode ser utilizada pelo *QoS Proxy* para efetuar reservas nos recursos requeridos de acordo com os níveis de qualidade desejados.

O Código 6.7 descreve a especificação no *Nível Recurso de Sistema* para o cliente PDA da aplicação *Live Media Streaming* (Figura 6.5). Nesse código são especificados os parâmetros de QoS para os recursos processamento (linhas 08 a 11) e memória (linhas 12 a

15) do dispositivo PDA. No campo `ThresholdList` os parâmetros de QoS declarados em alto nível na configuração da aplicação são mapeados para valores de níveis de recurso de sistema. Por exemplo, o parâmetro `very low` declarado para a propriedade `bandwidth` no Código 6.5 (linhas 35 a 37) pode ser mapeado para valores do nível de sistema, conforme descrito nas linhas 28 a 35 do Código 6.7.

```
01  <Client type = “required”>
02     <Hardware> PDA </Hardware>
03     <Software> Windows CE </Software>
04     ...
05     <Atomic type = “optional”>
06        <Name> Prefetcher </Name>
07        ...
08        <CPU unit = “%”>
09           <Average> 30 </Average>
10           <Deviation> 10 </Deviation>
11        </CPU>
12        <Memory unit = “KB”>
13           <Average> 3 </Average>
14           <Deviation> 1 </Deviation>
15        </Memory>
16          ...
17     </Atomic>
18     ...
19     <ThresholdList>
20        <high type = “CPU”>
21           <LowerBound unit = “%”>
22              40
23           </LowerBound>
24           <UpperBound unit = “%”>
25              60
26           </UpperBound>
27        </high>
28        <very low type = “Bandwidth”>
29           <LowerBound unit = “KB”>
30              60
31           </LowerBound>
32           <UpperBound unit = “KB”>
33              100
34           </UpperBound>
```

```
35        </very low>
36        ...
37     </ThresholdList>
38  </Client>
```

Código 6.7 – Especificação HQML no *Nível Recurso de Sistema* para os recursos do cliente PDA.

6.3.3 Ferramentas de Apoio

Para apoiar o uso da HQML foi criado um ambiente visual de programação QoS (*QoSTalk*) que permite ao desenvolvedor traçar graficamente configurações para aplicações e especificar seus requisitos de QoS no nível usuário e no nível aplicação [Gu et al. 2001]. Em seguida o desenvolvedor deve realizar testes de consistência sobre as configurações (*Testador de Consistência*) utilizando as ferramentas providas pelo *QoSTalk*. Caso nenhuma inconsistência seja detectada, as configurações são passadas ao *Compilador QoS Distribuído*, que se encarrega de ler os parâmetros de QoS para os recursos de sistema (declarados no *Nível Recurso de Sistema*) e de estabelecer automaticamente os mapeamentos entre esses parâmetros e os do nível de aplicação. Ao final, as configurações com especificações completas de QoS (os três níveis de especificação) são passadas ao *Gerador HQML*, responsável pela geração do arquivo HQML final, que é então salvo em um repositório no servidor Web/HQML. A arquitetura do ambiente *QoSTalk* está ilustrada na Figura 6.6.

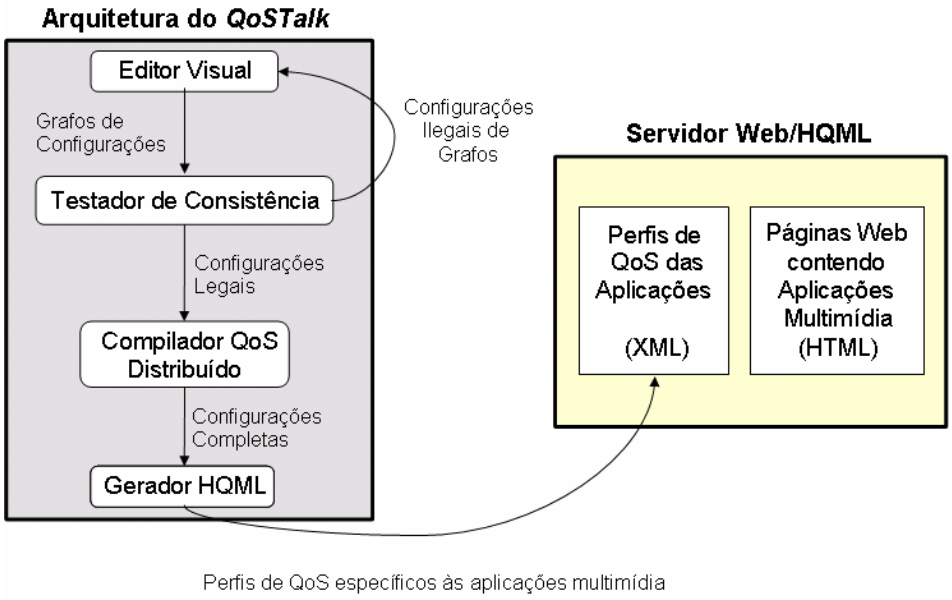

Figura 6.6 – Arquitetura do ambiente *QoSTalk*.

As especificações HQML geradas precisam ser interpretadas com o propósito de extrair as informações necessárias ao provimento de QoS às aplicações multimídia distribuídas. A Figura 6.7 mostra o funcionamento do suporte a esse provimento pela proposta HQML. O componente *Executor HQML*, que é instalado no navegador Web do cliente, é encarregado de interpretar especificações HQML coletando as informações necessárias em favor dos elementos de suporte a gerenciamento de QoS (*QoS Proxies*). Esses elementos não são implementados nesta proposta, sendo assumido que qualquer ferramenta com esse tipo de suporte existente no mercado pode ser utilizada. De acordo com a arquitetura da proposta HQML, os seguintes passos são efetuados no provimento de QoS a uma aplicação em execução:

(1) O *Executor HQML* intercepta as requisições *http* do cliente Web para uma determinada aplicação multimídia. Ele então solicita a *QoS Proxies*, como mecanismo de descoberta de recursos e monitores, os níveis de disponibilidade dos recursos do cliente (e.g., CPU, banda passante, memória, disco).

(2) O *Executor* passa as requisições do cliente, juntamente com os níveis dos seus recursos e as informações sobre a sua plataforma (e.g., PDA, laptop, Windows CE, PalmOS), para o servidor Web/HQML. O servidor procura em seu repositório o arquivo HQML relacionado com a aplicação requisitada (Perfis de QoS), tentando identificar nas especificações nele descritas configurações da aplicação que atendam os requisitos de qualidade desejados pelo cliente e os níveis de disponibilidade dos seus recursos. As configurações que satisfazem esses requisitos, ou uma mensagem de falha caso nenhuma seja encontrada, são retornadas para o cliente Web.

(3) O *Executor HQML* solicita ao usuário da aplicação que escolha uma das possíveis configurações recebidas do servidor. Na escolha o usuário pode se basear, por exemplo, nos diferentes preços associados às configurações possíveis. O *Executor* interpreta então a especificação da configuração escolhida, obtendo as informações necessárias para instrumentar os *QoS Proxies* que efetuarão as configurações necessárias no sistema (e.g., reserva de recursos).

