4
LindaQoS: LINguagem de Descrição de Arquitetura com QoS

Este capítulo apresenta a linguagem de domínio específico (DSL) LindaQoS (LINguagem de Descrição de Arquitetura com QoS), voltada para a descrição arquitetural dos meta serviços de negociação e sintonização de QoS. A Seção 4.1 introduz LindaQoS, formulando seus objetivos e requisitos. A seguir, na Seção 4.2, a linguagem é definida, incluindo seu EBNF (*Extended Backus Naur Form*). O Compilador LindaQoS é descrito na Seção 4.3 e, finalmente, os passos para a tradução para a ADL Wright são discutidos na Seção 4.4.

4.1
Objetivos e requisitos

A especificação de um sistema ou de uma família de sistemas utilizando os estilos arquiteturais em Wright pode se tornar muito difícil e tomar um tempo excessivo caso estejam envolvidos um grande número de recursos e de níveis de abstração. Em função disso, LindaQoS é uma linguagem de domínio específico (DSL) (Deursen et al., 2000) projetada para facilitar e agilizar a descrição arquitetural dos mecanismos de negociação e sintonização de QoS.

O projeto de LindaQoS foi feito tendo em mente que ela deveria se manter pequena, simples, concisa e fácil de aprender. Qualquer facilidade passível de ser incluída na linguagem, que violasse uma dessas diretrizes, foi renegada.

Esse enfoque, baseado no emprego de uma DSL, contorna a dificuldade de entendimento de especificações em Wright (que como mencionado também ocorre para outras ADLs) por meio da aplicação de uma notação (passível de tradução para Wright) mais próxima do nível do problema e mais simples. DSLs são definidas em (Deursen et al., 2000) como “linguagens de programação ou linguagens de especificação executável que oferecem, através de notações e abstrações apropriadas, expressivo poder focalizado, e usualmente restrito, a um domínio de problema particular”.

Ainda em (Deursen et al., 2000), são apresentados vários riscos e oportunidades para o uso de DSLs. É responsabilidade do projetista avaliar o equilíbrio sensato entre ambos para tirar maior proveito de seu uso. Os benefícios são:

– DSLs permitem que soluções sejam expressas no idioma e no nível da abstração do domínio do problema. Consequentemente, peritos no domínio, por si só, podem entender, validar, modificar e, até mesmo, desenvolver programas em DSLs;

– Programas com DSLs são concisos, auto-documentados em uma ampla extensão, e podem ser reusados para diferentes propósitos;

– DSLs ampliam a produtividade, disponibilidade, facilidade de manutenção e portabilidade;

– DSLs incorporam o conhecimento do domínio, e assim permitem a conservação e reuso desse conhecimento;

– DSLs permitem a validação e a otimização no nível do domínio;

– DSLs aperfeiçoam a capacidade de testes, seguindo certas abordagens.

As desvantagens do uso de uma DSL apontadas em (Deursen et al., 2000) são:

– Os custos de projeto, implementação e manutenção de uma DSL;

– Os custos de treinamento para usuários da DSL;

– A disponibilidade limitada de DSLs;

– A dificuldade de encontrar o escopo conveniente para uma DSL;

– A dificuldade de equilíbrio entre especificidade de domínio e construções de linguagem de programação de propósito geral;

– A perda potencial de eficiência quando comparada com software codificado manualmente.

No contexto dos mecanismos de provisão de QoS, o uso de uma DSL simplifica a utilização e entendimento por parte dos projetistas, que podem ficar mais voltados a abstrações específicas do domínio, como os conceitos de Negociadores e Sintonizadores de QoS e Controladores de Admissão e de Ajuste.

O projeto de LindaQoS leva em conta a implementação de um compilador como ferramenta para a tradução automática de sua especificação para uma ADL

(atualmente Wright), tomando todas as vantagens dessa descrição formal. Configurações arquiteturais (seguindo os estilos apresentados no Capítulo 3) são usadas para descrever os mecanismos recursivos de orquestração de recursos.

Outro compilador é proposto como trabalho futuro para servir de ferramenta para a tradução automática de especificações LindaQoS para uma linguagem de programação, contornando a ausência de ferramentas de geração de código em Wright (presente em outras ADLs, geralmente traduzindo para C/C++). Em outros termos, esse compilador pode ser usado no nível de implementação com o fim de traduzir arquivos fonte LindaQoS para uma especialização dos Frameworks Genéricos para Provisão de QoS (Gomes et al., 2001) (atualmente em JAVA). Tal especialização provê uma base para a instanciação dos mecanismos de *orquestração de recursos*. Dessa forma, o uso do Compilador LindaQoS permite que o projetista, através de uma única descrição concisa, obtenha automaticamente uma base para a implementação e uma descrição arquitetural completa, que pode ser usada para inferir propriedades.

4.2
Definição da linguagem

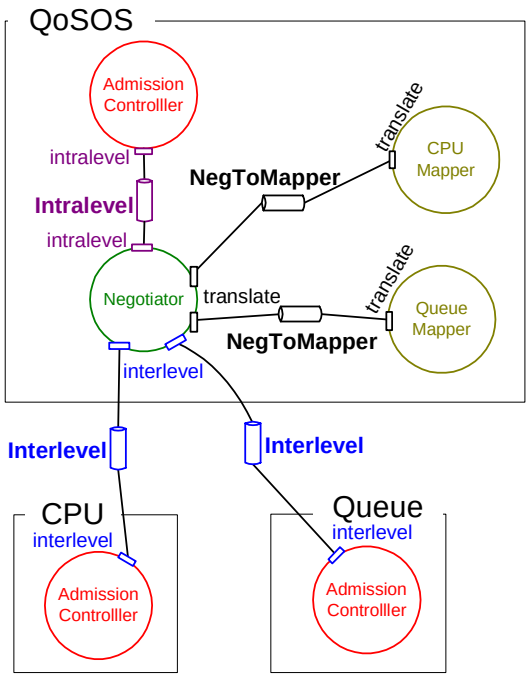
Um típico arquivo fonte em LindaQoS é dividido em várias seções. Há quatro tipos de seções: *System*; *Hierarchy*; *MtSystem*; e *MtHierarchy*. Em um arquivo fonte há uma única seção *Hierarchy* e uma única *MtHierarchy* (o prefixo Mt é originado de *maintenace*), que descrevem, respectivamente, hierarquias de subsistemas de negociação (descritos em seções *System*) e hierarquias de subsistemas de sintonização de QoS (descritos em seções *MtSystem*).

