3
Descrição Arquitetural dos Frameworks de Provisão de QoS

Neste capítulo são apresentados os estilos arquiteturais, desenvolvidos usando a ADL Wright (Allen, 1997), para servirem como base para a descrição arquitetural formal dos Frameworks de Orquestração de Recursos (Gomes, 1999).

3.1.
Mapeamento dos Frameworks de Provisão de QoS de UML para Wright

Conforme mencionado anteriormente (Seção 2.4.2.3.1), a negociação de QoS define os mecanismos responsáveis pela admissão de fluxos do usuário, durante as fases de requisição e estabelecimento, levando em consideração os recursos disponíveis no ambiente. Os três principais elementos que a constituem são o *Controlador de Admissão*, o *Negociador de QoS* e o *Mapeador de QoS*. Tais elementos são modelados em UML como classes, enquanto associações entre essas classes definem suas regras de interação (Gomes, 1999).

A sintonização de QoS também possui elementos centrais: o *Controlador de Ajuste*, o *Sintonizador de QoS*, o *Mapeador de QoS* e o *Monitor* (vide Seção 2.4.2.3.2). Suas funções são as de manter o nível contratado do serviço através da reorquestração de recursos em caso de iminência de violação, durante a fase de fornecimento do serviço, e também fornecer mecanismos para o monitoramento da real QoS que se oferece ao usuário. Tais elementos também são modelados em UML como classes.

Uma forma direta de mapear os framework para provisão de QoS na ADL Wright é entender cada um desses elementos centrais como tipos de componentes. Em termos mais gerais, são identificadas as classes mais importantes e, então, cada uma delas é vista como um tipo de componente Wright. Estilos arquiteturais podem ser usados tanto para agrupar e definir esses tipos, quanto para especificar tipos de conectores que podem servir para descrever as regras de interação entre

os componentes. Mais do que isso, os estilos permitem a definição de restrições topológicas ou da semântica de execução desses elementos.

Um enfoque ainda mais flexível para definir a arquitetura da negociação e sintonização de QoS é o de fornecer nos estilos apenas os tipos das interfaces e dos conectores, além de restrições. Nesse caso, o projetista fica livre para especificar os tipos dos componentes. A opção menos flexível de pré-definir os tipos de componentes é justificada pelo interesse em diminuir o trabalho do projetista e o de facilitar o posterior mapeamento de uma linguagem de domínio específico (DSL) para Wright (descrito no Capítulo 4).

Uma vez identificados os principais tipos de componentes, o próximo passo é especificar suas portas, descrevendo a forma como os componentes se relacionam com o mundo externo. A interface dos componentes deve ser definida precisamente por causa de uma importante diferença referente às regras de interação no uso de paradigmas de arquitetura de sistemas e de modelagem orientada a objetos (OO). Enquanto a modelagem OO permite apenas restrições na visibilidade da interface das classes (por exemplo, um método *protected* é visível apenas pela classe a que ele pertence e suas subclasses), a prática da arquitetura de sistemas vai além e permite que a ligação de portas de componentes seja restrita a portas específicas (por exemplo, a porta A de um componente só pode ser ligada por meio de um conector à porta B de outro componente). Na modelagem OO, nada impede que um método seja obrigado a executar outro, o que é definido apenas pela implementação. Restrições na ligação entre portas e as próprias regras das portas são importantes para definir a topologia do sistema e a forma como seus elementos constituintes (os componentes) se relacionam.

Uma idéia inicial para o mapeamento entre os métodos das classes e as portas dos componentes seria definir uma porta para cada método. O maior problema desse procedimento é que ele pode tornar as ligações entre instâncias de componentes excessivamente complicadas para o projetista. Quando há muitas chamadas entre métodos de duas classes quaisquer, a ligação entre instâncias de seus respectivos componentes acabaria por resultar na associação entre várias portas ao mesmo tempo. Os conectores acabariam por definir essas regras de associação, o que também os tornariam mais complicados.

Uma solução para essa dificuldade é definir cada possível relação entre classes (e não cada método das classes) como uma porta. A vantagem dessa

abordagem é que a interface dos componentes fica mais clara para o projetista, que não precisa mais se preocupar em como foi feita a implementação (onde se define que métodos executam outros métodos), mas apenas em como são os relacionamentos. Uma outra vantagem adicional é que a instanciação é facilitada quando se deseja associar dois componentes quaisquer, já que há sempre apenas uma porta para ser ligada entre cada um deles e seu conector intermediário.

Convém ressaltar que a intenção aqui não é definir uma metodologia geral para a descrição de frameworks orientados a objetos, como feito em (Arevalo, 2000), mas apenas descrever os passos usados especificamente na tradução dos frameworks de provisão de QoS para a ADL Wright.

O componente que representa o *Controlador de Admissão* pode ser visto na Figura 3.1. As portas *interlevel* e *intralevel* são usadas para fazer a ligação com negociadores de QoS respectivamente de outro subsistema e de seu próprio subsistema. O comportamento do controlador de admissão (*computation*) o caracteriza como uma fachada entre os subsistemas, repassando requisições de contrato de serviço vindas de um subsistema externo para um negociador de QoS ligado a ele.

Em um primeiro momento, pode parecer que a existência de um componente que apenas repassa requisições para admissão de fluxos seja descartável, podendo ser o próprio *Negociador de QoS* responsável por receber essas requisições. Na prática, a separação desses dois componentes permite a sutil e importante vantagem da interface dos subsistemas poder estar conceitualmente e fisicamente (em máquinas distintas) separada dos componentes de negociação. Tomemos como exemplo uma infra-estrutura simples com provisão de QoS contendo duas estações finais e um roteador ligando ambas. Em um alto nível de abstração, esse cenário pode ser modelado como um subsistema com dois *Controladores de Admissão* (nas estações finais) ligados a um *Negociador de QoS* central (no roteador). Se não houvesse o conceito do *Controlador de Admissão*, a modelagem seria muito menos expressiva.

Os controladores de admissão de mais baixo nível não precisam de mecanismos de negociação adicionais, agindo diretamente sobre o serviço de provisão de QoS na criação de recursos virtuais. Não havendo, portanto, negociadores de QoS em seus subsistemas, tais controladores não possuem porta *intralevel* (presente nos demais tipos de controladores de admissão). No lugar

dessa porta, eles são os únicos a possuírem uma porta *metainterface* que é usada na ligação com o serviço de comunicação com QoS[3]. Como será visto adiante, essa diferença na forma como é modelado o controlador de admissão acaba por resultar na definição de um estilo específico para o mais baixo nível de abstração.

Component AdmCtrl

Port interlevel = [*Recebe requisições para admissão de fluxos (admit) vindo de um Usuário ou de um Negociador de QoS em um subsistema externo, depois retorna um identificador de reserva (ResId). OU Recebe confirmações para reserva de recursos com um identificador de reserva (commit) vindo de um Usuário ou de um Negociador de QoS, depois retorna o resultado da confirmação.*]

Port intralevel = [*Existe apenas quando o componente não é primitivo. Repassa requisições para admissão de fluxos (admit) ao Negociador de QoS associado em seu subsistema (request), depois recebe como retorno um identificador de reserva (ResId). OU Repassa confirmações para reserva de recursos (commit) ao Negociador de QoS associado em seu subsistema (confirm), depois recebe como retorno o resultado dessa confirmação.*]

Port metainterface = [*Existe apenas quando o componente é primitivo. Realiza a admissão de fluxos diretamente sobre árvores de recursos virtuais primitivas gerando um identificador de reserva (ResId). Pode também realizar a confirmação da reserva de recursos diretamente sobre árvores de recursos virtuais primitivas.*]

Computation = [*Recebe requisições para admissão de fluxos (admit) vindo de um Usuário ou de um Negociador de QoS em um subsistema externo e as repassa para o Negociador de QoS associado em seu subsistema ou diretamente para árvores de recursos virtuais primitivas. Como resultado, recebe um identificador de reserva (ResId) e o retorna ao componente de onde partiu a requisição. OU Recebe confirmações para reserva de recursos (confirm) vindo de um Usuário ou de um Negociador de QoS em um subsistema externo e as repassa para o Negociador de QoS associado em seu subsistema ou diretamente para árvores de recursos virtuais primitivas e, então, retorna o resultado dessa confirmação.*]

Figura 3.1 – O componente Controlador de Admissão

[3] Um maior detalhamento da porta *metainterface* é deixado como trabalho futuro já que o presente trabalho é centralizado nos meta serviços de negociação e sintonização de QoS. A importância dessa porta está em demonstrar de que forma os meta serviços atuam sobre o serviço principal na criação de recursos virtuais.

O componente que representa o *Mapeador de QoS* é definido na Figura 3.2. A porta *translate* é usada para a ligação com o negociador de QoS associado. Conforme foi visto anteriormente, o papel do mapeador é traduzir parâmetros de QoS entre níveis de visão diferentes, o que é definido pela sua única porta. Isso é feito com base em uma estratégia de mapeamento, que é definida por meio de parametrização do componente.

```
Component QosMapper
   Port translate = [Recebe requisições para a tradução dos parâmetros de QoS
                     entre níveis de visão vindo de um Negociador de QoS ou de
                     um Sintonizador de QoS associado em seu subsistema e
                     retorna como resultado os parâmetros mapeados.]
   Computation    = [Recebe requisições para a tradução dos parâmetros de QoS
                     entre níveis de visão de QoS vindo de um Negociador de
                     QoS ou de um Sintonizador de QoS associado em seu
                     subsistema, realiza a tradução para parâmetros de
                     subsistemas internos, e retorna tais parâmetros mapeados
                     como resultado.]
```

Figura 3.2 – O componente Mapeador de QoS

A Figura 3.3 descreve o componente que representa o *Negociador de QoS*. A porta *translate* é usada para a ligação com a porta de mesmo nome de um mapeador de QoS relacionado com a finalidade de realizar a tradução dos parâmetros de QoS entre os níveis de visão. As portas *interlevel* e *intralevel* são usadas respectivamente em sua ligação com controladores de admissão de subsistemas externos e dentro do mesmo subsistema.

A porta *intraneg* é aplicada na ligação entre negociadores de QoS, quando a negociação naquele subsistema é distribuída. Quando a negociação for centralizada, não há necessidade de porta *intraneg* porque o único negociador gerencia todo o mecanismo de negociação, sem precisar envolver outros componentes. Como será visto adiante, essa diferença caracteriza a existência de um estilo próprio para a negociação de QoS centralizada e outro para a negociação de QoS distribuída.