Depois desses três passos efetuados, os *QoS Proxies* são responsáveis por garantir os requisitos de QoS requeridos. Para isso, eles consultam os Perfis de QoS do Usuário para personalizar as regras de adaptação e (re)configuração. Além disso, os *QoS Proxies* colaboram entre si no ambiente distribuído com intuito de configurar e manter os níveis de QoS de acordo com as políticas e requisitos recebidos do *Executor HQML*.

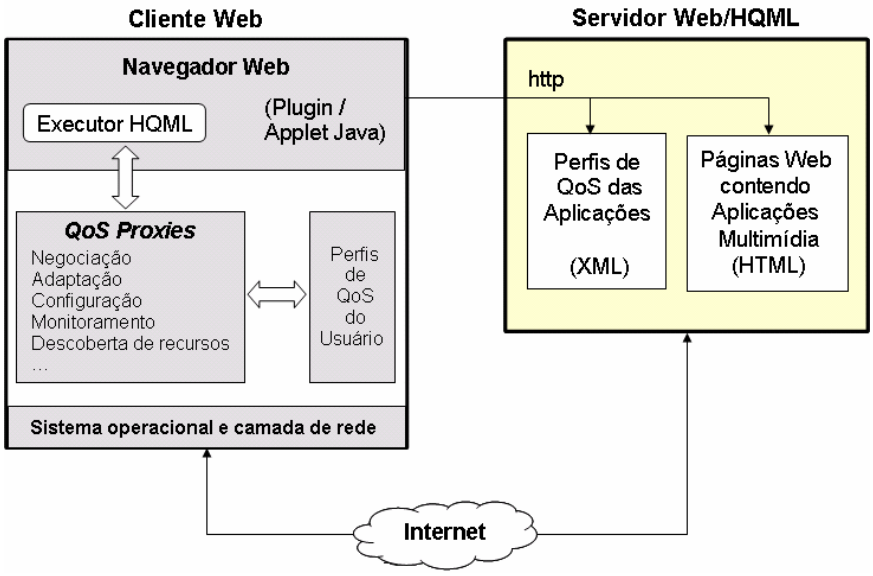

Figura 6.7 – Funcionamento da HQML no gerenciamento de requisitos de QoS.

6.3.4 Comparação

O objetivo da proposta HQML é prover qualidade de serviço para aplicações multimídia distribuídas na Web. Desta forma, o provimento de QoS fica restrito a aplicações no ambiente Web. Diferente disso, a proposta CR-RIO não possui nenhuma limitação no que diz respeito ao ambiente de uma aplicação a ser gerenciada, podendo inclusive gerenciar requisitos de QoS para aplicações Web.

Em relação à descrição de requisitos de QoS, a HQML utiliza modelos descritos na linguagem XML – definição de tipo de documento (DTD) – que consistem na definição de campos que serão utilizados para especificar os parâmetros de qualidade desejados para a aplicação. Essa especificação divide-se em três níveis (*Nível Usuário*, *Nível Aplicação* e *Nível Recurso de Sistema*), o que ajuda a garantir a separação de interesses, permitindo que a descrição dos parâmetros de QoS seja feita em diferentes níveis de abstração. O uso da linguagem XML para a estruturação e especificação dos parâmetros de qualidade facilita a interpretação das informações necessárias no processo de provimento de QoS.

Correlacionando, o CR-RIO utiliza a ADL CBabel para definir propriedades de recursos e para descrever contratos de QoS que especificam os limites desejados dessas propriedades para os serviços oferecidos por uma aplicação (ver seção 3.1). Um contrato em CBabel é descrito em nível arquitetural, garantindo um alto grau de abstração. Além disso, as propriedades de recursos utilizadas nos perfis de qualidade descritos no contrato também

podem ser especificadas em diferentes níveis de abstração. A propriedade que determina a qualidade desejada por um cliente para uma conferência (linha 02 do Código 4.8) é um exemplo de especificação em alto nível de abstração, e a propriedade que especifica o atraso em um canal de comunicação (linha 07 do Código 4.2), um exemplo em baixo nível. Em face à HQML, que possibilita apenas especificar os parâmetros de QoS para aplicações, CBabel possui a característica adicional de permitir orientar adaptações no sistema através de suas primitivas arquiteturais (seção 3.1.1). O uso dessas primitivas garante um poder maior para expressar as adaptações a serem realizados no sistema, necessárias para assegurar que os serviços de uma aplicação continuem a ser oferecidos perante variações nos níveis dos recursos por ela utilizados.

Diferente do *framework* CR-RIO que implementa uma infra-estrutura de suporte ao gerenciamento de aplicações com requisitos não-funcionais, a proposta HQML implementa apenas um elemento responsável por coordenar a escolha da configuração de uma aplicação e por orientar os elementos de suporte a provimento de QoS de acordo com os parâmetros de qualidade declarados nas especificações HQML (*HQML Executor*). Ela assume que o suporte necessário ao processo de provimento de QoS (negociação, adaptação, monitoramento e descoberta de recursos) pode ser oferecido por elementos de *middlewares* destinados a esse fim (*QoS Proxies*), sem se preocupar em definir os elementos encarregados de cada etapa desse processo. No CR-RIO, todo o processo de negociação de serviços com requisitos de QoS é implementado, e através da especialização dos seus elementos, os mecanismos de acesso a propriedades de recurso e de adaptação do sistema podem ser implementados ou acessados, seguindo um modelo padrão (seção 5.1).

6.4 Outras Propostas

Uma infra-estrutura que suporta o desenvolvimento de aplicações distribuídas que podem se adaptar automaticamente à qualidade de serviço de seus componentes é proposta em [Moura et al. 2002]. Baseada na linguagem de programação Lua e na plataforma CORBA, ela permite que componentes sejam selecionados dinamicamente de acordo com requisitos de qualidade. Um mecanismo (LuaMonitor) que permite o monitoramento de requisitos de qualidade, que podem ser especificados dinamicamente, efetua o processo de monitoração das propriedades relacionadas com a qualidade exigida por uma aplicação. Quando mudanças relevantes nas propriedades monitoradas são detectadas, estratégias de adaptação são ativadas

com intuito de adequar a aplicação ao novo estado do sistema. A linguagem Lua é utilizada para descrever as estratégias de adaptação, e por ela ser uma linguagem interpretada, essas estratégias podem ser atualizadas dinamicamente. Embora a linguagem Lua contribua para a capacidade de definição dinâmica de estratégias de adaptação, o seu nível de abstração (nível de implementação) é mais baixo que a linguagem CBabel utilizada em CR-RIO.