Conforme mencionado, uma instanciação completa do Framework de Negociação de QoS é composta de algumas seções *System* e uma única seção *Hierarchy*. Cada seção *System* descreve a instanciação de componentes em um nível de abstração particular (subseção *Instances*) e as ligações entre componentes daquele nível (subseção *Attachments*). A seção *Hierarchy*, por sua vez, descreve todas as relações entre os diferentes subsistemas. Há três tipos básicos de componentes de negociação em LindaQoS, usados na subseção *Instances*: o *AdmCtrl* (controlador de admissão); o *QoSNeg* (negociador de QoS); e o *QoSMap* (mapeador de QoS).

Para facilitar o entendimento, a Figura 4.1(a) ilustra a Arquitetura QoSOS (Moreno, 2002) de negociação, cujo objetivo é definir a provisão de QoS em sistemas operacionais. O arquivo fonte equivalente em LindaQoS é apresentado na Figura 4.1(b). Tal arquivo possui as seções de subsistema *QoSOS, CPU* e *LinkQueue* e a hierarquia *QoSOSNegFramework*. Cada um dos subsistemas *CPU* e *LinkQueue* instancia apenas um componente Controlador de Admissão de nome *AdmissionController* (isso os caracteriza no estilo *LowestNQoS*). O subsistema *QoSOS* centraliza a negociação de QoS no componente *Negotiator* (o caracterizando no estilo *CentralizedNQoS*). O mapeamento entre parâmetros de QoS condizentes com os subsistemas *CPU* e *LinkQueue* é feito respectivamente por meio dos componentes *CPUMapper* e *QueueMapper*. A interface de toda a arquitetura é feita pelo componente *AdmissionController* no subsistema *QoSOS*, de forma que componentes que representam usuários (aplicações multimídia requisitando QoS) são ligados à arquitetura por intermédio dele.

Uma instanciação do Framework de Sintonização de QoS, por sua vez, é descrita através de uma única seção *MtHierarchy* e algumas seções *MtSystem*. De forma similar à negociação, cada subsistema (seção *MtSystem*) possui instâncias de componentes em um nível de abstração particular (subseção *Instances*) e suas devidas ligações naquele nível (subseção *Attachments*). Os tipos básicos de componentes para a sintonização são o *AdjCtrl* (controlador de ajuste) e o *QosTun* (sintonizador de QoS e monitor implícito associado). A hierarquia de subsistemas de negociação da arquitetura QoSOS apresentada na Figura 4.1 possui uma hierarquia correspondente na sintonização de QoS, conforme é apresentado na Seção 4.7.

(a)



(b)

```
System QoSOS
   Instances
      AdmissionController : AdmCtrl();
      Negotiator : QoSNeg();
      CPUMapper : QoSMap();
      QueueMapper : QoSMap();
   Attachments
      AdmissionConotroller <=> Negotiator;
      Negotiator <=> CPUMapper;
      Negotiator <=> QueueMapper;
EndSystem

System CPU
   Instances
      AdmissionController : AdmCtrl();
   Attachments
EndSystem

System LinkQueue
   Instances
      AdmissionController : AdmCtrl();
   Attachments
EndSystem

Hierarchy QoSOSNegFramework
   QoSOS.Negotiator <=> CPU.AdmissionController;
   QoSOS.Negotiator <=> LinkQueue.AdmissionController;
Hierarchy
```

Figura 4.1 – Negociação de QoS na Arquitetura QoSOS: (a) representação gráfica; (b) especificação em LindaQoS.

A Figura 4.2 apresenta a notação EBNF (*Extended Backus Naur Form*) da linguagem. Por convenção, os símbolos não-terminais são escritos sem negrito e itálico, assim como os terminais (literais e palavras reservadas) são escritos entre aspas duplas e em negrito. Os símbolos entre chaves indicam construções opcionais da linguagem. Para facilitar o entendimento, o símbolo *NomeSimples* é excepcionalmente descrito em português.

```
---- LindaQoS v1.0 - EBNF
----        (LINguagem de Descrição de Arquitetura com QoS)
----         (QoS Architecture Description Language)


Spec := Bloco Spec | Bloco

Bloco := BlocoSystem | BlocoHierarchy
         | BlocoMtSystem | BlocoMtHierarchy

BlocoSystem := "System" NomeSimples ":"
               "Instances" ":"
               { Lista_NegInstance }
               "Attachments" ":"
               { Lista_Attachment }
               "EndSystem"

BlocoHierarchy := "Hierarchy" NomeSimples ":"
                  Lista_Attachment_Porta
                  "EndHierarchy"

BlocoMtSystem := "MtSystem" NomeSimples ":"
             "Instances" ":"
             { Lista_TunInstance }
             "Attachments" ":"
             { Lista_Attachment }
             "EndMtSystem"

BlocoMtHierarchy := "MtHierarchy" NomeSimples ":"
                Lista_Attachment_Porta
                "EndMtHierarchy"

Lista_NegInstance := NegInstance Lista_NegInstance
                   | NegInstance

NegInstance := QoSNegInst | AdmCtrlInst | QoSMapInst

QoSNegInst:= NomeSimples ":" "QosNeg" "(" NomeSimples ")" ";"

AdmCtrlInst := NomeSimples ":""AdmCtrl""(" NomeSimples ")"";"
               | NomeSimples ":" "AdmCtrl" "(" ")" ";"

QoSMapInst := NomeSimples ":""QoSMap" "(" NomeSimples ")" ";"
```

```
Lista_TunInstance := TunInstance Lista_TunInstance
                   | TunInstance

TunInstance := QoSTunInst | AdjCtrlInst

QoSTunInst := NomeSimples ":" "QoSTun"
              "(" NomeSimples ";" NomeSimples ")" ";"

AdjCtrlInst := NomeSimples ":""AdjCtrl""(" NomeSimples ")"";"
              | NomeSimples ":" "AdjCtrl" "(" ")" ";"

Lista_Attachment := Attachment Lista_Attachment |Attachment

Attachment := NomeSimples "<" "=" ">" NomeSimples ";"

Lista_Attachment_Porta :=
        Lista_Attachment_Porta Attachment_Porta
        | Attachment_Porta

Attachment_Porta := NomeSimples "." NomeSimples
                    "<" "=" ">"
                    NomeSimples "." NomeSimples

NomeSimples := uma letra seguida de letras e números ou "_"
               até o limite de 128 caracteres
```