Component QoSNeg

Port translate = [*Envia requisições para a tradução dos parâmetros de QoS entre níveis de visão diferentes para um Mapeador de QoS associado em seu subsistema e recebe como resultado os parâmetros mapeados.*]

Port interlevel = [*Envia requisições para admissão de fluxos (admit) para um Controlador de Admissão em um subsistema interno, depois recebe um identificador de reserva (ResId). OU Envia confirmações para reserva de recursos (commit) para um Controlador de Admissão em um subsistema interno, depois recebe o resultado dessa confirmação.*]

Port intralevel = [*Recebe requisições para admissão de fluxos (request) vindo de um Controlador de Admissão em seu subsistema, depois retorna um identificador de reserva (ResId). OU Recebe confirmações para reserva de recursos (confirm) vindo de um Controlador de Admissão em seu subsistema, depois retorna o resultado dessa confirmação.*]

Port intraneg = [*Existe apenas quando a negociação de QoS é distribuída. Repassa ou recebe requisições para admissão de fluxos ou confirmações para reserva de recursos para/de outros Negociadores de QoS em seu subsistema.*]

Computation = [*Recebe requisições para admissão de fluxos (request) ou confirmações para reserva de recursos (confirm), depois as repassa para outros Negociadores de QoS ou para Controladores de Admissão, o fazendo, nesse último caso, após traduzir os parâmetros de QoS entre os níveis de visão de QoS por meio dos Mapeadores de QoS associados*]

Figura 3.3 – O componente Negociador de QoS

Com relação à sintonização de QoS, a Figura 3.4 apresenta a definição abstrata do componente que representa o *Controlador de Ajuste*. Seu papel é servir como uma fachada entre subsistemas, de forma muito semelhante ao *Controlador de Admissão* na negociação de QoS. É importante ressaltar que a sintonização é muito mais orientada a eventos, enquanto a negociação é mais orientada a chamada de procedimentos. Os mecanismos de sintonização são acionados automaticamente sempre que uma violação de contrato de serviço estiver na iminência de acontecer, enquanto os mecanismos de negociação são sempre iniciados por uma chamada do usuário.

Na iminência de uma violação de contrato de serviço, o subsistema que a detecta pode decidir reorquestrar recursos nos subsistemas internos para manter o nível desejado de QoS. A requisição para essa sintonização é feita através da porta *interlevel* de um *Controlador de Ajuste* desses subsistemas internos. Quando isso ocorre, tais requisições são simplesmente redirecionadas para o *Sintonizador de*

QoS associado, através da porta *intralevel*. No caso desse *Controlador de Ajuste* estar em um nível de abstração onde a sintonização de QoS é primitiva, ou seja, sem levar em conta outros mecanismos de sintonização, não há porta *intralevel* (presente nos demais tipos de controladores de ajuste). Em seu lugar, há uma porta *metainterface* diretamente responsável pela reorquestração de recursos no provedor de serviços. A definição mais detalhada dessa porta é deixada como trabalho futuro. Nos subsistemas primitivos também está presente a porta *verify*, que pode se ligar a *Monitores* com a finalidade de realizar cálculos estatísticos sobre o fluxo em busca de violações (nesse caso sobre uma árvore de recursos virtuais primitiva).

```
Component AdjCtrl
   Port interlevel = [Recebe requisições para sintonização de recursos vindo de
                      Sintonizadores de QoS de subsistemas externos, depois retorna o
                      resultado da requisição. Repassa alertas de violação de
                      contratos de serviço a um Sintonizador de QoS externo ao
                      subsistema.]
   Port intralevel = [Existe apenas quando o componente não é primitivo.
                      Repassa requisições para sintonização de recursos ao
                      Sintonizador de QoS associado, depois recebe o resultado dessa
                      requisição. Recebe alertas de violação de contratos de serviço
                      vindo do Sintonizador de QoS associado no subsistema.]
   Port verify = [Existe apenas quando o componente é primitivo. Repassa ou
                  recebe requisições de cálculos estatísticos sobre o fluxo vindo de
                  um Monitor associado.]
   Port metainterface = [Existe apenas quando o componente é primitivo.
                         Realiza a sintonização de QoS diretamente sobre árvores de
                         recursos virtuais primitivas.]
   Computation = [Quando não é primitivo, recebe requisições para sintonização de
                  recursos e, então, as repassa para o Sintonizador de QoS
                  associado. Depois recebe o resultado e o retorna ao
                  Sintonizador de QoS de um subsistema externo que originou a
                  requisição. Além disso, recebe alertas de violação de contratos
                  de serviço e repassa para um sintonizador de QoS externo ao
                  subsistema. Quando é primitivo, recebe requisições para
                  sintonização de recursos e atua diretamente sobre uma árvore de
                  recursos virtuais primitiva, retornando o resultado ao
                  Sintonizador de QoS de um subsistema externo que originou a
                  requisição. Além disso, envia alertas de violação de contratos de
                  serviço a um Sintonizador de QoS externo ao subsistema.]
```

Figura 3.4 – O componente Controlador de Ajuste

A Figura 3.5 apresenta a descrição do *Monitor*. A porta *verify* é usada para disparar cálculos estatísticos sobre o fluxo do usuário. Baseado nesses cálculos, o *Sintonizador de QoS* ou *Controlador de Ajuste* primitivo ligado a essa porta verifica a iminência de violações de contratos de serviço.

Component Monitor
Port verify = [*Recebe requisições de cálculos estatísticos vindo de um Sintonizador de QoS ou de um Controlador de Ajuste primitivo associado no subsistema e usa uma estratégia de monitoração para fazer as medidas e, depois, as retorna. Ou ainda usa uma estratégia de monitoração para realizar cálculos estatísticos sobre o fluxo e envia essas medidas para o Sintonizador de QoS ou Controlador de Ajuste primitivo associado.*]
Computation = [*Recebe requisições de cálculos estatísticos vindo de um Sintonizador de QoS ou de um Controlador de Ajuste primitivo associado no subsistema e usa uma estratégia de monitoração para fazer as medidas e, depois, as retorna. Ou ainda usa uma estratégia de monitoração para realizar cálculos estatísticos sobre o fluxo e envia essas medidas para o Sintonizador de QoS ou Controlador de Ajuste primitivo associado*]

Figura 3.5 – O componente Monitor

O componente *Sintonizador de QoS* é representado na Figura 3.6. A porta *verify* é ligada a um *Monitor*, para permitir consultas periódicas aos cálculos estatísticos sobre o fluxo ou o recebimento dessas medidas sem requisitá-las, em busca de violações. A porta *translate* é semelhante à porta de mesmo nome do *Negociador de QoS* e é usada para a tradução dos parâmetros de QoS entre os níveis de visão do componente e de seus subsistemas internos.

A porta *intralevel* do sintonizador de QoS deve ser ligada a controladores de ajuste dentro do mesmo subsistema. Nesse caso, os controladores de ajuste atuam apenas como uma fachada, repassando as requisições recebidas para que ações de reorquestração sejam tomadas pelo sintonizador de QoS.

Quando o sintonizador detecta violações ou recebe pedidos de sintonização originados de subsistemas externos e repassados pelo controlador de ajuste associado através da porta *intralevel*, ele pode optar por reorquestrar recursos nos subsistemas internos para manter o nível desejado do serviço. Isso é feito através da porta *interlevel*, que se liga a controladores de ajuste nos subsistemas internos.

A porta *intratuner* (Figura 3.6) é usada para a ligação entre sintonizadores de QoS dentro do subsistema. Como será visto na próxima seção, a presença ou ausência dessa porta é uma diferença marcante entre os estilos de sintonização centralizado e distribuído.

```
Component QoSTuner
   Port verify = [Envia requisições de cálculos estatísticos e recebe essas medidas
                  da QoS sendo fornecida por meio do Monitor associado. Ou
                  ainda recebe cálculos estatísticos sobre o fluxo de um Monitor
                  associado, sem requisitá-las.]
   Port intralevel = [Recebe requisições para sintonização de recursos vindo de
                  um Controlador de Ajuste dentro de seu subsistema e retorna o
                  resultado dessa requisição. Ou envia alertas de violação de
                  contratos de serviço a um Controlador de Ajuste associado.]
   Port intratuner = [Recebe e envia requisições para sintonização de recursos
                     de/para outros Sintonizadores de QoS presentes em seu
                     subsistema, depois retorna o resultado dessa requisição.]
   Port interlevel = [Repassa requisições para sintonização de recursos a
                      Controladores de Ajuste em subsistemas internos, depois
                      recebe o resultado dessa requisição.]
   Port translate = [Envia requisições para a tradução dos parâmetros de QoS
                     entre níveis de visão diferentes para um Mapeador de QoS
                     associado em seu subsistema e recebe como resultado os
                     parâmetros mapeados.]
   Computation = [Periodicamente, envia requisições de cálculos estatísticos ao
                  Monitor associado, recebendo como resposta algumas medidas
                  da real QoS sendo fornecida, que são usadas para verificar (a
                  iminência de) violações, caso em que se iniciam os
                  MECANISMOS DE SINTONIZAÇÃO;
                  OU recebe cálculos estatísticos vindo de um Monitor associado e
                  verifica (a iminência de) violações, caso em que se iniciam os
                  MECANISMOS DE SINTONIZAÇÃO ; OU recebe requisições de
                  sintonização de QoS vindo de Controladores de Ajuste
                  associados em seu subsistema OU vindo de outros
                  Sintonizadores de QoS em seu próprio subsistema, onde iniciam
                  os MECANISMOS DE SINTONIZAÇÃO;
                  ONDE MECANISMO DE SINTONIZAÇÃO = (i) repassa a
                  requisição para sintonização de QoS a Controladores de Ajuste
                  de subsistemas internos depois de traduzir os parâmetros de
                  forma condizente; OU (ii) repassa a requisição para
                  sintonização de QoS a outros Sintonizadores de QoS em seu
                  subsistema; OU (iii) alerta um outro Sintonizador de QoS
                  externo ao subsistema por intermédio de um Controlador de
                  Ajuste associado.]
```

Figura 3.6 – O componente Sintonizador de QoS

3.2. Estilos Arquiteturais para Negociação de QoS

Conforme mencionado na seção anterior, a especificação dos mecanismos de negociação de QoS na ADL Wright exige a definição de quatro estilos, que são apresentados nesta seção: *LowestNQoS*, para os subsistemas diretamente relacionados a árvores de recursos virtuais primitivas; *CentralizedNQoS*, para

subsistemas com um único negociador de QoS centralizando os mecanismos de negociação; *DistributedNQoS*, para subsistemas com diversos negociadores de QoS interagindo entre si; e *HierarchyNQoS*, definindo a ligação entre subsistemas.