Uma arquitetura de componentes aberta e reflexiva, chamada QuA (*Quality of service-aware component Architecture*), que suporta aplicações com requisitos de QoS é proposta em [Amundsen et al. 2004]. Essa arquitetura provê um ambiente de execução para componentes e permite que mecanismos de gerenciamento de QoS sejam implantados em tempo de execução para garantir requisitos de qualidades. Para suportar essa característica, QuA utiliza o conceito de reflexão através de um protocolo de meta-objeto (QuAMOP) implementado por um componente do seu núcleo. No modelo proposto por QuA o programador de uma aplicação só precisa especificar os tipos de componentes que serão utilizados, a ligação entre eles e as propriedades de QoS requeridas. Depois disso, a plataforma é responsável por selecionar a implementações para os tipos de componentes que atendam os requisitos definidos para a aplicação. Para suportar esse mecanismo é utilizado o conceito de “plano de serviço” (*service plan*), que descreve a configuração para um tipo de serviço e as propriedades de QoS por ele requeridas. Um tipo de serviço pode ter vários planos de serviço associados, especificando configurações alternativas para a sua implementação. Quando um usuário requisita um serviço, será escolhido, levando em consideração a disponibilidade dos recursos, o plano de serviço que melhor atende os requisitos de QoS definidos pelo programador da aplicação. Em contraste com a proposta CR-RIO, em QuA não houve a preocupação com a especificação dos requisitos não-funcionais de uma aplicação em alto nível de abstração (eles são definidos nas implementações alternativas para cada tipo de serviço).

Alguns outros trabalhos, além dos considerados neste capítulo, propõem linguagens para descrição de QoS, suporte a monitoração de propriedades de recursos e a gerenciamento dos aspectos não-funcionais de aplicações com requisitos de QoS [Sztajnberg et al. 2005].

6.5 Conclusão do Capítulo

As propostas de *frameworks* com suporte a gerenciamento de requisitos de QoS apresentadas neste capítulo correlacionam-se com a CR-RIO, permitindo a realização de comparações entre as abordagens utilizadas por elas para prover esse tipo de gerenciamento. De certa forma, podemos identificar em todas elas os elementos que constituem a infra-estrutura geral de suporte a gerenciamento de aplicações com requisitos não-funcionais apresentada na seção 2.2.

Uma delas é a proposta *Quality Objects* (QuO), que permite a especificação e o gerenciamento de requisitos de qualidade para aplicações na plataforma CORBA. Embora QuO ofereça separação de interesses por possuir um conjunto de linguagens (QDL, *Quality Description Languages*) para descrição dos diversos aspectos de QoS envolvidos em suas aplicações, sua linguagem para descrição de contrato (CDL, *Contract Description Language*) faz referências ao nível de implementação. Essa característica compromete a separação de interesses por misturar aspectos de descrição de requisitos de QoS com a implementação da aplicação.

A proposta *Rainbow* define uma linguagem para descrição de componentes e estratégias de adaptação para aplicações. Através dessa linguagem, o projetista de uma aplicação pode especificar quais aspectos devem ser monitorados e em quais condições uma adaptação no sistema deve ocorrer para garantir os requisitos de qualidade da aplicação. A descrição de uma estratégia de adaptação é feita no nível arquitetural e se restringe a especificar adaptações para serviços já em andamento, perante violações das condições de qualidade desejadas. Diferentemente, um contrato de QoS em CBabel descreve os serviços de uma aplicação associados aos seus requisitos de qualidade, garantindo que um serviço somente seja estabelecido caso seus requisitos possam ser atendidos.

Outra proposta apresentada foi a linguagem HQML (*Hierarchical QoS Markup Language*), que permite a especificação de políticas e requisitos de QoS para aplicações distribuídas na Web, através de modelos criados em XML. É definido um elemento de suporte (*HQML Executor*) responsável por gerenciar o processo de estabelecimento de configuração de sistema, de acordo com os parâmetros de qualidade declarados nas especificações HQML para uma determinada aplicação. Diferente do CR-RIO que não restringe o escopo de aplicações a ser gerenciadas, HQML se atém a aplicações no ambiente Web.

Algumas outras propostas foram descritas de forma mais sucinta na seção 6.4. As comparações das propostas apresentadas neste capítulo com a CR-RIO, permitiram evidenciar algumas questões relacionadas com a abordagem utilizada em CR-RIO: (a) o uso do conceito de arquitetura de *software* utilizado em CR-RIO permite que as configurações dos serviços oferecidos por aplicações sejam descritos em um nível alto de abstração. Isso facilita o entendimento da configuração da aplicação e a realização de adaptações em sua arquitetura; (b) o conceito de serviços utilizado nos contratos de CR-RIO mostrou-se mais adequado, já que eles permitem descrever, em nível arquitetural, as configurações dos elementos da arquitetura da aplicação para um serviço e associá-las a restrições de QoS que determinam a qualidade desejada e os níveis requeridos para os recursos utilizados; (c) o escopo de aplicação do CR-RIO não é restrito como em HQML, já que sua infra-estrutura de suporte pode ser utilizada para gerenciar qualquer aplicação em ambiente distribuído; (d) o conceito de estratégia de adaptação definida em *Rainbow* permite expressar configurações na arquitetura de um sistema de forma mais poderosa, porém a definição de políticas através da descrição de propriedades de categorias de QoS em CBabel, pode ser utilizada para orientar melhor as configurações a serem realizadas no sistema.

	QuO	**Rainbow**	**HQML**	**CR-RIO**
Escopo de aplicação	Aplicações na plataforma CORBA	Qualquer aplicação[1]	Aplicações Web	Qualquer aplicação[1]
Uso de ADL	Não	Acme	Não	CBabel
Abordagem das linguages	Regiões de QoS	Estratégia de adaptação	Especificação em XML	Serviços com requisitos de QoS
Nível de abstração	Implementação	Arquitetural	Níveis usuário, aplicação e recurso de sistema	Arquitetural
Ferramenta de apoio	Compilador / gerdador de código	AcmeStudio	QoSTalk	Compilador / verificador para CBabel (pesquisa)

Tabela 6.4 – Resumo das principais características das propostas QuO, Rainbow, HQML e CR-RIO.

[1] Em determinados ambientes (e.g., um ambiente móvel) pode ser necessário o uso de algum mecanismo (e.g., o uso de um *gateway*) que permita a comunicação com os componentes de suporte dessas propostas.

A Tabela 6.4 resume as principais características das propostas QuO, *Rainbow*, HQML e CR-RIO. Além dessas, há várias outras propostas para suporte a gerenciamento de requisitos de QoS para aplicações no ambiente distribuído [Wang et al. 2000, Poellabauer et al. 2002, Wohlstadter et al. 2004, Weis et al. 2004, Wang et al. 2004, Suthon et al. 2004, Al-Ali et al. 2003].

Capítulo 7

Conclusão

Neste capítulo são descritas inicialmente as contribuições que este trabalho oferece em relação ao gerenciamento de aplicações com requisitos não-funcionais. Depois são apresentadas e discutidas algumas sugestões de trabalhos futuros, que levantam alguns pontos que devem ser tratados na implementação do *framework* CR-RIO, com intuito de facilitar a sua utilização e garantir a sua confiabilidade no gerenciamento de contratos de QoS para aplicações no ambiente distribuído.