Figura 4.2 – Notação EBNF de LindaQoS

4.3 Compilador LindaQoS v1.0

O *Compilador LindaQoS v1.0* traduz especificações *LindaQoS* em um arquivo objeto com uma descrição arquitetural equivalente. Tal descrição toma por base os estilos em Wright apresentados no Capítulo 3. A implementação do *Compilador* foi feita manualmente e levando em consideração a necessidade de uma futura extensão que também permita a tradução do código fonte para uma linguagem de programação.

Uma vantagem especial no fato da gramática de *LindaQoS* ser bastante pequena e simples é que a implementação manual de um compilador para a linguagem também é simples. Nesse caso especial, chega a justificar a não-utilização de um gerador automático de *parsers* (como o YACC), já que ambas as abordagens teriam um tempo de implementação compatível. No mais, a performance do compilador não é um requisito, já que o algoritmo é linear. Esse cenário motivou a implementação manual em C++ do Compilador como um

parser recursivo descendente (Aho & Ullman, 1995). Sua estrutura básica pode ser vista na Figura 4.3, extraída de (Grune et al., 2000).

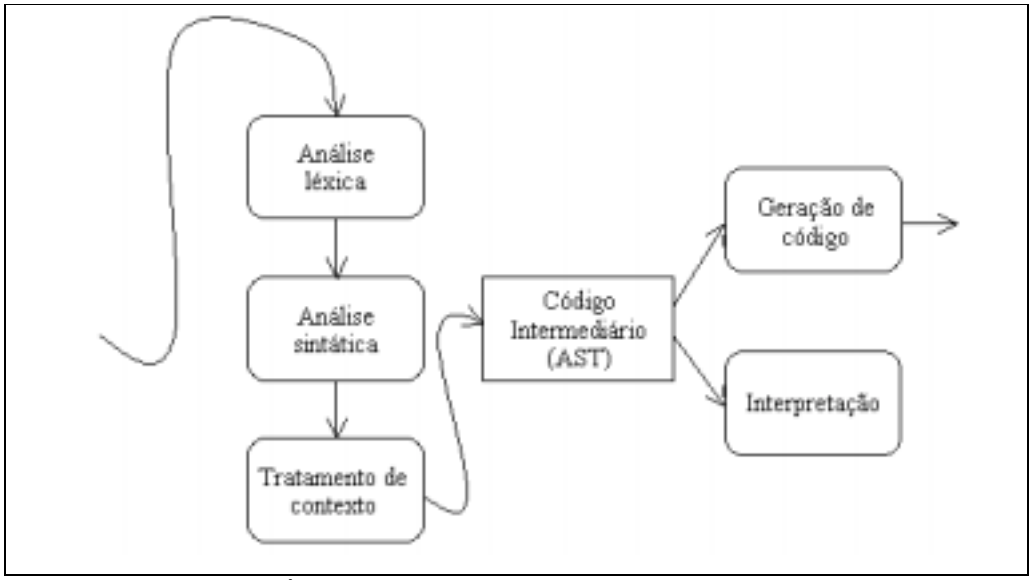


Figura 4.3 – Estrutura básica de um compilador.

Convém ressaltar que o uso do método recursivo descendente falha em construir um *parser* correto para qualquer gramática. A mais importante restrição é que a gramática de *LindaQoS* teve que ser escrita livre de recursões à esquerda.

Outro problema está na regra `TunInstance := QoSTunInst | AdjCtrlInst`. Isso porque tanto `QoSTunInst` quanto `AdjCtrlInst` iniciam com uma seqüência de símbolos iguais (`NomeSimples ":"`). Nesse caso, como a árvore sintática é construída *top-down*, o *parser* precisaria decidir entre construir um nó filho representando `QoSTunInst` ou `AdjCtrlInst` ao consumir o primeiro token `NomeSimples`, o que não é possível.

A solução encontrada na implementação foi visualizar a regra como se fosse `TunInstance := InicioInstance FimQoSTun | InicioInstance FimAdjCtrl` , onde `InicioInstance := NomeSimples ":"`. Os símbolos `FimQoSTun` e `FimAdjCtrl` correspondem respectivamente ao restante das regras `QoSTunInst` e `AdjCtrlInst` (apresentadas na Figura 4.2). De uma forma simples, o Compilador lê os dois primeiros *tokens* e posterga a decisão de construir o nó correspondente apenas depois de ler o *token* `QoSTun` ou o `AdjCtrl`. Problema similar é encontrado na regra `NegInstance` e resolvido da mesma forma.

Baseada nessa estrutura, foi feita uma divisão modular do programa percebendo a clara separação entre cada fase de compilação. Os módulos que compõe o programa são *Msg*, *Lexico*, *Syntax*, *SymbTable* e *CodeGen*. O módulo *Msg* tem a função de fornecer suporte aos outros módulos para o tratamento de erros durante o processo de compilação e exibir as mensagens de aviso e erro. Com essa abordagem, a interface com o usuário fica toda separada nesse módulo, o que facilita a portabilidade e a rápida alteração de mensagens para torná-las mais claras.