Por motivo de organização, todos os estilos apresentam inicialmente as interfaces usadas para descrever (em CSP) as regras das portas dos componentes. Em seguida, os tipos de componentes são definidos, explorando os tipos de interfaces previamente descritos e explicitando seu comportamento (*computation*). Por fim, os conectores descrevem as regras de interação entre esses tipos de componentes.

As restrições presentes nesses estilos são descritas em português e não em Wright, por esse último ainda apresentar um vocabulário pobre nesse aspecto. O prefixo *intra* no nome das portas indica que ela deve ser ligada a um componente interno ao subsistema, enquanto o prefixo *inter* indica a ligação com componentes externos ao subsistema. Os sufixos *Input* e *Output* no nome dos tipos de interface são usados para distinguir, respectivamente, as que iniciam um processo de recebimento de requisição e aquelas que iniciam um processo de envio de requisição.

O estilo *LowestNQoS*, apresentado graficamente na Figura 3.7, descreve os subsistemas que agem diretamente sobre árvores de recursos virtuais primitivas no controle de admissão e na reserva de recursos. Em tais subsistemas, os únicos componentes presentes são os *Controladores de Admissão*, que são responsáveis pela ligação do meta serviço de negociação de QoS com o serviço principal, manipulando seus recursos através das *árvores de recursos virtuais.*

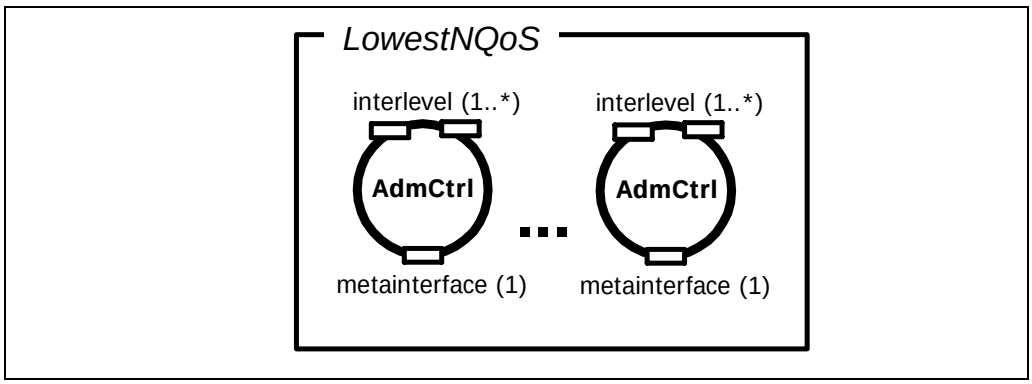

Figura 3.7 – Representação gráfica do estilo *LowestNQoS*

A Figura 3.8 apresenta a especificação em Wright do estilo *LowestNQoS*. O tipo de interface *InterlevelInput* descreve as portas *interlevel* dos *Controladores de Admissão*. Tais portas são associadas a *Negociadores de QoS* de subsistemas externos, por onde são recebidas requisições de admissão de novos fluxos.

Por outro lado, o tipo de interface *MetaInterfaceOutput* regula operações sobre as *fábricas de recursos virtuais* para a criação de novos recursos virtuais. A porta *metainterface*, que só existe nos *Controladores de Admissão* de mais baixo nível, possui esse tipo de interface e deve ser ligada ao serviço principal. Uma definição mais detalhada da meta interface de negociação de QoS é deixada como trabalho futuro.

O tipo de componente *AdmCtrl* recebe como parâmetros tanto uma computação que representa uma estratégia de admissão (*AdmStrategy*) quanto o número de negociadores de QoS nos subsistemas externos (*enegs*). Esse último parâmetro define a cardinalidade da porta *interlevel* já que, para cada *Negociador de QoS* externo, deve haver uma porta desse tipo ligada a um conector *Interlevel* intermediário.

A computação do componente *AdmCtrl* envolve a admissão (*admit*) e confirmação (*commit*) da requisição de novos fluxos. O recebimento de pedidos de admissão por qualquer porta *interlevel* é seguido do acionamento da estratégia de admissão, que se baseia em consultas às *árvores de recursos virtuais*. Tais pedidos são respondidos por meio de um ResId (identificador de recurso), caracterizando a aceitação, ou não, daquele fluxo. Essa resposta é recursivamente retornada aos subsistemas externos (pela mesma porta *interlevel* por onde foi recebida a requisição) até retornar ao usuário.

De posse do ResId, o usuário pode realizar a confirmação (*commit*) daquela requisição, que também desce recursivamente os subsistemas até alcançar novamente os *Controladores de Admissão* de mais baixo nível. Quando isso ocorre, novos recursos virtuais podem ser criados por meio da porta *metainterface*. Finalizando o processo, o resultado final daquela admissão é retornado ao usuário, fazendo o caminho recursivo inverso.

```
Style LowestNQoS

   Interface Type InterlevelInput =
                  (admit?QoSnFlowSpec ->
                   ret_admit!ResId ->
                   InterlevelInput)
                   []
                   (commit?ResId ->
                   ret_commit!result ->
                   InterlevelInput)
                   []
                   Success

   Interface Type MetaInterfaceOutput =
                   ( createVR!ResId ->
                   ret_createVR?result ->
                   MetaInterfaceOutput )
                   |~|
                   Success

   Component AdmCtrl (AdmStrategy : Computation; enegs : 1..)
      Port interlevel_1..enegs = InterlevelInput
      Port metainterface = MetaInterfaceOutput
      Computation = ([] i : 1..enegs @
                     (interlevel_i.admit?QoSnFlowSpec ->
                      AdmStrategy ->
                      interlevel_i.ret_admit!ResId ->
                      Computation))
                    []
                    ([] i : i..enegs @
                     (interlevel_i.commit?ResId ->
                      metainterface.createVR!ResId ->
                      metainterface.ret_createVR?result ->
                      interlevel_i.ret_commit!result
                      Computation))
                    []
                    Success
   Constraints
End Style
```

Figura 3.8 – Estilo *LowestNQoS* para Negociação de QoS

É importante observar que a computação de *AdmCtrl* não possui qualquer restrição impedindo que a confirmação seja feita apenas após a admissão, nem mesmo há garantias de que toda admissão de sucesso seja seguida por uma confirmação de reserva de recursos. A inclusão desse formalismo é deixada como trabalho futuro tanto nesse estilo como nos demais.

O estilo *CentralizedNQoS*, representado graficamente pela Figura 3.9, é usado em subsistemas onde a negociação de QoS é feita de forma centralizada. Por esse motivo, a topologia de tais subsistemas envolve a instanciação de um único *Negociador de QoS* com um ou mais *Controladores de Admissão* ligados a ele.

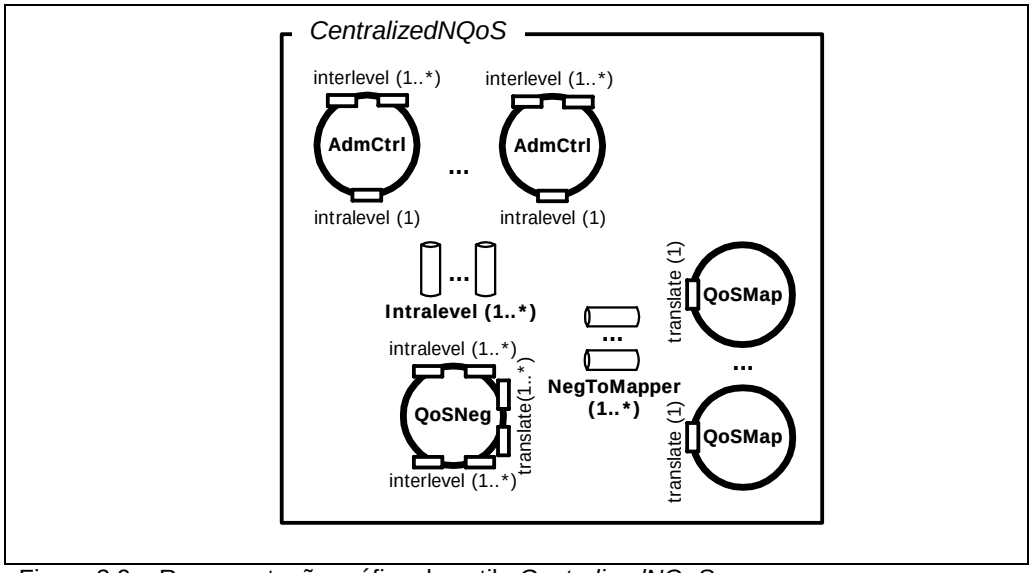

Figura 3.9 – Representação gráfica do estilo *CentralizedNQoS*

A Figura 3.10 apresenta a definição em Wright do estilo *CentralizedNQoS*. Os tipos de interface *IntralevelOutput* e *IntralevelInput* descrevem respectivamente as regras das portas *intralevel* dos *Controladores de Admissão* e *Negociadores de QoS*. O conector *Intralevel* relaciona o papel desses dois componentes, especificando como se dá a comunicação entre eles. Quando ocorre uma requisição de admissão de novos fluxos, o *Controlador de Admissão* apenas repassa os parâmetros recebidos de especificação de QoS e de fluxo para o *Negociador de QoS* associado. Além disso, quando há uma confirmação de reserva de recursos (inicialmente originada pelo usuário com posse de um identificador de reserva), o *Controlador de Admissão* apenas repassa o ResId recebido para o *Negociador de QoS* associado, que então repassa recursivamente aos subsistemas internos.

Os tipos de interface *InterlevelOutput* e *InterlevelIntput* descrevem conjuntamente as regras de interação entre *Negociadores de QoS* e *Controladores de Admissão* em subsistemas internos. O conector *Interlevel* é definido (apenas no estilo *HierarchyNQoS*) de forma muito parecida ao *Intralevel*. A diferença é que os papéis ficam invertidos: agora é o *Negociador de QoS* quem repassa as requisições enquanto o *Controlador de Admissão* as recebe.