7.1 Contribuição

Este trabalho teve como objetivo refinar o *framework* CR-RIO, proposto originalmente em [Curty 2002], e desenvolver a sua implementação, permitindo mostrar a sua viabilidade no gerenciamento de aplicações com requisitos não-funcionais. Inicialmente foi feito um estudo sobre as questões envolvidas no processo de gerenciamento desse tipo de aplicação. Esse estudo permitiu identificar as etapas necessárias para esse gerenciamento e uma infra-estrutura geral de suporte formada por elementos – Especificação, Monitoramento, Gerência e Configuração – responsáveis pelas as ações requeridas nesse processo. Algumas propostas relacionadas com as responsabilidades de cada um desses elementos são apresentadas nesse estudo.

7.1.1 Refinamentos do CR-RIO

Depois do estudo inicial, a proposta CR-RIO foi examinada, levantando alguns pontos de melhoria. Ela incorpora o conceito de serviços com qualidade diferenciada, onde é possível descrever os serviços a serem oferecidos por uma aplicação associados aos níveis de qualidade requeridos. Essa descrição é feita através da CBabel, uma linguagem de descrição de arquitetura (ADL) que foi estendida para permitir a descrição de contratos de QoS para aplicações. Esse trabalho propôs algumas modificações para essa linguagem, com o intuito de otimizá-la, tornando-a de mais fácil compreensão, e de resolver alguns aspectos formais, de forma a permitir a sua interpretação consistente. Além da linguagem CBabel, a proposta CR-RIO conta com uma infra-estrutura de suporte composta por um conjunto de componentes responsáveis pelo gerenciamento de aplicações com requisitos não-funcionais. A arquitetura dessa infra-estrutura foi reavaliada neste trabalho, resultando na otimização de sua configuração e permitindo identificar melhor as responsabilidades de cada um de seus componentes no processo de gerenciamento de requisitos de QoS de aplicações no ambiente distribuído.

Para validar as modificações propostas e exemplificar o uso do CR-RIO no gerenciamento de aplicações com requisitos de QoS, foram apresentados alguns exemplos de aplicações que requerem níveis de qualidade para garantir o provimento dos seus serviços. Um deles foi uma aplicação de vídeo sob demanda, onde clientes interessados em reproduzir vídeos em suas máquinas podem, através da descrição de um contrato de QoS CBabel, definir as qualidades desejadas e aceitáveis para um vídeo e os níveis de recurso requeridos por cada qualidade. Nesse exemplo, o gerenciamento feito pelo CR-RIO garante que um vídeo sendo transmitido seja transcodificado para um formato de pior qualidade (descrito como de menor preferência no contrato), caso algum recurso fique escasso ao ponto de não satisfazer os níveis de recurso requisitados para a qualidade preferida. Esse gerenciamento garantiria a continuidade da reprodução do vídeo sem a necessidade de algum tipo de interferência pelo usuário.

Um outro exemplo, uma aplicação cliente-servidor Web, permitiu mostrar a funcionalidade do CR-RIO no gerenciamento de aplicações em um ambiente distribuído mais complexo. Neste exemplo, foram descritos dois contratos: um para o cliente da aplicação e outro para o gerenciamento de seus servidores. Desta forma, cada cliente poderia expressar o nível de qualidade aceitável, no caso o tempo de resposta observado para atendimento de sua

requisição enviada ao servidor. Isso possibilitaria ao CR-RIO gerenciar os recursos necessários para satisfazer a qualidade requerida por cada cliente, ou seja, ele gerenciaria os contratos de cada cliente de forma independente. Para os servidores da aplicação, o contrato se encarrega de especificar as políticas de gerenciamento de réplicas de servidor (e.g., adição ou remoção de uma réplica), de acordo com a demanda por recurso de processamento. No exemplo foram mostrados dois grupos de servidores que teriam seus contratos gerenciados pelo CR-RIO de forma independente. Isso garante que cada grupo terá seu controle de réplica gerenciado de acordo com a demanda por processamento de cada um, podendo expressar suas preferências (e.g., política de distribuição de carga) e condições para o ajuste do número de réplicas de forma particular.

O último exemplo apresentado foi uma aplicação de videoconferência, cujo cenário distribuído é composto por clientes heterogêneos. Nesse contexto, foi apresentado como o CR-RIO poderia ser utilizado para gerenciar os contratos de QoS descritos para cada usuário, levando em consideração suas particularidades no que diz respeito aos seus recursos e à qualidade por eles desejada. Por exemplo, um cliente responsável em participar de uma videoconferência utilizando um dispositivo PDA, descreveria em seu contrato um serviço cuja configuração determine o uso de um *gateway* responsável por efetuar a negociação necessária para que ele participe da videoconferência. Além disso, esse cliente poderá especificar em seu contrato os formatos de vídeo que ele suporta e os níveis de seus recursos necessários para a reprodução de vídeo em cada um dos formatos suportados. Em contrapartida, um cliente que queira participar de uma videoconferência através de uma máquina *desktop*, não precisaria de um *gateway* e poderia especificar que deseja reproduzir o vídeo recebido em formatos de maior qualidade, já que dispõe de mais recursos para isso.

7.1.2 Implementação do CR-RIO

Após ter exemplificado o uso do *framework* CR-RIO através dos exemplos apresentados, foram mostrados alguns aspectos de sua implementação. A implementação do CR-RIO foi modelada e desenvolvida de forma a permitir que ela pudesse ser especializada às particularidades de cada aplicação, seguindo um modelo padrão (*v*. seção 5.1.4). Esse modelo facilitou o uso do *framework* sem restringir a flexibilidade necessária para o tratamento de questões específicas ao gerenciamento de cada aplicação (*hotspots*). Para testar e validar a implementação do CR-RIO, foi desenvolvida uma aplicação de vídeo sob demanda. Essa aplicação consiste de um servidor que recebe solicitações de transmissão de vídeos por

clientes conectados através de uma rede. Ao enviar uma solicitação, um cliente informa a qualidade em que ele deseja receber o vídeo, escolhendo entre duas possibilidades, MJPEG ou H.263. Através de um contrato de QoS CBabel foram definidos para cada cliente os níveis dos seus recursos de processamento e a banda passante necessária para reproduzir um vídeo em cada uma das opções de qualidade. De maneira a facilitar a realização dos testes, criando as diversas circunstâncias de gerenciamento, o processo de monitoração das propriedades desses recursos (utilização de CPU e banda passante disponível) foi feito de forma simulada, utilizando interfaces gráficas pelas quais foi possível especificar os valores para essas propriedades, ao longo da execução da aplicação. Contudo, ferramentas como o NWS e o Remos (*v*. seção 2.2.2) poderiam ser utilizadas para prover a medição dessas propriedades.

Nos cenários dos testes realizados para a aplicação de vídeo sob demanda, o servidor foi implantado e um *host* da rede e os clientes em outros *hosts* distintos. Esse cenário permitiu testar o gerenciamento dos serviços descritos no contrato, de acordo com os níveis dos recursos requeridos. A qualidade de vídeo preferida foi a MJPEG e, quando uma escassez de recurso foi simulada, o *framework* gerenciou a adaptação dinâmica para transcodificar o vídeo para o formato H.263. Além dos testes funcionais, foi possível mensurar o desempenho das operações efetuadas pelo CR-RIO no processo de gerenciamento de contrato de QoS para aplicações distribuídas com requisitos não-funcionais. Foi constatado que a sobrecarga média imposta por esse gerenciamento foi de 2,3 ms, um valor que pode ser tolerado por muitas aplicações que necessitam desse gerenciamento dinâmico. A maior parte desse tempo é gasta com comunicação de rede, assim, ele pode variar, dependendo da latência no canal de comunicação entre os componentes remotos do *framework*.