Para a análise léxica, foi definido o módulo *Lexico*, que lê o arquivo fonte e o repassa como uma seqüência de *tokens*. A análise sintática, feita pelo módulo *Syntax*, consome esses *tokens* e constrói uma árvore sintática abstrata (ou AST), levando em conta informações de contexto e tabela de símbolos, ambas gerenciadas pelo módulo *SymbTable*. Uma vez construída uma representação intermediária, que é a AST, o módulo *CodeGen* efetua a geração do código resultante (nesse caso, uma descrição arquitetural equivalente em Wright).

Conforme mencionado anteriormente, o Compilador reflete o fato que LindaQoS foi pensada para futuramente ser estendida para um contexto mais geral, em que não apenas o meta serviço de provisão de QoS será descrito, mas também o provedor de serviços onde ele atua. Essa linguagem estendida poderá até mesmo vir a ser uma nova ADL ou uma extensão de uma ADL existente, dependendo das futuras implicações. A necessidade de uma linguagem simples para descrever a instanciação torna-se clara quando se quer ter uma visão de mais alto nível do meta-serviço e também uma ferramenta automática para analisar aspectos arquiteturais. O objetivo aqui é ter uma forma simplificada de descrever o meta serviço de provisão de QoS, sem levar em conta detalhes arquiteturais ou de implementação, em conjunto com uma ferramenta para traduzir diretamente essa especificação para ambos os níveis.

A descrição resultante em Wright pode ser utilizada para realizar várias verificações formais (Allen, 1997):

1. Consistência de Porta-Computação (componente)
2. Conector livre de deadlock (conector)
3. Papéis livres de deadlock (papel)
4. Único iniciador (conector)
5. Confirmação de iniciador (qualquer processo)

6. Substituição de parâmetro (instância)
7. Range check (instância)
8. Compatibilidade porta-papel (attachment)
9. Restrições de estilo (configuração)
10. Consistência de estilo (estilo)
11. Completude de attachment (configuração)

Finalmente, a mesma linguagem pode ser usada futuramente para também gerar código objeto em Java, caso em que esse Compilador deverá ser estendido para tal propósito. Isso completa as necessidades de uma ferramenta para tradução da linguagem, apresentando código objeto final tanto em nível arquitetural quanto em nível de implementação (gerando frameworks que funcionam como esqueletos de código que precisam ser instanciados).

A interface do programa é simples. O Compilador recebe como parâmetro o nome do arquivo de entrada para ser traduzido e responde com a tradução desse programa para Wright ADL na saída padrão (tela). Para simplificar o uso, não há a necessidade de parâmetros opcionais além do nome do arquivo de entrada.

As eventuais mensagens de erro léxico ou sintático são mostradas na própria tela. Uma única classe (*Msg*) é responsável pela saída do programa de forma a simplificar a portabilidade para outros ambientes operacionais quando for necessário.

O Compilador também testa se as ligações entre componentes (*attachments*) são válidas, exibindo mensagens de erro em caso negativo. As ligações nos blocos de hierarquia devem ser verificadas para constatar se todas associam dois componentes de subsistemas distintos (um *QoSNeg* e um *AdmCtrl*). Por sua vez, as cláusulas de ligação dentro de um mesmo subsistema possuem duas restrições: i) devem associar cada componente *AdmCtrl* a um único componente *QoSNeg*; ii) devem associar componentes *QoSNeg* a um ou mais componentes *QoSMap*. Convém ressaltar que a validação das restrições de estilo é feita pelas ferramentas de análise de Wright, e não pelo Compilador, para evitar a implementação redundante de funcionalidades.

4.4
Tradução para a ADL Wright

Já foi visto que um arquivo fonte em LindaQoS é composto de várias seções *System* ou *MtSystem*, uma única seção *Hierarchy*, e uma única *MtHierarchy*. Ao traduzir essa especificação para Wright, cada subsistema (*System* ou *MtSystem*) define um tipo de componente composto, enquanto cada hierarquia (seja de negociação ou de sintonização) é vista como uma configuração arquitetural, onde cada subsistema se torna uma instância de um componente composto e cada ligação entre subsistemas é vista como uma instância de um conector *Interlevel*.

A descrição arquitetural resultante, portanto, é composta de duas configurações: uma representando a hierarquia de subsistemas de negociação e outra a hierarquia de sintonização. A seguir, os passos para a tradução são mais bem detalhados, tomando como exemplo a arquitetura QoSOS apresentada na Figura 4.1.

1 – Cada subsistema de negociação (*System*) é traduzido como um tipo de componente composto. Um subsistema responsável pela negociação de QoS na CPU, extraído da Figura 4.1, é apresentado na Figura 4.4(a) e é mapeado de acordo com a Figura 4.4(b);

(a)

```
System CPU
   Instances
      AdmissionController
      : AdmCtrl();
   Attachments
EndSystem
```

(b)

```
Component CPType_CPU
 Port out3 = InterlevelInput
 Computation =
  Configuration CPU
   Style LowestNQos
    Instances
     ca : AdmCtrl(admstr1;1);
    Attachments
  End Configuration
 Bindings
  out3 = ca.interlevel 1;
 End Bindings
```

Figura 4.4 – O subsistema CPU da arquitetura QoSOS: (a) em LindaQoS; (b) em Wright

a) Em Wright, um componente composto é definido como um componente representado por uma configuração, que pode seguir um estilo. Por motivo de didática, inicialmente será apresentada a montagem dessa configuração e só depois é demonstrado o mapeamento das portas externas. No caso de um subsistema de negociação de QoS, o nome dessa configuração é igual ao dado no arquivo em LindaQoS (em nosso

exemplo, CPU). O estilo, por sua vez, é automaticamente identificado pelo Compilador (*LowestNQoS, CentralizedNQoS* ou *DistributedNQoS*).