O conector *NegToMapper* define as regras de comunicação entre *Negociadores* e *Mapeadores de QoS*. O tipo de interface *TranslateOutput*

especifica que cada pedido de mapeamento de parâmetros de QoS (*map!QoS*) é seguido do retorno desses parâmetros mapeados (*ret_map?MappedQoS*). De forma inversa, *TranslateInput* especifica que cada recebimento de um pedido de mapeamento de parâmetros (*map?QoS*) é seguido do retorno (envio) desses parâmetros mapeados (*ret_map!MappedQoS*). Cabe à especificação do conector *NegToMapper* definir o papel dos componentes envolvidos e a relação desses papéis (*glue*). Nesse caso, todo pedido de mapeamento gerado pelo negociador é recebido pelo mapeador, que depois envia parâmetros mapeados que são recebidos pelo negociador.

O componente *AdmCtrl* é especificado nesse estilo de forma parecida ao feito em *LowestNQoS*. A diferença é que o papel dos *Controladores de Admissão* passa a ser apenas de uma fachada entre subsistemas e não mais atuam diretamente sobre árvores de recursos virtuais primitivas. Por esse motivo, o componente *AdmCtrl* não é mais parametrizável por uma estratégia de admissão.

A definição do componente *QoSMapper* é bastante simples. Isso porque ele possui apenas uma porta, a *translate*, e sua computação se resume a receber pedidos de mapeamento de parâmetros de QoS, traduzi-los para o nível de abstração dos provedores internos (levando em conta sua estratégia de mapeamento) e enviá-los de volta traduzidos.

O tipo de componente *QoSNeg* recebe como parâmetro de instanciação uma estratégia de negociação responsável pelo cálculo de responsabilidades pela provisão de QoS entre os subsistemas (*NegotiationStrategy*) e os números de *Controladores de Admissão* associados dentro do subsistema (*icas*) e fora do subsistema (*ecas*).

A cardinalidade das portas *translate* e *interlevel* são iguais ao parâmetro *ecas*. O motivo é que o *Negociador de QoS* deve ser capaz de traduzir os parâmetros de seu nível de visão para cada nível dos subsistemas internos, o que requer um *Mapeador* relacionado a cada subsistema. Como o número de subsistemas é igual ao de *Controladores de Admissão* externos e ligados ao *Negociador de QoS*, a cardinalidade de ambas as portas é igual. No mais, a cardinalidade da porta *intralevel* é especificada pelo parâmetro *icas*.

A computação de *QoSNeg* é dividida no trecho correspondente à admissão de fluxos (*request*) e no de confirmação da reserva de recursos (*confirm*). Ambos iniciam com uma requisição recebida de algum *Controlador de Admissão* interno

ao subsistema (via porta *intralevel*). No caso da admissão, o processo continua com o disparo da estratégia de negociação, identificando os subsistemas com parcelas de responsabilidade sobre a provisão de QoS daquele serviço. Depois disso, cada subsistema é acionado através do envio de uma nova admissão (*admit*) com parâmetros mapeados para aquele nível de visão. Em seguida, cada subsistema retorna um ResId para que o *Negociador de QoS* verifique se a concatenação daquelas parcelas de responsabilidade pode prover o serviço como um todo. Enfim, um ResId correspondente à concatenação da admissão nos subsistemas é calculado e enviado de volta ao *Controlador de Admissão*, que requisitou a admissão. O processo de confirmação ocorre de forma muito parecida, o que dispensa maiores detalhes.

```
Style CentralizedNQoS

   Interface Type IntralevelInput =
                  (request?QoSnFlowSpec ->
                  ret_request!ResId->
                  IntralevelInput)
                  []
                  (confirm?ResId ->
                  ret_confirm!result ->
                  IntralevelInput)
                  []
                  Success

   Interface Type IntralevelOutput =
                  (request!QoSnFlowSpec ->
                  ret_request?ResId ->
                  IntralevelOutput)
                  |~|
                  (confirm!ResId ->
                  ret_confirm?result ->
                  IntralevelOutput)
                  |~|
                  Success

   Interface Type InterlevelInput =
                  (admit?QoSnFlowSpec ->
                  ret_admit!ResId ->
                  InterlevelInput)
                  []
                  (commit?ResId ->
                  ret_commit!result ->
                  InterlevelInput)
                  []
                  Success

   Interface Type InterlevelOutput =
                  (admit!QoSnFlowSpec ->
                  ret_admit?ResId ->
                  InterlevelOutput)
```

```
                        |~|
                        (commit!ResId ->
                        ret_commit?result ->
                        InterlevelOutput)
                        |~|
                        Success

      Interface Type TranslateInput =
                        (map?QoS ->
                        ret_map!MappedQoS ->
                        TranslateInput)
                        []
                        Success

      Interface Type TranslateOutput =
                        (map!QoS ->
                        ret_map?MappedQoS ->
                        TranslateOutput)
                        |~|
                        Success

      Component AdmCtrl(enegs : 1..)
         Port intralevel = IntralevelOutput
         Port interlevel_1..enegs = InterlevelInput
         Computation = ([] i : 1..enegs @
                         (interlevel_i.admit?QoSnFlowSpec ->
                          intralevel.request!QoSnFlowSpec ->
                          intralevel.ret_request?ResId ->
                          interlevel_i.ret_admit!ResId ->
                          Computation))
                        []
                        ([] i : i..enegs @
                         (interlevel_i.commit?ResId ->
                          intralevel.confirm!ResId ->
                          intralevel.ret_confirm?result_cnf ->
                          interlevel_i.ret_commit!result_cmt ->
                          Computation))
                        []
                        Success

   Component QoSMapper (MappingStrategy : Computation) =
      Port translate = TranslateInput
         Computation = (translate.map?QoS ->
                        MappingStrategy ->
                        translate.ret_map!MappedQoS ->
                        Computation)
                     []
                     Success

   Component QoSNeg(NegotiationStrategy : Computation;
                     icas : 1..; ecas : 1..) =
      Port translate_1..ecas =   TranslateOutput
      Port intralevel_1..icas = IntralevelInput
      Port interlevel_1..ecas = InterlevelOutput
         Computation = ([] j : 1..icas @
                         (intralevel_j.request?QoSnFlowSpec ->
                          NegotiationStrategy ->
                          (; i:1..ecas
                           ( ignoreInternalProvider
                             |~|
                             (translate_i.map!QoS ->
```

```
                                translate i.ret_map?MappedQoS ->
                                interlevel i.admit!MappedQoS ->
                                interlevel i.ret_admit?ResId
                              )
                            )
                          )
                          computeResIds ->
                          intralevel j.ret_request!ResIdGlobal ->
                          Computation
                        )
                        []
                        ([] j : 1..icas @
                         (intralevel j.confirm?ResIdGlobal ->
                          NegotiationStrategy ->
                          (; i:1..ecas
                           ( ignoreInternalProvider
                             |~|
                             (
                              translateResIdGlobal ->
                              interlevel i.commit!ResId ->
                              interlevel i.ret_commit?result
                             )
                           )
                          )
                          computeResults ->
                          intralevel j.ret_confirm!resultGlobal ->
                          Computation
                        )

        Connector NegToMapper
           Role mapper = TranslateInput
           Role neg = TranslateOutput
           Glue = (neg.map!QoS ->
                   mapper.map?QoS ->
                   mapper.ret_map!QoS ->
                   neg.ret_map?QoS ->
                   Glue)
                   []
                   Success

        Connector Intralevel
           Role neg = IntralevelInput
           Role ca = IntralevelOutput
           Glue = (ca.request!QoSnFlowSpec ->
                   neg.request?QoSnFlowSpec->
                   Glue)
                   []
                   Success

        Constraints
         – Existe um e apenas um componente QoSNeg instanciado
         – Para cada componente AdmCtrl instanciado, há um
         conector Intralevel instanciado
         – O único componente QoSNeg está associado pelas portas
         intralevel a todos os icas conectores Intralevel pelo
         papel neg
         – Todos os icas componentes AdmCtrl estão associados a um
         único conector Intralevel pelo papel ca
         – O único componente QosNeg está associado pelas portas
         translate a todos os ecas conectores NegToMapper pelo
         papel neg
```

```
        –Todos os ecas componentes QoSMapper estão associados
        pela porta translate a um conector NegToMapper pelo papel
        mapper
End Style
```

Figura 3.10 – Estilo *CentralizedNQoS* para Negociação de QoS

O estilo *DistributedNQoS*, graficamente apresentado na Figura 3.11, é utilizado na descrição de subsistemas onde os mecanismos de negociação de QoS são delegados a mais de um negociador de QoS. Vários tipos de interfaces, componentes e conectores são semelhantes aos apresentados no estilo *CentralizedNQoS*, portanto só serão retomados a seguir os tipos ausentes ou alterados com relação ao anterior.

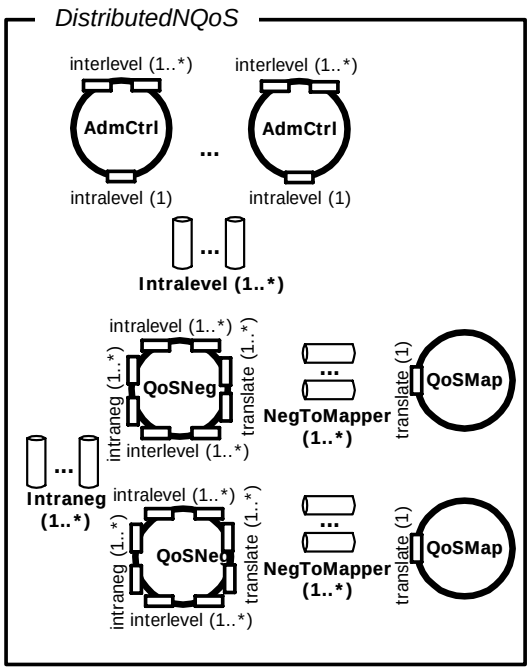

Figura 3.11 – Representação gráfica do estilo *DistributedNQoS*

A Figura 3.12 lista a especificação do estilo *DistributedNQoS*. O tipo de interface *IntranegInOut* descreve a porta *intraneg* dos *Negociadores de QoS*. A função dessa porta é permitir que um negociador de QoS negocie a responsabilidade sobre a admissão de um fluxo com outros negociadores dentro do subsistema. Uma restrição importante do estilo é que deve haver um caminho ligando quaisquer dois negociadores dentro do subsistema, por meio da porta *intraneg*.