7.1.3 Comparação com Outros Trabalhos

Ainda como contribuição dada por este trabalho, foi apresentado um estudo sobre algumas propostas correlatas ao CR-RIO. Esse estudo permitiu realizar comparações entre as abordagens utilizadas pelas propostas no provimento de suporte a gerenciamento de aplicações com requisitos de QoS. Baseado nessas comparações, foi mostrado que a abordagem arquitetural, centrada no uso da ADL CBabel, utilizada em CR-RIO é adequada para a especificação de aplicações com requisitos não-funcionais, sendo a única que permite descrever as configurações necessárias e os requisitos de QoS para que um serviço possa ser estabelecido, em um nível alto de abstração (arquitetural). Essa característica ajuda a garantir a propriedade de separação de interesses, já que os aspectos funcionais e não-funcionais de

uma aplicação podem ser descritos de forma independente. Além disso, o nível alto de abstração da linguagem e o conceito de serviço com qualidade diferenciada por ela utilizado tornam o processo de descrição de requisitos de QoS para aplicações mais simples, facilitando o seu entendimento. Com relação às infra-estruturas de suporte a gerenciamento de requisitos de QoS providas por algumas das propostas correlatas estudadas, a infra-estrutura de suporte desenvolvida no CR-RIO não é amarrada a nenhuma plataforma específica (como em QuO e em HQML) e mostrou-se bastante flexível, ao permitir o gerenciamento de aplicações em cenários complexos e heterogêneos de sistemas distribuídos.

7.2 Trabalhos Futuros

Com o propósito de facilitar a descrição de arquiteturas de aplicação e de contratos de QoS CBabel, um ambiente visual poderia ser desenvolvido, permitindo que projetistas especifiquem a arquitetura de uma aplicação (seus módulos, conectores e a interligação entre eles) e associem os requisitos não-funcionais requeridos através de uma interface gráfica. Esse ambiente ajudaria um projetista de aplicação a identificar as categorias de QoS (*QoSCategory*), contratos e descrições de arquiteturas contidos no repositório do *framework* CR-RIO, incentivando, assim, a reutilização desses elementos na elaboração de descrições CBabel para novas aplicações.

No modelo de especialização da implementação do *framework* CR-RIO a cada tipo de recurso e contrato de QoS (*v*. seção 5.1.4) foi observado que a implementação de algumas classes especializadas (especializadas de `ResourceState` e `Contractor`) é feita por um mesmo conjunto de passos, diferindo apenas, em algumas informações. Essas informações se referem aos nomes de categorias de QoS e suas propriedades e ao nome de contrato. Como essas informações podem ser obtidas da descrição das categorias e dos contratos de QoS, é possível criar essas classes especializadas de forma automática. Esse recurso poderia ser oferecido através do ambiente visual sugerido anteriormente e reduziria a implementação dos *hotspots* do *framework* à criação das classes especializadas de `Configurator` e `ResourceAgent`.

Outro ponto de sugestão diz respeito aos testes realizados na implementação do *framework* CR-RIO. Embora a aplicação de vídeo sob demanda desenvolvida tenha permitido avaliar os aspectos funcionais do *framework*, seria interessante efetuar testes com aplicações em cenários mais complexos e heterogêneos, como o exemplo de videoconferência

apresentado na seção 4.3. Além disso, testes do gerenciamento provido pelo CR-RIO integrado a um ambiente de suporte a adaptação dinâmica e utilizando mecanismos de descoberta e monitoramento de recursos poderiam ser realizados. Em paralelo a este trabalho, está sendo desenvolvido pelo nosso grupo de pesquisa um ambiente com suporte a adaptação dinâmica de sistemas distribuídos baseados em componentes [Santos 2005]. Ele utiliza a tecnologia *Java Management Extensions* (JMX) e o conceito de arquitetura de *software*, onde através de uma linguagem de descrição arquitetural (ADL) é possível descrever a configuração de uma aplicação e os comandos que orientam a sua implantação. O ambiente provê uma infra-estrutura de suporte responsável por interpretar essa descrição e efetuar as configurações necessárias no sistema para implantar a aplicação. Além desse ambiente, está sendo desenvolvido um suporte a descoberta e monitoramento de recursos distribuídos [Cardoso 2005]. A idéia é integrar esse novo ambiente e suporte ao CR-RIO, permitindo realizar testes mais completos no gerenciamento de aplicações distribuídas com requisitos não-funcionais.

Além de testes envolvendo aplicações em cenários mais complexos, outro ponto que ainda deve ser tratado refere-se à garantia de atomicidade de algumas operações efetuadas pelo *framework* CR-RIO. Por exemplo, como tratar a interrupção ou falhas no processo de estabelecimento de um serviço, como garantir que recursos alocados serão liberados quando o serviço que os utiliza for finalizado. Desta forma, torna-se necessário que seja garantida a atomicidade de algumas operações do CR-RIO, o que poderia ser alcançado através de um mecanismo de suporte a transações. Com o uso de um mecanismo desse tipo, poderia ser garantido, por exemplo, que se na tentativa de estabelecimento de um serviço (*v.* Figura 5.3) uma das ações que envolvem esse processo falhar, o estado inicial ao processo será restabelecido.

Uma outra sugestão para trabalho futuro é representar a descrição de categorias e contratos de QoS CBabel na linguagem XML. As descrições feitas em XML poderiam ser armazenadas em um banco de dados nativo XML, como o Apache Xindice [Xindice 2005], o que facilitaria a consulta pelas informações nelas contidas. Um banco desse tipo seria útil para armazenar informações sobre os recursos distribuídos disponíveis em uma rede, permitindo consultas para localizá-los no processo de descoberta de recursos. Além disso, espera-se que no futuro fabricantes descrevam as características de recursos em uma linguagem padronizada. Assim, as descrições de recursos poderiam ser fornecidas como arquivos XML,

o que facilitaria a capacidade de interoperabilidade entre diversos sistemas que necessitam das informações contidas nessas descrições.

Além das sugestões acima citadas, vêm sendo estudados novos refinamentos na linguagem CBabel, com o propósito de tornar mais clara a idéia da aplicação de contratos para adaptação dinâmica de aplicações. Um dos pontos estudados é a classificação das propriedades de recurso descritas nas categorias de QoS, de forma a permitir a separação entre os perfis que devem ser utilizados na iniciação de serviços (e.g., reserva de recursos) e os perfis que especificam as restrições que devem ser monitoradas para garantir a qualidade requerida pelos serviços. Outro ponto desse estudo é a inclusão da definição de ações de reparo associadas aos serviços do contrato. Essas ações permitiriam descrever, através da linguagem de configuração arquitetural de CBabel, as configurações que devem se efetuadas no sistema toda vez que um serviço for estabelecido. A configuração inicial para um serviço, que deve ser efetuada quando ele for estabelecido pela primeira vez, seria descrito fora das ações de reparo, garantindo que ela seja realizada somente uma vez.