i. Se a subseção *Instances* possuir apenas componentes do tipo *AdmCtrl* instanciados, o estilo será *LowestNQoS*;

ii. Se a subseção *Instances* tiver apenas um componente do tipo *QoSNeg* instanciado, o estilo usado será *CentralizedNQoS*;

iii. Havendo mais de um componente *QoSNeg* instanciado, o estilo usado será o *DistributedNQoS*.

b) Cada sentença da subseção *Instances* em LindaQoS é traduzida diretamente como uma sentença de instanciação de componente na configuração em Wright.

i. O tipo AdmCtrl é mapeado como um componente Wright do tipo AdmCtrl. Seus parâmetros são a estratégia de admissão (no caso de ser primitivo) e a cardinalidade da porta *interlevel*, que é calculada pelo Compilador de acordo com o número de ligações externas a esse componente;

ii. O tipo QoSMap é diretamente convertido em um componente Wright do tipo QoSMap. A estratégia de mapeamento é passada como parâmetro;

iii. O tipo QoSNeg é mapeado como um componente Wright do tipo QoSNeg. Seus parâmetros envolvem a cardinalidade das portas *interlevel* e *intralevel* (que são calculadas automaticamente pelo Compilador) e a estratégia para negociação, ou seja, a forma como deve ser feita a divisão de responsabilidade entre os subsistemas.

c) Cada sentença da subseção *Attachments* em LindaQoS (que omite a existência de um conector intermediário em Wright) é traduzida para Wright de acordo com os tipos de componentes que são ligados:

i. Uma ligação entre um controlador de admissão e um negociador de QoS necessita da instanciação de um conector `Intralevel` intermediário (na subseção *Instances* em Wright). Depois, os respectivos componentes associados em LindaQoS são ligados por duas cláusulas em Wright. A primeira liga uma porta *intralevel* do controlador de admissão ao papel *ca* do conector

Intralevel. A segunda liga uma porta *intralevel* no negociador de QoS ao mesmo conector pelo papel *neg*;

ii. Uma ligação entre um negociador de QoS e um mapeador de QoS exige a instanciação de um conector `NegToMapper` intermediário (na subseção *Instances* em Wright). A ligação entre ambos é feita por duas cláusulas em Wright. A primeira delas liga uma porta *translate* no negociador de QoS ao papel *neg* no conector intermediário. A segunda cláusula liga a porta *translate* no mapeador de QoS ao papel *mapper* no mesmo conector;

iii. Demais associações são inválidas, o que é apontado por mensagens de erro geradas pelo Compilador.

d) O nome desse tipo de componente composto em Wright será igual ao nome dado ao subsistema no código fonte em LindaQoS acrescido do prefixo "`CPType_`";

e) Finalizando o procedimento, o Compilador faz a associação entre as portas dos componentes internos com as do componente composto. As portas do componente composto são explicitadas, exatamente como em um componente não-composto, através de sentenças que iniciam com a palavra reservada *Port* (como em "`Port out3 = InterlevelInput`" na Figura 4.4(b)). Depois, essas portas do componente composto são mapeadas em portas dos componentes internos na seção *Bindings* (como em "`out3 = ca.interlevel 1;`" na Figura 4.4(b)).

2 – Uma vez que cada subsistema de negociação de QoS (seções *System* em LindaQoS) é traduzido como um tipo de componente composto, a seção *Hierarchy* é traduzida como uma configuração desses componentes compostos ligados entre si. A Figura 4.5 ilustra a configuração formada pela especificação da negociação de QoS na arquitetura QoSOS (apresentada na Figura 4.1).

a) Para cada cláusula de ligação na seção *Hierarchy* em LindaQoS, um conector *Interlevel* é instanciado. Além disso, cada subsistema de negociação que aparecer em pelo menos uma cláusula é instanciado uma única vez;

b) Cada ligação na seção *Hierarchy* deve ser traduzida como duas sentenças em Wright. Cada uma delas associa a devida porta externa

de um dos componentes compostos (subsistemas) ao papel *ca* ou *neg* do conector *Interlevel* (dependendo de qual é o componente interno associado a essa porta).

```
Configuration QoSNegFramework
   Style Hierarchy
   Component CPType_CPUManager
      Port out3 = CA_QoSNegProtocol
      Computation = (...)
      Bindings
         out3 = ca.interlevel1;
      End Bindings

   Component CPType_OS
      Port externo1 = CA_QoSNegProtocol
      Port out2 = CA_QoSNegProtocol
      Port out4 = CA_QoSNegProtocol
      Computation = (...)
      Bindings
         out2 = neg.interlevel2;
         out4 = neg.interlevel1;
      End Bindings

   Component CPType_QueueManager
      Port out1 = CA_QoSNegProtocol
      Computation = (...)
      Bindings
         out1 = ca.interlevel1;
      End Bindings

   Instances
      CP_CPUManager : CPType_CPUManager;
      CP_OS : CPType_OS;
      CP_MemManager : CPType_MemManager;
      MemManager_out1_INTER_OS_out2 : Interlevel();
      CPUManager_out3_INTER_OS_out4 : Interlevel();
   Attachments
      CP_MemManager.out1 as MemManager_out1_INTER_OS_out2.ca;
      CP_OS.out2 as MemManager_out1_INTER_OS_out2.neg;
      CP_CPUManager.out3 as CPUManager_out3_INTER_OS_out4.ca;
      CP_OS.out4 as CPUManager_out3_INTER_OS_out4.neg;
End Configuration
```

Figura 4.5 – Hierarquia de negociação de QoS na arquitetura QoSOS

A tradução de especificações em LindaQoS de mecanismos de sintonização de QoS é feita de forma semelhante. Componentes compostos são criados a partir de cada seção *MtSystem* e depois são organizados em uma configuração seguindo o estilo *MtHierarchy*. Conectores *Interlevel* são instanciados para regularem a associação entre diferentes subsistemas de sintonização.

O tipo AdjCtrl é traduzido para Wright como um componente de mesmo nome. O tipo QoSTun em LindaQoS é equivalente a um sintonizador de QoS e um monitor (implícito) associado, o que resulta na instanciação de dois componentes em Wright, sendo um QoSTun e outro Monitor, além de um conector Verify que os associa. O Compilador faz a ligação entre esses componentes e o conector intermediário.