A computação do componente *QoSNeg* no estilo *DistributedNQoS* está estendida com relação à apresentada em *CentralizedNQoS*. Isso porque em um ambiente distribuído, os negociadores de QoS podem negociar requisições de admissão de fluxo entre si.

```
Style DistributedNQoS

   Interface Type IntralevelInput =
                    (request?QoSnFlowSpec ->
                    ret_request!ResId ->
                    IntralevelInput)
                    []
                    (confirm?ResId ->
                    ret_confirm!result ->
                    IntralevelInput)
                    []
                    Success

   Interface Type IntralevelOutput =
                    (request!QoSnFlowSpec ->
                    ret_request?ResId ->
                    IntralevelOutput)
                    |~|
                    (confirm!ResId ->
                    ret_confirm?result ->
                    IntralevelOutput)
                    |~|
                    Success

   Interface Type InterlevelInput =
                    (admit?QoSnFlowSpec ->
                    ret_admit!ResId ->
                    InterlevelInput)
                    []
                    (commit?ResId ->
                    ret_commit!result ->
                    InterlevelInput)
                    []
                    Success

   Interface Type InterlevelOutput =
                    (admit!QoSnFlowSpec ->
                    ret_admit?ResId ->
                    InterlevelOutput)
                    |~|
                    (commit!ResId ->
                    ret_commit?result ->
                    InterlevelOutput)
                    |~|
                    Success

   Interface Type TranslateInput =
                    (map?QoS ->
                    ret_map!MappedQoS ->
                    TranslateInput)
                    []
                    Success
```

```
      Interface Type TranslateOutput =
                        (map!QoS ->
                        ret_map?MappedQoS ->
                        TranslateOutput)
                        |~|
                        Success

      Interface Type IntranegInOut =
                        (InterlevelInput
                        [] InterlevelOutput)
                        |~|
                        Success

      Component AdmCtrl(enegs : 1..)
         Port intralevel = IntralevelOutput
         Port interlevel_1..enegs = InterlevelInput
         Computation = ([] i : 1..enegs @
                         (interlevel_i.admit?QoSnFlowSpec ->
                          intralevel.request!QoSnFlowSpec ->
                          intralevel.ret_request?ResId          ->
                    interlevel_i.ret_admit!ResId ->
                          Computation))
                        []
                        ([] i : i..enegs @
                         (interlevel_i.commit?ResId ->
                          intralevel.confirm!ResId ->
                          intralevel.ret_confirm?result_cnf ->
                          interlevel_i.ret_commit!result_cmt ->
                          Computation))
                        []
                        Success

   Component QoSMapper (MappingStrategy : Computation) =
      Port translate = TranslateInput
         Computation = (translate.map?QoS ->
                        MappingStrategy                         ->
                    translate.ret_map!MappedQoS ->
                        Computation)
                    []
                    Success

   Component QoSNeg(NegotiationStrategy : Computation;
                    inegs : 1..; icas : 1..; ecas : 1..) =
      Port translate_1..ecas =   TranslateOutput
      Port intraneg_1..inegs = IntranegInOut
      Port intralevel_1..icas = IntralevelInput
      Port interlevel_1..ecas = InterlevelOutput
         Computation = (
                         ([] j : 1..icas @
                           (intralevel_j.request?QoSnFlowSpec ->
                            InternalRequestMechanisms ->
                            intralevel_j.ret_request!ResIdGlobal->
                            Computation)
                         )
                         []
                         ([] k : 1..inegs @
                           (intraneg_k.request?QoSnFlowSpec ->
                            InternalRequestMechanisms ->
                            intraneg_k.ret_request!ResIdGlobal ->
                            Computation)
```

```
 )
)
where
{InternalRequestMechanisms =
 NegotiationStrategy ->
 (
    (
      (; i : 1..ecas @
        ( ignoreInternalProvider
          |~|
          (translate i.map!QoS ->
           translate i.ret_map?MappedQoS->
           interlevel i.admit!MappedQoS ->
           interlevel i.ret_admit?ResId)
        )
      )
      -> ComputeResIds
    )
    |~|
    (
     (|~| t : 1..inegs @
      (intraneg t.request!QoSnFlowSpec->
      intraneg t.ret_request?ResIdGlobal)
     )
    )
 )
}

[]

(
 ([] j : 1..icas @
   (intralevel j.confirm?ResId ->
    InternalConfirmMechanisms ->
    intralevel j.ret_confirm!result ->
    Computation)
 )
 []
 ([] k : 1..inegs @
   (intraneg k.confirm?QoSnFlowSpec ->
    InternalConfirmMechanisms ->
    intraneg k.ret_confirm!ResIdGlobal ->
    Computation)
 )
)
where
{InternalConfirmMechanisms =
 NegotiationStrategy ->
 (
    (
      (; i : 1..ecas @
        ( ignoreInternalProvider
          |~|
          (interlevel i.commit!ResId ->
           interlevel i.ret_commit?result)
        )
      )
      -> ComputeResults
    )
    |~|
    (
```

```
                                (|~| t : 1..inegs @
                                  (intraneg t.confirm!ResId ->
                                  intraneg t.ret_confirm?result)
                                )
                              )
                            )
                          }

        Connector NegToMapper
           Role mapper = TranslateInput
           Role neg = TranslateOutput
           Glue = (neg.map!QoS ->
                  mapper.map?QoS ->
                  mapper.ret_map!QoS ->
                  neg.ret_map?QoS ->
                  Glue)
                  []
                  Success

        Connector Intralevel
           Role neg = IntralevelInput
           Role ca = IntralevelOutput
           Glue = (ca.request!QoSnFlowSpec ->
                   neg.request?QoSnFlowSpec ->
                   neg.ret_request!ResId ->
                   ca.ret_request?ResId
                   -> Glue )
                  []
                  (ca.confirm!ResId ->
                   neg.confirm?ResId ->
                   neg.ret_confirm!result ->
                   neg.ret_confirm?result ->
                   Glue )
                  []
                  Success

        Connector Intraneg
           Role neg1 = (IntralevelInput [] IntralevelOutput) ->
                       neg1
                       [] Success
           Role neg2 = (IntralevelInput [] IntralevelOutput) ->
                       neg2
                       [] Success
           Glue = (neg1.request!QoSnFlowSpec ->
                   neg2.request?QoSnFlowSpec ->
                   neg2.ret_request!ResId ->
                   neg1.ret_request?ResId ->
                   Glue)
                  []
                  (neg2.request!QoSnFlowSpec ->
                   neg1.request?QoSnFlowSpec ->
                   neg1.ret_request!ResId ->
                   neg2.ret_request?ResId ->
                   Glue)
                  []
                  (neg1.confirm!ResId ->
                   neg2.confirm?ResId ->
                   neg2.ret_confirm!result ->
                   neg1.ret_confirm?result ->
                   Glue)
                  []
```

```
                    (neg2.confirm!ResId ->
                     neg1.confirm?ResId ->
                     neg1.ret_confirm!result ->
                     neg2.ret_confirm?result ->
                     Glue)

        Constraints
-   Há pelo menos dois componentes QoSNeg instanciados
-   Para cada componente AdmCtrl instanciado, há um componente
    QosMapper e um conector NegToMapper instanciados
-   Para cada componente AdmCtrl instanciado, há um conector
    Intralevel instanciado
-   Todo componente QosNeg está associado pela porta translate a um
    conector NegToMapper pelo papel neg
-   Todo componente QoSMapper está associado pela porta translate a
    um conector NegToMapper pelo papel mapper
-   Todo componente AdmCtrl está associado pela porta intralevel a
    um conector Intralevel pelo papel ca
-   Todo conector Intralevel está associado pelo papel neg a pelo
    menos um componente QoSNeg pela porta intralevel
-   Todo componente QoSNeg está associado pela porta internegs a
    pelo menos um conector NegToNeg pelos papéis neg1 ou neg2
-   Todo conector NegToNeg está associado pelas portas neg1 e neg2
    a dois componentes QoSNeg pela porta internegs de cada um
-   Para quaisquer dois componentes QoSNeg, há um caminho de
    conectores Intraneg e componentes QoSNeg ligando os dois
End Style
```

Figura 3.12 – Estilo DistributedNQoS para Negociação de QoS

Uma vez que cada subsistema de negociação de QoS pode ser descrito separadamente usando os três estilos anteriores (*LowestNQoS*, *CentralizedNQoS* e *DistributedNQoS*), um outro estilo é necessário para realizar a ligação entre os subsistemas. O estilo *HierarchyNQoS*, descrito na Figura 3.13, define o conector *Interlevel*, que regula a interação entre negociadores de QoS (*InterlevelOutput*) e controladores de admissão (*InterlevelInput*) de subsistemas diferentes.

A definição do estilo *HierarchyNQoS* se restringe apenas em formalizar os conectores que ligam subsistemas diferentes, o que é útil para a definição de configurações de subsistemas cooperando entre si para prover o meta serviço de negociação de QoS, mas não tem utilidade na descrição de *frameworks* arquiteturais por meio de estilos. Na prática, essa forma particular de definir o estilo *HierarchyNQoS* é resultado da limitação de ADLs em fornecerem estilos recursivos, o que seria vital para a especificação de um estilo voltado para descrever o aninhamento de subsistemas.

```
Style HierarchyNQoS

   Interface Type InterlevelInput =
                  (admit?QoSnFlowSpec ->
                  ret_admit!ResId ->
                  InterlevelInput)
                  []
                  (commit?ResId ->
                  ret_commit!result ->
                  InterlevelInput)
                  []
                  Success

   Interface Type InterlevelOutput =
                  (admit!QoSnFlowSpec ->
                  ret_admit?ResId ->
                  InterlevelOutput)
                  |~|
                  (commit!ResId ->
                  ret_commit?result ->
                  InterlevelOutput)
                  |~|
                  Success

   Connector Interlevel
      Role neg = InterlevelOutput
      Role ca = InterlevelInput
      Glue = (neg.admit!QoSnFlowSpec ->
              ca.admit?QoSnFlowSpec ->
              ca.ret_admit!ResId ->
              neg.ret_admit?ResId ->
              Glue )
             []
             (neg.commit!ResId ->
              ca.commit?ResId ->
              ca.ret_commit!result ->
              neg.ret_commit?result ->
              Glue )
             []
             Success
   Constraints

End Style
```

Figura 3.13 – Estilo HierarchyNQoS para a Negociação de QoS

3.3. Estilos Arquiteturais para Sintonização de QoS

Para facilitar o entendimento, a especificação em Wright dos mecanismos de sintonização de QoS foi feita o mais próxima possível da negociação, conforme apresentada na Seção 3.2. Para tanto, ambos partilham a mesma nomenclatura para tipos de interface, portas e conectores. No mais, a sintonização também é descrita por meio de quatro diferentes estilos: *LowestTQoS*, usado em subsistemas que atuam diretamente sobre árvores de recursos virtuais primitivas;

CentralizedTQoS, usado para descrever subsistemas em que todos os mecanismos de sintonização são de responsabilidade de um único componente; *DistributedTQoS*, descrevendo subsistemas mais complexos, em que os mecanismos de sintonização são de responsabilidade de vários componentes interagindo entre si; e *HierarchyTQoS*, que especifica a ligação entre subsistemas.