Bibliografia

[Aagedal e Ecklund 2002] Aagedal, J. Ø.; Ecklund, E. F. **Modelling QoS: Towards a UML Profile**. Proceedings of <<UML>> 2002, p. 275-289, Dresden, Alemanha, Setembro, 2002.

[Agha 2002] Agha, G. **Adaptive Middleware**. Communications of the ACM, v. 45, n. 6, p. 31-32, Junho, 2002.

[Al-Ali et al. 2003] Al-Ali, R. J.; Rana, O. F.; Walker, D. W. **G-QoSM: A Framework for Quality of Service Management**. Proceedings of UK e-Science All Hands Meeting 2003, p. 679-686, Nottingham, Reino Unido, Setembro, 2003.

[Amundsen et al. 2004] Amundsen, S.; Lund, K.; Eliassen, F.; Staehli, R. **QuA: Platform-Managed QoS for Components Architectures**. Proceedings of Norwegian Informatics Conference (NIK), Noruega, Novembro, 2004.

[Ansaloni 2003] Ansaloni, S. **Proposta de Padrão Arquitetural para Descrição e Implementação de Contratos de QoS.** Dissertação de Mestrado, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), Julho, 2003.

[Becker e Geihs 1997] Becker, C.; Geihs, K. **MAQS - Management for Adaptive QoS-enabled Services**. IEEE Workshop on Middleware for Distributed Real-Time Systems and Services, São Francisco, EUA, 1997.

[Beugnard et al. 1999] Beugnard, A.; Jézéquel, J.-M.; Plouzeau, N.; Watkins, D. **Making Components Contract Aware**. IEEE Computer, v. 32, n. 7, p. 38-45, Julho, 1999.

[Cardellini et al. 2002] Cardellini, V.; Casalicchio, E.; Colajanni, M.; Yu, P. S. **The State of the Art in Locally Distributed Web-Server Systems**. ACM Computing Surveys, v. 34, n. 2, p. 263-311, Junho, 2002.

[Cardoso 2005] Cardoso, L. X. T. **Integração de Plataformas de Monitoração no Contexto de Arquiteturas Adaptáveis de Software**. Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), 2005.

[Carvalho 2001] Carvalho, S. **Um Design Pattern para a Configuração de Arquiteturas de Software.** Dissertação de Mestrado, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), Janeiro, 2001.

[Cheng et al. 2002] Cheng, S.; Garlan, D.; Schmerl, B.; Sousa, J. P.; Spitznagel, B.; Steenkiste, P. **Using Architectural Style as a Basis for Self-repair**. Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, p. 45-59, Montreal, Canadá, Agosto, 2002.

[Cheng et al. 2004] Cheng, S.; Huang, A.; Garlan, D.; Schmerl, B.; Steenkiste, P. **Rainbow: Architecture-based Self-adaptation with Reusable Infrastructure.** Proceedings of the International Conference on Autonomic Computing (ICAC'04), p. 276-277, Nova Iorque, Nova Iorque, EUA, Maio, 2004.

[Chiba 2000] Chiba, S. **Load-time Structural Reflection in Java**. Proceedings of the 14th European Conference on Object-Oriented Programming (ECOOP '00), p. 313-336, Londres, Reino Unido, 2000.

[Chu et al. 2000] Chu, Y.-H.; Rao, S. G.; Zhang, H. **A Case for End System Multicast**. Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'00), p. 1-12, Santa Clara, Califórnia, EUA, Junho, 2000.

[Curty 2002] Curty, R. **Uma Proposta para Descrição e Implementação de Contratos para Serviços com Qualidade Diferenciada**. Dissertação de Mestrado, Instituto de Computação - Universidade Federal Fluminense (IC/UFF), Novembro, 2002.

[Dinda et al. 2001] Dinda, P. A.; Gross, T.; Karrer, R.; Lowekamp, B.; Miller, N.; Steenkiste, P.; Sutherland, D. **The Architecture of the Remos System**. Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing, p. 252-265, São Francisco, Califórnia, EUA, Agosto, 2001.

[Dowling e Cahill 2001] Dowling, J.; Cahill, V. **The K-Component Architecture Meta-Model for Self-Adaptive Software**. Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, p. 81-88, Londres, Reino Unido, 2001.

[Duran-Limon e Blair 04] Duran-Limon, H. A.; Blair, G. S. **QoS Management Specification Support for Multimedia Middleware**. Journal of Systems and Software, v. 72, n. 1, p. 1-23, 2004.

[Eugene e Zhang 2002] Eugene, T. S.; Zhang, H. **Predicting Internet Network Distance with Coordinates-Based Approaches**. Proceedings of the IEEE INFOCOM 2002, p. 170-179, Nova Iorque, Nova Iorque, EUA, Junho, 2002.

[Fleury e Reverbel 2003] Fleury, M.; Reverbel, F. **The JBoss Extensible Server**. Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware 2003), p. 344-373, Rio de Janeiro, Brasil, 2003.

[Florissi 1996] Florissi, P. G. S. **QoSME: QoS Management Environment**. Tese de doutorado, Universidade de Columbia, 1996.

[Florissi e Yemini 1994] Florissi, P. G. S.; Yemini, Y. **Management of Application Quality of Service**. Proceedings of Fifth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Toulouse, França, Outubro, 1994.

[Freitas et al. 2005] Freitas, G. H.; Cardoso, L. X. T.; Santos, A. L. G.; Loques, O. **Otimizando a Utilização de Servidores através de Contratos Arquiteturais**. Anais do VII Workshop de Tempo Real, p. 35-43, Fortaleza, Brasil, Maio, 2005.

[Frolund e Koistinen 1998] Frolund, S.; Koistinen, J. **Quality of Service Specification in Distributed Object Systems Design**. Proceedings of the 4th USENIX Conference on ObjectOriented Technologies and Systems (COOTS), p. 1-18, Santa Fé, Novo México, Abril, 1998.

[Gamma et al. 1995] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. **Design Patterns – Elements of Reusable Object-Oriented Software**, Addison Wesley Publishing, EUA, 1995.

[Ganek e Corbi 2003] Ganek, A. G.; Corbi, T. A. **The dawning of the autonomic computing era**. IBM Systems Journal, v. 42, n. 1, p. 5-18, Janeiro, 2003.

[Garlan et al. 1997] Garlan, D.; Monroe, R.; Wile, D. **ACME: An Architecture Description Interchange Language**. Proceedings of CASCON'97, p. 169-183, Toronto, Canadá, Novembro, 1997.

[Garlan et al. 2001] Garlan, D.; Schmerl, B.; Chang, J. **Using Gauges for Architecture-Based Monitoring and Adaptation**. The Working Conference on Complex and Dynamic Systems Architecture, Dezembro, 2001.