Novamente, quando é necessário calcular a cardinalidade das portas dos componentes, isso é automaticamente feito pelo Compilador. Os estilos usados nas configurações dos componentes compostos também são inferidos pelo Compilador entre as opções *LowestTQoS*, *CentralizedTQoS* e *DistributedTQoS*. Um maior detalhamento desse procedimento de tradução é omitido por ser muito semelhante ao de negociação de QoS.

4.5 Aplicação de LindaQoS para a orquestração de QoS intra-mídia e inter-mídia

Nesta seção será apresentada uma aplicação de LindaQoS no contexto da provisão de QoS em um sistema hipermídia . A intenção é oferecer uma visão integrada da orquestração de QoS intra-mídia e inter-mídia em apresentações hipermídia. A Figura 4.6 faz a representação gráfica da negociação de QoS em tal contexto. A representação em LindaQoS é apresentada na Figura 4.7. Para não poluir excessivamente o exemplo, os mapeadores de QoS são omitidos.

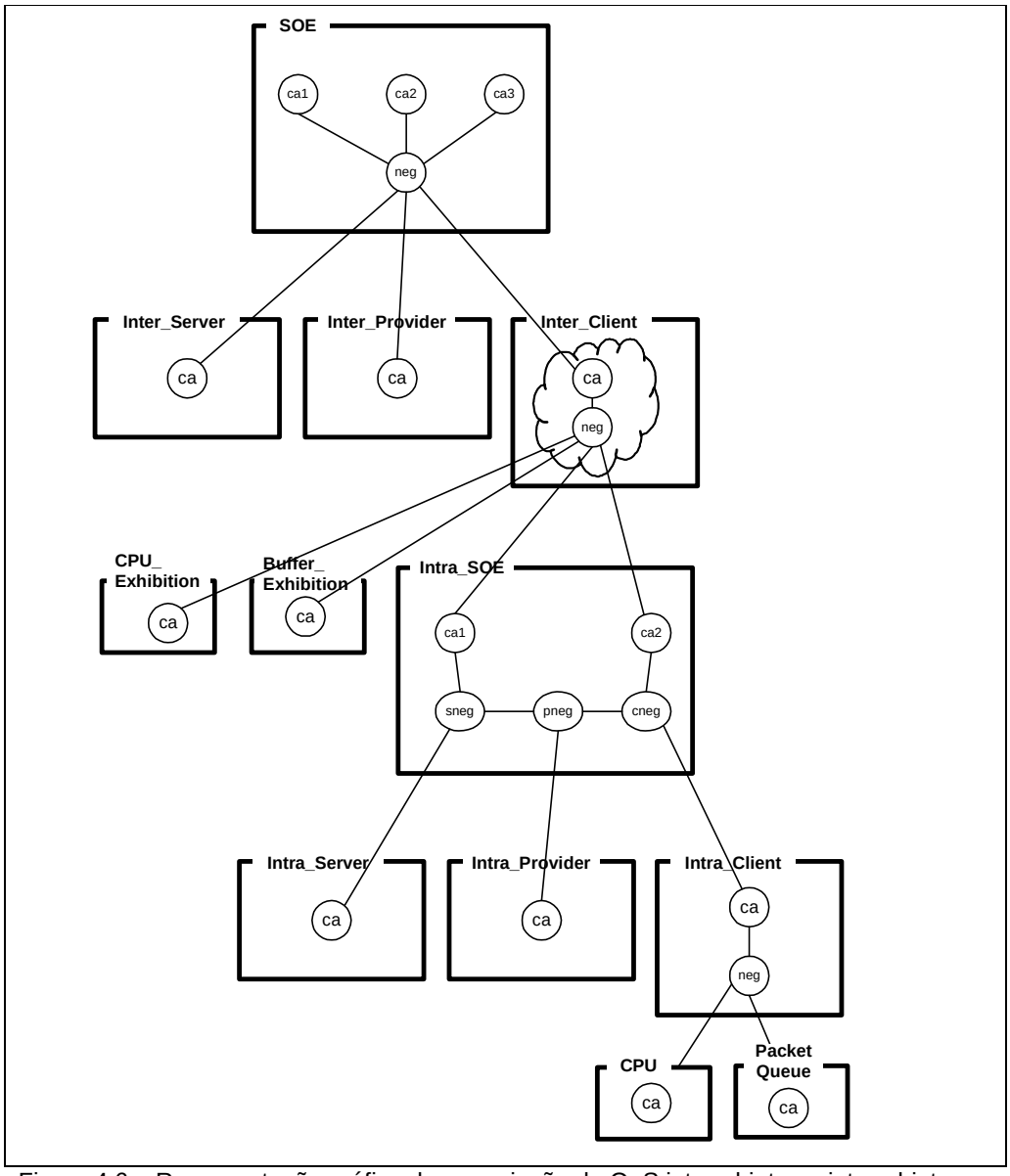


Figura 4.6 – Representação gráfica da negociação de QoS intra-objetos e inter-objetos em sistemas hipermídia

A admissão do fluxo (nesse caso, um documento multimídia) é feita pelo subsistema SOE, que tem a função de dividir a QoS inter-objetos (sincronização inter-mídia) entre o Servidor de Documentos (Inter_Server), o provedor de comunicação (Inter_Provider) e a plataforma de exibição (Inter_Client). A orquestração é centralizada. Os subsistemas servidor e provedor são considerados primitivos, apenas para tornar o exemplo mais simples.

O subsistema Inter_Client, para orquestrar a apresentação sincronizada dos objetos do documento, atua como um usuário de QoS intra-objeto (sincronização

intra-mídia) da plataforma de exibição (CPU e Buffer) e do ambiente SOE (para carregar individualmente cada objeto de mídia do documento em um tempo de pré-busca calculado).