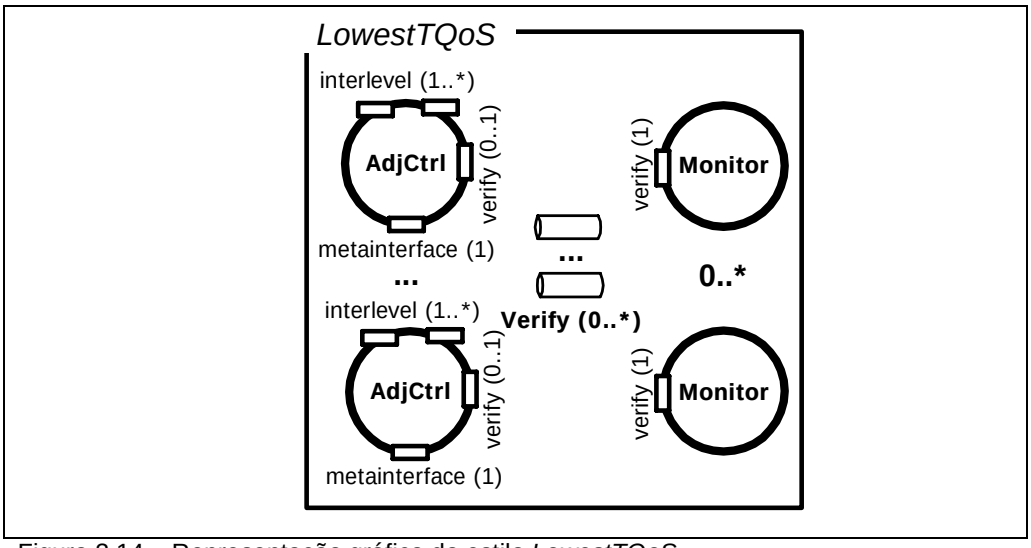

Figura 3.14 – Representação gráfica do estilo *LowestTQoS*

A Figura 3.14 representa graficamente o estilo *LowestTQoS* descrito textualmente na Figura 3.15. O tipo de componente *AdjCtrl* recebe dois parâmetros na instanciação: uma estratégia de ajuste (*AdjStrategy*) e o número de sintonizadores de QoS externos a que o componente se liga (*etuners*). A porta *interlevel* tem a cardinalidade igual ao número de sintonizadores associados, ou seja, é equivalente ao parâmetro *etuners*.

O tipo de componente Monitor representa o elemento da sintonização de QoS de mesmo nome. Ele atua nos subsistemas primitivos efetuando cálculos estatísticos que são usado pelos controladores de ajuste primitivos para verificarem a iminência de violações de contratos de serviço. A porta *verify* é usada para ligar ambos os tipos de componente.

```
Style LowestTQoS
   Interface Type InterlevelInput =
                  (adjust?QoS ->
                   ret_adjust!result ->
                   InterlevelInput)
                  []
                  (alert!violation ->
                   InterlevelInput)
                  []
                  Success
```

```
Interface Type InterlevelOutput =
               (adjust!QoS ->
                ret_adjust?result ->
                InterlevelOutput)
               []
               (alert?violation ->
                InterlevelOutput)
               []
               Success

Interface Type VerifyMonitor =
               (getStatistics ->
                ret_getStatistics!stats ->
                VerifyMonitor)
               []
               Success

Interface Type VerifyAdjCtrl =
               (getStatistics ->
                ret_getStatistics?stats ->
                VerifyAdjCtrl)
               []
               Success

Interface Type MetaInterfaceInput =
               tuneResources?QoS ->
               ret_tuneResources!result ->
               MetaInterfaceInput
               []
               Success

Interface Type MetaInterfaceOutput =
               tuneResources!QoS ->
               ret_tuneResources?result ->
               MetaInterfaceOutput
               []
               Success

Component AdjCrl (AdjStrategy : Computation; etuners: 1..)
   Port interlevel etuners = InterlevelInput
   Port verify = VerifyAdjCtrl
   Port METAINTERFACE = MetainterfaceOutput
   Computation = ([] i : 1..etuners @
                  interlevel i.adjust?QoS ->
                  AdjStrategy ->
                  MetaInterface.tuneResources!QoS ->
                  MetaInterface.ret_tuneResources?Result ->
                  interlevel i.ret_adjust!result ->
                  Computation)
                 []
                 (verify.getStatistics ->
                  verify.ret_getStatistics?stats ->
                  testViolation ->
                  (Computation
                   |~|
                   (MetaInterface.tuneResources!QoS ->
                    MetaInterface.ret_tuneResources?Result ->
                    Computation)
                   |~|
                   (|~| k : 1..etuners @
```

```
                        alert k!violation ->
                        Computation)
                      )
                    )
                    []
                    Success

   Component Monitor (MonitoringStrategy : Computation)
      Port verify = VerifyMonitor
      Computation = (verify.getStatistics ->
                     MonitoringStrategy ->
                     verify.ret_getStatistics!stats ->
                     Computation)
                    []
                    Success

   Connector Verify
      Role Monitor  = VerifyMonitor
      Role AdjCtrl = VerifyAdjCtrl
      Glue = (AdjCtrl.getStatistics ->
              Monitor.getStatistics ->
              Monitor.ret_getStatistics!stats ->
              AdjCtrl.ret_getStatistics?stats ->
              Glue)
             []
             (Monitor.getStatistics ->
              AdjCtrl.getStatistics ->
              Monitor.ret_getStatistics!stats ->
              AdjCtrl.ret_getStatistics?stats ->
              Glue)
             []
              Success
   Constraints
-  Há pelo menos um componente AdjCtrl instanciado;
-  Para cada componente Monitor instanciado, há um conector Verify
   instanciado;
-  Todo componente Monitor instanciado está ligado pela porta
   verify a um único conector Verify instanciado, através do papel
   Monitor;
-  Todo conector Verify instanciado está ligado pelo papel AdjCtrl
   a um único componente AdjCtrl instaciado, através da porta
   verify;
-  Todo componente AdjCtrl está ligado pela porta Verify a nenhum
   ou apenas um conector Verify pelo papel AdjCtrl.
EndStyle
```

Figura 3.15 – Estilo *LowestTQoS* para Sintonização de QoS

A porta *metainterface* deve ser ligada ao serviço principal, atuando na reorquestração de recursos. Essa última porta requer um maior detalhamento a fim de descrever como o meta serviço de sintonização atua sobre o serviço principal para manter o nível contratado de serviço, mas isso é deixado como trabalho futuro.

A Figura 3.16 descreve graficamente o estilo *CentralizedTQoS*, cuja especificação em Wright é apresentada na Figura 3.17. Os tipos de interface *VerifyInput* e *VerifyOutput* descrevem como se dá a verificação periódica do

fluxo, seja feita pelos sintonizadores de QoS através de seu monitor associado, ou ainda a partir do repasse dos cálculos estatísticos iniciado pelo monitor (*upcalls*), de forma que o conector *Verify* regula o papel de cada um desses componentes.

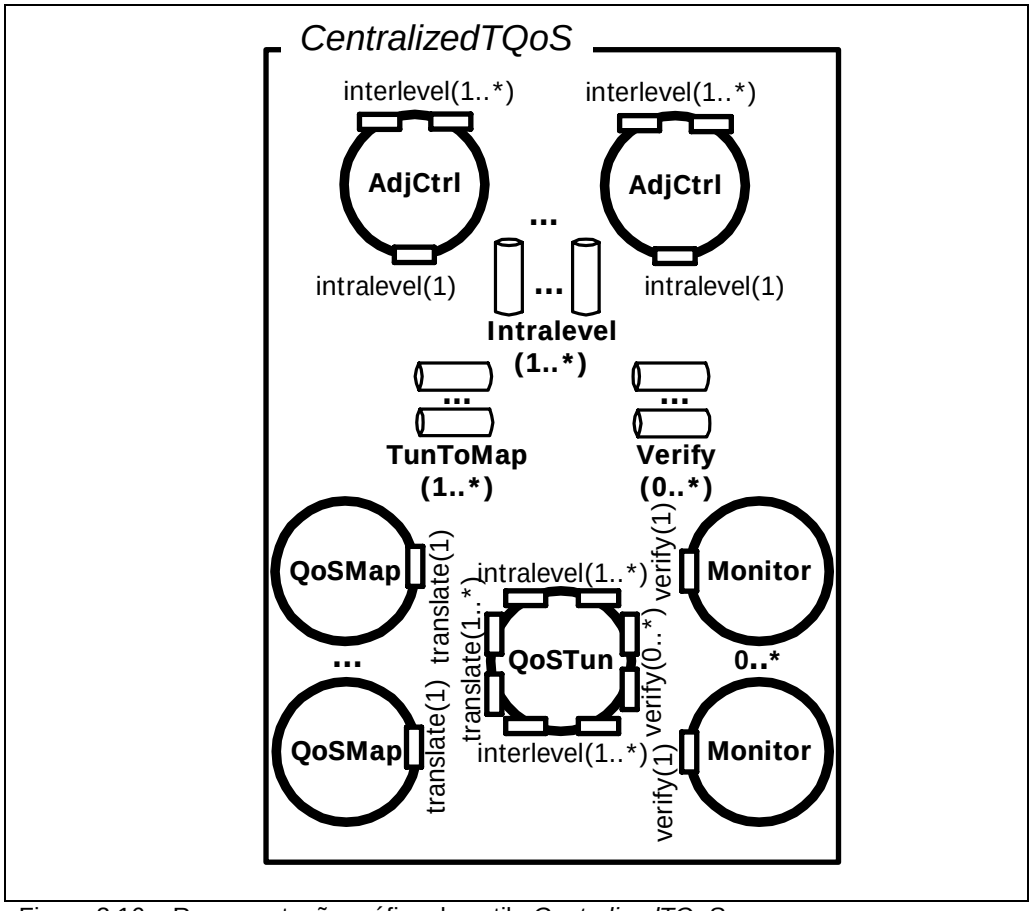

Figura 3.16 – Representação gráfica do estilo *CentralizedTQoS*

Finalmente, os tipos de interface *IntraleveInput* e *IntralevelOutput* descrevem como é feita a associação entre controladores de ajuste (*IntralevelOutput*) e sintonizadores de QoS (*IntralevelInput*) presentes em um mesmo subsistema. O conector *Intralevel* especifica o papel desses componentes na associação, determinando que toda requisição de sintonização recebida pelo controlador de ajuste é repassada ao sintonizador associado para que, enfim, seja retornado o resultado desse processo de manutenção do nível contratado de serviço. Esse conector também leva em conta o envio de alertas a partir do sintonizador de QoS para um controlador de ajuste associado, com a intenção de informar um agente de mais alto nível sobre uma violação no contrato de serviço. Os tipos de interface *InterlevelInput* e *InterlevelOutput* descrevem as regras de