[Garlan et al. 2004] Garlan, D.; Cheng, S.; Huang, A.; Schmerl, B.; Steenkiste, P. **Rainbow: Architecture-Bases Self Adaptation with Reusable Infrastructure**. IEEE Computer, v. 37, n. 10, p. 46-54, Outubro, 2004.

[Gu et al. 2001] Gu, X.; Wichadakul, D.; Nahrstedt, K. **Visual QoS Programming Environment for Ubiquitous Multimedia Services**. Proceedings of IEEE International Conference on Multimedia and Expo 2001, p. 575-578, Tókio, Japão, Agosto, 2001.

[Gu et al. 2002] Gu, X.; Nahrstedt, K.; Yuan, W.; Wichadakul, D.; Xu, D. **An XML-based Quality of Service Enabling Language for the Web**. Journal of Visual Language and Computing (JVLC), special issue on Multimedia Languages for the Web, v. 13, n. 1, p. 61-95, Academic Press, Fevereiro, 2002.

[Handley et al. 1999] Handley, M.; Schulzrinne, H.; Schooler, E.; Rosenberg, J. **SIP: session initiation protocol**. Request for Comments 2543, Internet Engineering Task Force, Março, 1999. Disponível na Internet. http://www.ietf.org/rfc/rfc2543.txt em 25 de Junho de 2005.

[Horn 2001] Horn, P. **Autonomic Computing: IBM's Perspective on the State of Information Technology**. IBM Corporation, Outubro, 2001. Disponível na Internet. http://researchweb.watson.ibm.com/autonomic/manifesto/autonomic_computing.pdf em 20 de Maio de 2005.

[Huang e Steenkiste 2003] Huang, A.-C.; Steenkiste, P. **Network-Sensitive Service Discovery**. Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03), p. 239-252, Seattle, Washington, EUA, Março, 2003.

[Jin e Nahrstedt 2004] Jin, J.; Nahrstedt, K. **QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy**. IEEE MultiMedia, v. 11, n. 3, p. 74-87, Julho, 2004.

[JMF 2005] Sun Microsystems. **Java Media Framework API (JMF)**. Disponível na Internet. http://java.sun.com/products/java-media/jmf em 10 de Junho de 2005.

[Karr et al. 2001] Karr, D. A.; Rodrigues, C.; Loyall, J. P.; Schantz, R. E.; Krishnamurthy, Y.; Pyarali, I. **Application of the QuO Quality-of-Service Framework to a Distributed Video Application**. Proceedings of the International Symposium on Distributed Objects and Applications, p. 299-308, Roma, Itália, Setembro, 2001.

[Keeney e Cahill 2003] Keeney, J.; Cahill, V. **Chisel: a policy-driven, context-aware, dynamic adaptation framework**. Proceedings of the IEEE 4th International Workshop on Policies for Distributed Systems and Networks, p. 3-14, Lake Como, Itália, Junho, 2003.

[Kon et al. 2002] Kon, F.; Costa, F.; Blair, G.; Campbell, R. H. **The Case for Reflective Middleware**. Communications of the ACM, v. 45, n. 6, p. 33-38, Junho, 2002.

[Lamanna et al. 2003] Lamanna, D. D.; Skene, J.; Emmerich, W. **SLAng: A Language for Defining Service Level Agreements**. Proceedings of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS'03), p. 100-106, Porto Rico, Maio, 2003.

[Li e Nahrstedt 1999] Li, B.; Nahrstedt, K. **A Control-Based Middleware Framework for Quality of Service Adaptations**. IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms, v. 17, n. 9, p. 1632-1650, 1999.

[Lisbôa 2003] Lisbôa, J. **Utilização do design pattern architecture configurator em um ambiente de suporte à configuração de arquiteturas**. Dissertação de Mestrado, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), Abril, 2003.

[Loques et al. 2000] Loques, O.; Sztajnberg, A.; Leite, J.; Lobosco, M. **On the Integration of Configuration and Meta-Level Programming Approaches**. Proceedings of Reflection and Software Engineering, p. 191-210, Springer-Verlag, Heidelberg, Alemanha, Junho, 2000.

[Loques et al. 2004] Loques, O.; Sztajnberg, A.; Cerqueira, R. C.; Ansaloni, S. **A contract-based approach to describe and deploy non-functional adaptations in software architectures**. Journal of the Brazilian Computer Society, v. 10, n. 1, p. 5-18, Julho, 2004.

[Loques et al. 2005] Loques, O. (coordenador); Sztajnberg, A.; Braga, C.; Abelém, A. **Caravela – Contratos para Aplicações em Redes de Alta Velocidade. Relatório 02**. Projeto GIGA – Coordenações Temáticas RNP, Maio, 2005.

[Loques e Sztajnberg 2004] Loques, O.; Sztajnberg, A. **Customizing Component-Based Architectures by Contract**. Proceedings of Second International Working Conference on Component Deployement, p. 18-34, Edimburgo, Reino Unido, Maio, 2004.

[Loyall et al. 1998] Loyall, J. P.; Schantz, R. E.; Zinky, J. A.; Bakken, D. E. **Specifying and Measuring Quality of Service in Distributed Object Systems**. Proceedings of The 1st IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Kyoto, Japão, p.43-52, Abril, 1998.

[Melcher e Mitchell 2004] Melcher, B.; Mitchell, B. **Towards an Autonomic Framework: Self-Configuring Network Services and Developing Autonomic Applications**. Intel Technology Journal, v. 08, n. 04, Novembro, 2004.

[Moreira et al. 2003] Moreira, R. S.; Blair, G. S.; Carrapatoso, E. **Constraining Architectural Reflection for Safely Managing Adaptation**. Proceedings of The 2nd Workshop on Reflective and Adaptive Middleware, IEEE/ACM/USENIX International Middleware Conference, p. 139-143, Rio de Janeiro, Brasil, Junho, 2003.

[Moura et al. 2002] Moura, A. L.; Ururahy, C.; Cerqueira, R.; Rodriquez, N. **Dynamic Support for Distributed Auto-Adaptative Applications**. AOPDCS: Workshop on Aspect Oriented Programming for Distributed Computing Systems (IEEE ICDCS 2002), p. 451-456, Viena, Áustria, Julho, 2002.

[Nahrstedt et al. 2001] Nahrstedt, K.; Xu, D.; Wichadakul, D.; Li, B. **QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments**. IEEE Communications Magazine, v. 39, n. 11, p. 140-148, Novembro, 2001.

[Oreizy et al. 1999] Oreizy, P.; Gorlick, M. M.; Taylor, R. N.; Johnson, G.; Medvidovic, N.; Quilici, A.; Rosenblum, D.; Wolf, A. **An Architecture-Based Approach to Self-Adaptive Software**. IEEE Intelligent Systems, v. 14, n. 3, p. 54-62, Maio, 1999.

[Pal et al. 2000] Pal, P.; Loyall, J.; Schantz, R.; Zinky, J.; Shapiro, R.; Megquier, J. **Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration**. Proceedings of the Third IEEE International Symposium on Object-Oriented Real-time Distributed Computing, 2000 (ISORC 2000), p. 310-319, Newport Beach, Califórnia, EUA, Março, 2000.