O Intra_SOE, ao receber uma requisição do subsistema Inter_Client, deve dividir a responsabilidade sobre a QoS intra-mídia entre os subsistemas Intra_Server, Intra_Provider e Intra_Client. Essa orquestração é constituída por um mecanismo distribuído. Novamente, para tornar o exemplo simples, os subsistemas do servidor e provedor são considerados primitivos.

O subsistema Intra_Client divide sua responsabilidade de QoS entre os subsistemas primitivos CPU e Packet Queue (subsistema de rede do sistema operacional), em uma orquestração centralizada, terminando todo o processo de orquestração.


```
System SOE
   Instances
      ca1 : AdmCtrl();
      ca2 : AdmCtrl();
      ca3 : AdmCtrl();
      neg : QoSNeg();
   Attachments
      ca1 <=> neg;
      ca2 <=> neg;
      ca3 <=> neg;
EndSystem

System Inter_Server
   Instances
      ca : AdmCtrl();
   Attachments
EndSystem

System Inter_Provider
   Instances
      ca : AdmCtrl();
   Attachments
EndSystem

System Inter_Client
   Instances
      ca : AdmCtrl();
      neg : QoSNeg();
   Attachments
      ca <=> neg;
EndSystem

System CPU_Exhibition
   Instances
      ca : AdmCtrl();
   Attachments
EndSystem
```

```
System Intra_SOE
   Instances
      ca1 : AdmCtrl();
      ca2 : AdmCtrl();
      neg_server : QoSNeg();
      neg_provider : QoSNeg();
      neg_client : QoSNeg();
   Attachments
      ca1 <=> neg_server;
      ca2 <=> neg_client;
      neg_server <=> neg_provider;
      neg_provider <=> neg_client;
EndSystem

System Intra_Server
   Instances
      ca : AdmCtrl();
   Attachments
EndSystem

System Intra_Provider
   Instances
      ca : AdmCtrl();
   Attachments
EndSystem

System Intra_Client
   Instances
      ca : AdmCtrl();
      neg : QoSNeg();
   Attachments
      ca <=> neg;
EndSystem

System CPU
   Instances
      ca : AdmCtrl();
```

```
System Buffer_Exhibition
   Instances
      ca : AdmCtrl();
   Attachments
EndSystem
```

```
   Attachments
EndSystem

System Packet_Queue
   Instances
      ca : AdmCtrl();
   Attachments
EndSystem
```

```
Hierarchy SOE_Structure
   SOE.neg <=> Inter_Server.ca;
   SOE.neg <=> Inter_Provider.ca;
   SOE.neg <=> Inter_Client.ca;
   Inter_Client.neg <=> CPU_Exhibition.ca;
   Inter_Client.neg <=> Buffer_Exhibition.ca;
   Inter_Client.neg <=> Intra_SOE.ca1;
   Inter_Client.neg <=> Intra_SOE.ca2;
   Intra_SOE.neg_server <=> Intra_Server.ca;
   Intra_SOE.neg_provider <=> Intra_Provider.ca;
   Intra_SOE.neg_client <=> Intra_Client.ca;
   Intra_Client.neg <=> CPU.ca;
   Intra_Client.neg <=> Packet_Queue.ca;
EndHierarchy
```

Figura 4.7 – Representação em LindaQoS da negociação de QoS intra-objetos e inter-objetos em sistemas hipermídia

4.6 Aplicação de LindaQoS em um ambiente de oferecimento de QoS

Para demonstrar a aplicação de LindaQoS, um outro cenário de oferecimento de QoS é proposto na Figura 4.8. O exemplo é simples e formado apenas por duas estações finais e um elemento intermediário (um roteador) onde a reserva de recursos é feita através do protocolo RSVP (). O sistema operacional de cada elemento também fornece garantias de QoS de acordo com a arquitetura QoSOS (Moreno, 2002).

Quando uma nova admissão de fluxo é feita, o subsistema *RSVPNetwork*, que a recebe, inicia os mecanismos de orquestração de recursos. Para admitir o fluxo, é necessário reservar recursos em todos os elementos que compõe a arquitetura, ou seja, é preciso acionar recursivamente os subsistemas *QoSOS1*, *QoSOS2* e *QoSOSn.*

A negociação de QoS prossegue, com cada um dos subsistemas acionados na arquitetura QoSOS (*QoSOS1*, *QoSOS2* e *QoSOSn*) repassando a responsabilidade sobre a provisão de QoS a seus subsistemas internos primitivos. Tais subsistemas são os de gerenciamento de CPU (respectivamente *CPU1*, *CPU2* e *CPUn*) e os subsistemas de comunicação do sistema operacional (nas estações

finais, os subsistemas *Queue1* e *Queue2*, individualmente, gerenciam as filas de pacotes do SO. Já no elemento intermediário os subsistemas *Queuen1* e *Queuen2* operam sobre as filas de pacotes para ambas as estações finais).