associação entre sintonizadores de QoS (*InterlevelOutput*) e controladores de ajuste de subsistemas internos (*InterlevelInput*), além de incluirem o mecanismo de alerta em caso de violação da QoS (conforme previamente descrito).

O tipo de componente *AdjCtrl* descreve a interface e comportamento dos controladores de ajuste. Ele é parametrizável por um inteiro *etuners* que identifica o número de sintonizadores de QoS associados àquele componente (ou seja, o número de portas *interlevel*). Assim, cada porta *interlevel* de uma instância de um *AdjCtrl* é associada a uma porta *interlevel* de uma instância de um componente *QoSTuner* de um subsistema externo. No mais, quando da instanciação, a porta *intralevel* de cada componente *AdjCtrl* deve ser ligada a uma porta de mesmo nome em um único sintonizador de QoS, a quem são repassadas todas as suas requisições de sintonização recebidas.

O monitor é representado pelo tipo de componente *Monitor*. Ele é parametrizável por uma estratégia de monitoração (*MonitoringStrategy*), que especifica os cálculos estatísticos que devem ser aplicados sobre o fluxo para que se avalie a iminência de violação. Instâncias de componentes *Monitor* devem ser ligada a sintonizadores de QoS por meio de suas respectivas portas *verify*.

O tipo de componente *QoSTuner* representa o sintonizador de QoS. Os parâmetros *icas* e *ecas* são usados, respectivamente, para definir o número de controladores de ajuste associados em seu subsistema e em subsistemas externos, o que implica na cardinalidade da porta *intralevel* ser igual a *icas* e a cardinalidade da porta *interlevel* ser igual a *ecas*.

```
Style CentralizedTQoS
   Interface Type InterlevelInput =
                  (adjust?QoS ->
                   ret_adjust!result ->
                   InterlevelInput)
                  []
                  (alert!violation ->
                   InterlevelInput)
                  []
                  Success

   Interface Type InterlevelOutput =
                  (adjust!QoS ->
                   ret_adjust?result ->
                   InterlevelOutput)
                  []
                  (alert?violation ->
                   InterlevelOutput)
                  []
                  Success
```

```
Interface Type VerifyMonitor =
               (getStatistics ->
                ret_getStatistics!stats ->
                VerifyMonitor)
               []
               Success

Interface Type VerifyQoSTuner =
               (getStatistics ->
                ret_getStatistics?stats ->
                VerifyQoSTuner)
               []
               Success

Interface Type IntralevelInput =
               (tune?QoS ->
                ret_tune!result ->
                IntralevelInput)
               []
               (alert!violation ->
                IntralevelInput)
               []
               Success

Interface Type IntralevelOutput =
               (tune!QoS ->
                ret_tune?result ->
                IntralevelOutput)
               []
               (alert?violation ->
                IntralevelOutput)
               []
               Success

Interface Type TranslateInput =
               (map?QoS ->
                ret_map!MappedQoS ->
                TranslateInput)
               []
               Success

Interface Type TranslateOutput =
               (map!QoS ->
                ret_map?MappedQoS ->
                TranslateOutput)
               |~|
               Success

Component QoSMapper (MappingStrategy : Computation) =
   Port translate = TranslateInput
   Computation = (translate.map?QoS ->
                  MappingStrategy ->
                  translate.ret_map!MappedQoS ->
                  Computation)
                 []
                 Success

Component AdjCtrl (etuners: 1..)
   Port interlevel etuners = InterlevelInput
   Port intralevel = IntralevelOutput
   Computation = ([] i : 1..etuners @
```

```
                    interlevel_i.adjust?QoS ->
                    intralevel.tune!QoS ->
                    intralevel.ret_tune?result ->
                    interlevel_i.ret_adjust!result ->
                    Computation)
                   []
                    (intralevel.alert?violation ->
                     (|~| j : 1..etuners @
                      interlevel_j.alert!violation) ->
                     Computation)
                   []
                    Success

Component Monitor (MonitoringStrategy : Computation)
   Port verify = VerifyMonitor
   Computation = verify.getStatistics ->
                 MonitoringStrategy ->
                 verify.ret_getStatistics!stats ->
                 Computation
                 []
                 Success

Component QoSTuner (icas: 1..; imons: 0..; ecas: 1..)
   Port verify_imons = VerifyQoSTuner
   Port translate_ecas = TranslateOutput
   Port intralevel_icas = IntralevelInput
   Port interlevel_ecas = InterlevelOutput
   Computation = (
                   ([] i : 1..icas @
                    intralevel_i.adjust?QoS ->
                    TuningMechanisms ->
                    intralevel_i.ret_adjust!result ->
                    Computation)
                   []
                   ([] k : 1..imons @
                    verify_k.getStatistics ->
                    verify_k.ret_getStatistics?stats ->
                    testViolation ->
                    (Computation
                     |~|
                     TuningMechanisms ->
                     Computation)
                   )
                  )
                  where
                  {TuningMechanisms =
                  (
                   ( ; j : 1 .. ecas @
                    (translate_j.map!QoS
                     translate_j.ret_map?MappedQoS
                     interlevel_j.adjust!MappedQoS ->
                     interlevel_j.ret_adjust?result
                     |~|
                     ignoreProvider)
                   )
                   |~|
                   (|~| m : 1 .. icas @
                    intralevel_m.alert!violation)
                  )
                 }
```

```
Connector  Verify
   Role Monitor  = VerifyMonitor
   Role QoSTuner = VerifyQoSTuner
   Glue = QoSTuner.getStatistics ->
          Monitor.getStatistics ->
          Monitor.ret_getStatistics!stats ->
          QoSTuner.ret_getStatistics?stats ->
          Glue
          []
          Monitor.getStatistics ->
          QoSTuner.getStatistics ->
          Monitor.ret_getStatistics!stats ->
          QoSTuner.ret_getStatistics?stats ->
          Glue
          []
          Success

Connector Intralevel
   Role AdjCtrl = IntralevelOutput
   Role QoSTuner = IntralevelInput
   Glue = (AdjCtrl.adjust!QoS ->
           QoSTuner.adjust?QoS ->
           QoSTuner.ret_adjust!result ->
           AdjCtrl.ret_adjust?result ->
           Glue)
          []
          (QoSTuner.alert!violation ->
           AdjCtrl.alert?violation)
          []
          Success

Connector TunToMapper
   Role mapper = TranslateInput
   Role tun = TranslateOutput
   Glue = (tun.map!QoS ->
           mapper.map?QoS ->
           mapper.ret_map!mappedQoS ->
           tun.ret_map?mappedQoS ->
           Glue)
          []
          Success

Constraints
-  Há um único componente QoSTuner instanciado
-  Há um ou mais componentes QoSMap instanciados
-  Para cada componente QoSMap instanciado há um conector TunToMap
   instanciado
-  Todo conector TunToMap está ligado pelo papel mapper a um
   componente QoSMap pela porta translate
-  Todo conector TunToMap está ligado pelo papel tun ao único
   componente QoSTuner pela porta translate
-  Há zero ou mais componentes Monitor instanciados
-  Para cada componente Monitor instanciado, há um conector Verify
   instanciado
-  Todo conector Verify está ligado pelo papel Monitor a um
   componente Monitor pela porta verify
-  Todo conector Verify está ligado pelo papel QoSTuner ao único
   componente QoSTuner pela porta verify
-  Há um ou mais componentes AdjCtrl instanciados
```

```
-  Para cada componente AdjCtrl instanciado há um conector
   Intralevel instanciado
-  Todo conector Intralevel está ligado pelo papel AdjCtrl a um
   componente AdjCtrl pela porta intralevel
-  Todo conector Intralevel está ligado pelo papel QoSTuner a um
   componente QoSTuner pela porta intralevel
End Style
```

Figura 3.17 – Estilo *CentralizedTQoS* para Sintonização de QoS

O estilo *DistributedTQoS* é apresentado graficamente na Figura 3.18 e textualmente na Figura 3.19. Com relação ao estilo *CentralizedTQoS*, as principais diferenças são as inclusões do tipo de interface *IntratunerInOut*, do conector *Intratuner* e da porta *intratuner* no tipo de componente *QoSTuner*. Além disso, a computação do componente *QoSTuner* passa a levar em conta, também, a interação com outros sintonizadores presentes no subsistema, seja para receber requisições, seja para repassá-las.