[Poellabauer et al. 2002] Poellabauer, C.; Abbasi, H.; Schwan, K. **Cooperative Run-time Management of Adaptive Applications and Distributed Resources**. Proceedings of the tenth ACM international conference on Multimedia, p. 402-411, Juan-les-Pins, França, Dezembro, 2002.

[Rademaker et al. 2004] Rademaker, A.; Braga, C.; Sztajnberg, A. **A Rewriting Semantics for a Software Architecture Description Language**. Anais do Simpósio Brasileiro de Métodos Formais (SBMF 2004), Recife, PE, Brasil, p. 249-255, Novembro, 2004.

[Rademaker 2005] Rademaker, A. **Uma Ferramenta para Especificação e Análise de Arquiteturas de Software**. Dissertação de Mestrado, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), Maio, 2005.

[Rajkumar et al. 1997] Rajkumar, R.; Lee, C.; Lehoczky, J.; Siewiorek, D. **A Resource Allocation Model for QoS Management**. Proceedings of the 18th IEEE Real-time Systems Symposium, p. 298-307, São Francisco, Califórnia, EUA, Dezembro, 1997.

[Santos 2005] Santos, A. L. G. **Técnicas de Configuração Dinâmica de Arquiteturas de Software**. Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), 2005.

[Schantz et al. 2002] Schantz, R.; Loyall, J.; Atighetchi, M.; Pal, P. **Packaging Quality of Service Control Behaviors for Reuse**. Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC 2002), p. 375-385, Washington, DC, EUA, Abril, 2002.

[Schmerl e Garlan 2002] Schmerl, B.; Garlan, D. **Exploiting Architectural Design Knowledge to Support Self-repairing Systems**. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, p. 241-248, Ischia, Itália, Julho, 2002.

[Schmerl e Garlan 2004] Schmerl, B.; Garlan, D. **AcmeStudio: Supporting Style-Centered Architecture Development**. Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), p. 704-705, Edimburgo, Escócia, Maio, 2004.

[Shaw et al. 1995] Shaw, M.; DeLine, R.; Klein, D. V.; Ross, T. L.; Young, D. M.; Zelesnik, G. **Abstractions for Software Architecture and Tools to Support Them**. IEEE Transactions on Software Engineering, v. 21, n. 4, p. 314-335, 1995.

[Shaw et al. 1996] Shaw, M.; DeLine, R.; Zelesnik, G. **Abstractions and Implementations for Architectural Connections**. Proceedings of the Third International Conference on Configurable Distributed Systems, p. 2-10, Annapolis, Maryland, EUA, Maio, 1996.

[Suthon et al. 2004] Suthon, S.-W.; Pung, H.-K.; Zhou, L. **QMan: An Adaptive End-to-End QoS Management Architecture**. Proceedings of the IEEE International Conference on Networks, p. 797-803, Cingapura, Novembro, 2004.

[Sztajnberg 2002] Sztajnberg, A. **Flexibilidade e Separação de Interesses para a Concepção e Evolução de Aplicações Distribuídas**. Tese de Doutorado, COPPE/PEE/UFRJ, Maio, 2002.

[Sztajnberg e Braga 2003] Sztajnberg, A.; Braga, C. **FormArch: A Formal Architecture Development Environment using Eclipse**. eTX:Eclipse Technology Exchange, Sessão de Poster, OOPSLA 2003, Anaheim, Califórnia, EUA, Outubro, 2003. Disponível na Internet. http://www.ic.uff.br/~cbraga/vas/publications/braga-sztajnberg-etx03.pdf em 28 de Julho de 2005.

[Sztajnberg et al. 2005] Sztajnberg, A.; Corradi, A.; Santos, A.; Barros, F.; Cardoso, L.; Loques, O. **Especificação e Suporte de Requisitos Não-Funcionais para Serviços de Nível Alto**. Caderno de Minicursos do 23º Simpósio Brasileiro de Redes de Computadores, p. 223-284, Maio, 2005.

[Toga e Ott 1999] Toga, J.; Ott, J. **ITU-T standardization activities for interactive multimedia communications on packet-based networks: H.323 and related recommendations**. Computer Networks, v. 31, n. 3, p. 205-223, Fevereiro, 1999.

[Tripathi 2002] Tripathi, A. **Challenges Designing Next-Generation Middleware Systems**. Communications of the ACM, v. 45, n. 6, p. 39-42, Junho, 2002.

[Venkatasubramanian 2002] Venkatasubramanian, N. **Safe 'Composability' of Middleware Services**. Communications of the ACM, v. 45, n. 6, p. 49-52, Junho, 2002.

[Wang et al. 2000] Wang, P. Y.; Yemini, Y.; Florissi, D.; Florissi, P. **QoSME: Toward QoS Management and Guarantees**. Proceedings of the International Conference on Communication Technology (WCC - ICCT 2000), v. 1, p. 868-875, Pequim, China, Agosto, 2000.

[Wang et al. 2001] Wang, N.; Schmidt, D. C.; Kircher, M.; Parameswaran, K. **Adaptive and Reflective Middleware for QoS-Enabled CCM Applications**. IEEE Distributed Systems Online, v. 2, n. 5, Julho, 2001.

[Wang et al. 2004] Wang, G.; Chen, A.; Wang, C.; Fung, C.; Uczekaj, S. A. **Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures**. Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC 2004), p. 21-32, Monterey, Califórnia, EUA, Setembro, 2004.

[Weis et al. 2004] Weis, T.; Ulbrich, A.; Geihs, K.; Becker, C. **Quality of Service in Middleware and Applications: A Model-Driven Approach**. Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC 2004), p. 160-171, Monterey, Califórnia, EUA, Setembro, 2004.

[Wohlstadter et al. 2004] Wohlstadter, E.; Tai, S.; Mikalsen, T. A.; Rouvellou, I.; Devanbu, P. T. **GlueQoS: Middleware to Sweeten Quality-of-Service Policy**. Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), p. 189-199, Edimburgo, Reino Unido, Maio, 2004.

[Wolski et al. 1999] Wolski, R.; Spring, T. N.; Hayes, J. **The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing**. Future Generation Computer Systems, v. 15, n. 5-6, p. 757-768, 1999.

[Xindice 2005] The Apache Software Foundation. **The Apache XML Project**. Disponível na Internet. http://xml.apache.org/xindice/ em 12 de Julho de 2005.

[Yahiaoui et al. 2004] Yahiaoui, N.; Traverson, B.; Levy, N. **Classification and comparison of Adaptable Platforms**. Proceedings of the First International Workshop on Coordination and Adaptation Techniques for Software Entites (WCAT04), p. 55-61, Oslo, Noruega, Junho, 2004.

[Yan et al. 2004] Yan, H.; Garlan, D.; Schmerl, B.; Aldrich, J.; Kazman, R. **DiscoTect: A System for Discovering Architectures from Running Systems**. Proceedings of the 26th International Conference on Software Engineering, p. 470-479, Edimburgo, Escócia, Maio, 2004.