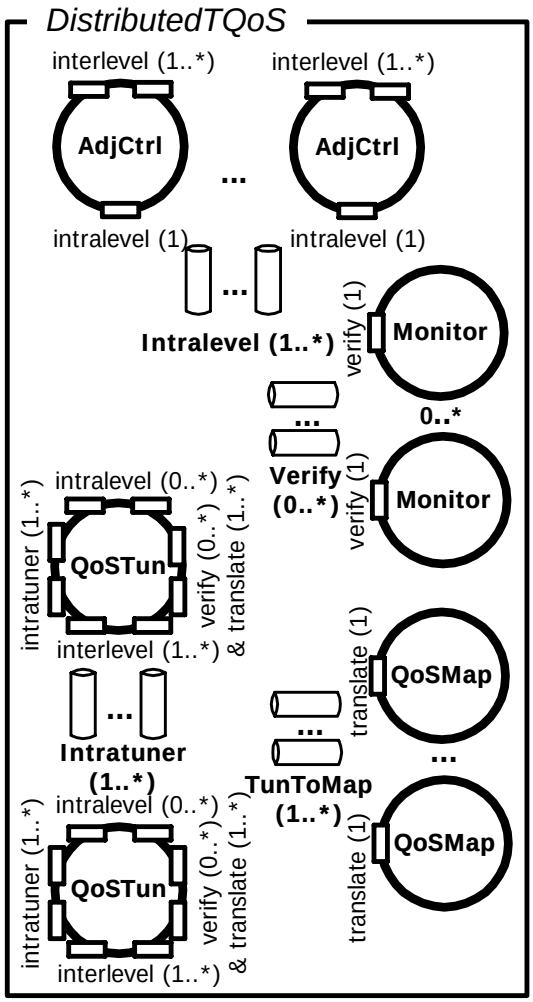

Figura 3.18 – Representação gráfica do estilo *DistributedTQoS*

O tipo de interface *IntratunerInOut* regula como é feita a troca de mensagens entre sintonizadores de QoS em um mesmo subsistema, no caso da sintonização ser distribuída. O conector *Intratuner* define o papel de cada par de sintonizadores de QoS, associados através da porta *intratuner*.

```
Style DistributedTQoS

   Interface Type InterlevelInput =
                  (adjust?QoS ->
                   ret_adjust!result ->
                   InterlevelInput)
                  []
                  (alert!violation ->
                   InterlevelInput)
                  []
                  Success

   Interface Type InterlevelOutput =
                  (adjust!QoS ->
                   ret_adjust?result ->
                   InterlevelOutput)
                  []
                  (alert?violation ->
                   InterlevelOutput)
                  []
                  Success

   Interface Type VerifyMonitor =
                  (getStatistics ->
                   ret_getStatistics!stats ->
                   VerifyMonitor)
                  []
                  Success

   Interface Type VerifyQoSTuner =
                  (getStatistics ->
                   ret_getStatistics?stats ->
                   VerifyQoSTuner)
                  []
                  Success

   Interface Type IntralevelInput =
                  (tune?QoS ->
                   ret_tune!result ->
                   IntralevelInput)
                  []
                  (alert!violation ->
                   IntralevelInput)
                  []
                  Success

   Interface Type IntralevelOutput =
                  (tune!QoS ->
                   ret_tune?result ->
                   IntralevelOutput)
                  []
                  (alert?violation ->
```

```
                    IntralevelOutput)
                   []
                   Success

Interface Type TranslateInput =
                   (map?QoS ->
                    ret_map!MappedQoS ->
                    TranslateInput)
                   []
                   Success

Interface Type TranslateOutput =
                   (map!QoS ->
                    ret_map?MappedQoS ->
                    TranslateOutput)
                   |~|
                   Success

Interface Type IntraTunerInOut =
                   (IntralevelInput
                   |~|
                   IntralevelOutput)
                   []
                   Success

Component AdjCtrl (etuners: 1..)
   Port interlevel etuners = InterlevelInput
   Port intralevel = IntralevelOutput
   Computation = ([] i : 1..etuners @
                    interlevel_i.adjust?QoS ->
                    intralevel.tune!QoS ->
                    intralevel.ret_tune?result ->
                    interlevel_i.ret_adjust!result ->
                    Computation)
                   []
                    (intralevel.alert?violation ->
                     (|~| j : 1..etuners @
                      interlevel j.alert!violation) ->
                     Computation)
                   []
                    Success

Component Monitor (MonitoringStrategy : Computation)
   Port verify = VerifyMonitor
   Computation = verify.getStatistics ->
                  MonitoringStrategy ->
                  verify.ret_getStatistics!stats ->
                  Computation
                  []
                  Success

Component QoSTuner (icas: 1..; imons: 0..; ituners : 1.. ;
                     ecas: 1..)
   Port verify imons = VerifyQoSTuner
   Port translate ecas = TranslateOutput
   Port intralevel icas = IntralevelInput
   Port interlevel ecas = InterlevelOutput
   Port intratuner ituners = IntratunerInOut
   Computation = (
                    ([] i : 1..icas @
                     intralevel i.adjust?QoS ->
```

```
                        TuningMechanisms ->
                        intralevel i.ret_adjust!result ->
                        Computation)
                       []
                       ([] k : 1..imons @
                        verify k.getStatistics ->
                        verify k.ret_getStatistics?stats ->
                        testViolation ->
                        (Computation
                         |~|
                         TuningMechanisms ->
                         Computation)
                       )
                       []
                       ([] p : 1..ituners @
                        intratuner p.tune?QoS ->
                        TuningMechanisms ->
                        Intratuner p.ret_tune!result)
                      )
                      where
                      {TuningMechanisms =
                      (
                       ( ; j : 1 .. ecas @
                        (translate j.map!QoS
                         translate j.ret_map?MappedQoS
                         interlevelj.adjust!MappedQoS ->
                         interlevelj.ret_adjust?result
                         |~|
                         ignoreProvider)
                       )
                       |~|
                       (|~| m : 1 .. icas @
                        intralevel m.alert!violation)
                      )
                       |~|
                       ( [] n : 1..ituners @
                         intratuner n.tune!QoS ->
                         intratuner n.ret_tune?result )
                     }

   Connector  Verify
      Role Monitor  = VerifyMonitor
      Role QoSTuner = VerifyQoSTuner
      Glue = QoSTuner.getStatistics ->
             Monitor.getStatistics ->
             Monitor.ret_getStatistics!stats ->
             QoSTuner.ret_getStatistics?stats ->
             Glue
             []
             Monitor.getStatistics ->
             QoSTuner.getStatistics ->
             Monitor.ret_getStatistics!stats ->
             QoSTuner.ret_getStatistics?stats ->
             Glue
             []
             Success

   Connector Intralevel
      Role AdjCtrl = IntralevelOutput
      Role QoSTuner = IntralevelInput
      Glue = (AdjCtrl.adjust!QoS ->
```

```
                QoSTuner.adjust?QoS ->
                QoSTuner.ret_adjust!result ->
                AdjCtrl.ret_adjust?result ->
                Glue)
               []
               (QoSTuner.alert!violation ->
                AdjCtrl.alert?violation)
               []
               Success

   Connector TunToMapper
      Role mapper = TranslateInput
      Role tun = TranslateOutput
      Glue = (tun.map!QoS ->
               mapper.map?QoS ->
               mapper.ret_map!mappedQoS ->
               tun.ret_map?mappedQoS ->
               Glue)
              []
              Success

   Connector Intratuner
      Role tuner1 = IntraTunerInOut
      Role tuner2 = IntraTunerInOut
      Glue = (tuner1.tune!QoS ->
               tuner2.tune?QoS->
               tuner2.ret_tune!result ->
               tuner1.ret_tune?result ->
               Glue)
              []
              (tuner2.tune!QoS ->
               tuner1.tune?QoS ->
               tuner1.ret_tune!result ->
               tuner2.ret_tune?result ->
               Glue)
              []
              Success
```

Constraints

- Há pelo menos dois componentes *QoSTuner* instanciados
- Há um ou mais componentes *QoSMap* instanciados
- Para cada componente *QoSMap* instanciado há um conector *TunToMap* instanciado
- Todo conector *TunToMap* está ligado pelo papel *mapper* a um componente *QoSMap* pela porta *translate*
- Todo conector *TunToMap* está ligado pelo papel *tun* a um componente *QoSTuner* pela porta *translate*
- Há zero ou mais componentes *Monitor* instanciados
- Para cada componente *Monitor* instanciado, há um conector *Verify* instanciado
- Todo conector *Verify* está ligado pelo papel *Monitor* a um componente *Monitor* pela porta *verify*
- Todo conector *Verify* está ligado pelo papel *QoSTuner* a um componente *QoSTuner* pela porta *verify*
- Há um ou mais componentes *AdjCtrl* instanciados
- Para cada componente *AdjCtrl* instanciado há um conector *Intralevel* instanciado
- Todo conector *Intralevel* está ligado pelo papel *AdjCtrl* a um componente *AdjCtrl* pela porta *intralevel*

```
-   Todo conector Intralevel está ligado pelo papel QoSTuner a um
    componente QoSTuner pela porta intralevel
-   Para quaisquer dois componentes QoSTuner, há um caminho de
    conectores Intratuner e componentes QoSTuner ligando ambos.
EndStyle
```

Figura 3.19 – Estilo *DistributedTQoS* para Sintonização de QoS

Para finalizar, a Figura 3.20 apresenta o estilo *HierarchyTQoS*. Seu uso é voltado para a ligação de diferentes subsistemas. Assim como o *HierarchyNQoS* (vide Seção 3.2), ele é definido com o intuito de facilitar a especificação de configurações arquiteturais de subsistemas cooperando entre si. Isso torna o estilo bastante simples e centralizado no tipo de conector *Interlevel*, que define o papel dos sintonizadores de QoS (*InterlevelOutput*) e controladores de ajuste (*InterlevelInput*) associados em diferentes subsistemas.

```
Style HierachyTQoS
   Interface Type InterlevelInput =
                  (adjust?QoS ->
                   ret_adjust!result ->
                   InterlevelInput)
                  []
                   (alert!violation ->
                   InterlevelInput)
                  []
                   Success

   Interface Type InterlevelOutput =
                  (adjust!QoS ->
                   ret_adjust?result ->
                   InterlevelOutput )
                  []
                  (alert?violation ->
                   InterlevelOutput)
                  []
                  Success

   Connector Interlevel
      Role QoSTuner = InterlevelOutput
      Role AdjCtrl = InterlevelInput
      Glue = (QoSTuner.adjust!QoS ->
              AdjCtrl.adjust?QoS ->
              AdjCtrl.ret_adjust!result ->
              QoSTuner.ret_adjust?result ->
              Glue)
            []
             (AdjCtrl.alert!violation ->
              QoSTuner.alert?violation ->
              Glue)
            []
             Success
   Constraints
EndStyle
```

Figura 3.20 – Estilo *HierarchyTQoS* para Sintonização de QoS