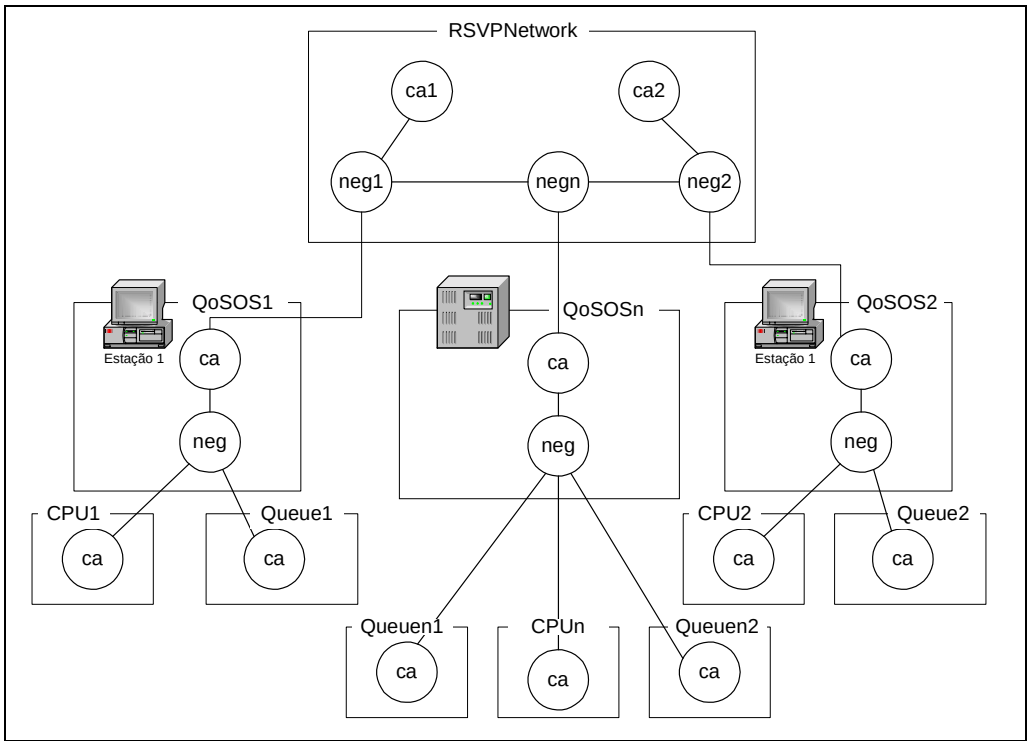


Figura 4.8 – Uma arquitetura para provisão de QoS

A especificação em LindaQoS da arquitetura proposta na Figura 4.8 é apresentada na Figura 4.9. Como se vê, a linguagem ainda não apresenta muitas facilidades para promover um maior reuso (por exemplo, na repetição da arquitetura QoSOS nos três nós). Mesmo assim, a descrição é suficientemente concisa para permitir um entendimento mais natural que em comparação a sua implementação ou descrição arquitetural.

```
#################################################################
#
# 01/07/2003
#
# End-to-end RSVP network with 3 nodes
# (2 end-systems, 1 router).
#
# OS support for both layers in end-systems and
# OS support for network layer only in the router.
```

```
# OS manages CPU and link.
# For router, 2 links are managed.
#
############################################################

system CPU1:
  instances:
    ca : AdmCtrl ( CPUadmstr1 );
  attachments:
endsystem

system Queue1:
  instances:
    ca : AdmCtrl ( Queueadmstr1 );
  attachments:
endsystem

system QoSOS1:
  instances:
    ca  : AdmCtrl ();
    neg : QoSNeg( negstr1; mapstr1 );
  attachments:
    ca <=> neg;
endsystem

#####################################################

system CPU2:
  instances:
    ca : AdmCtrl ( CPUadmstr2 );
  attachments:
endsystem

system Queue2:
  instances:
    ca : AdmCtrl ( Queueadmstr2 );
  attachments:
endsystem

system QoSOS2:
  instances:
    ca  : AdmCtrl ();
    neg : QoSNeg( negstr2; mapstr2 );
  attachments:
    ca <=> neg;
endsystem

#####################################################

system CPUn:
  instances:
    ca : AdmCtrl ( CPUadmstrn );
  attachments:
endsystem

system Queuen1:
  instances:
    ca : AdmCtrl ( Queueadmstrn1 );
  attachments:
```

```
endsystem

system Queuen2:
  instances:
    ca : AdmCtrl ( Queueadmstrn2 );
  attachments:
endsystem

system QoSOSn:
  instances:
    ca  : AdmCtrl ();
    neg : QoSNeg( negstrn; mapstrn );
  attachments:
    ca <=> neg;
endsystem

#####################################################

system RSVPNetwork:
  instances:
    ca1  : AdmCtrl ();
    ca2  : AdmCtrl ();
    neg1 : QoSNeg( negstr3; mapstr3 );
    neg2 : QoSNeg( negstr3; mapstr3 );
    negn : QoSNeg( negstr3; mapstr3 );
  attachments:
    ca1  <=> neg1;
    ca2  <=> neg2;
    neg1 <=> negn;
    negn <=> neg2;
endsystem

#####################################################

hierarchy h:
  QoSOS1.neg <=> CPU1.ca;
  QoSOS1.neg <=> Queue1.ca;
  QoSOS2.neg <=> CPU2.ca;
  QoSOS2.neg <=> Queue2.ca;
  QoSOSn.neg <=> CPUn.ca;
  QoSOSn.neg <=> Queuen1.ca;
  QoSOSn.neg <=> Queuen2.ca;
  RSVPNetwork.neg1 <=> QoSOS1.ca;
  RSVPNetwork.neg2 <=> QoSOS2.ca;
  RSVPNetwork.negn <=> QoSOSn.ca;
endhierarchy
```

Figura 4.9 – Especificação LindaQoS de uma arquitetura para provisão de QoS

4.7 Sintonização de QoS no ambiente QoSOS

A utilização de LindaQoS no contexto da sintonização de QoS é semelhante à negociação. A Figura 4.10 ilustra como se dá a sintonização de QoS na arquitetura QoSOS. O subsistema QoSOS centraliza os mecanismos de sintonização no componente Tuner. Os subsistemas primitivos CPU e Queue são modelados como um controlador de ajuste primitivo associado a um monitor.

Periodicamente, os controladores de ajuste primitivos recebem ou requisitam cálculos estatísticos sobre o fluxo, feitos pelo respectivo monitor. Quando detectam uma violação no contrato de serviço nesse nível de abstração, os controladores tentam reorganizar sua árvore de recursos virtuais primitiva associada. Caso isso não seja possível, eles enviam um alerta ao componente Tuner do subsistema QoSOS. Cabe a esse sintonizador de QoS reorganizar os recursos para manter o nível do serviço. Isso ocorre através de novas chamadas para reorquestração de recursos nos subsistemas internos, o que é intermediado por chamadas ao respectivo mapeador de QoS para a tradução dos parâmetros para o nível de visão dos subsistemas primitivos. Quando o sintonizador não pode manter o nível do serviço inicialmente negociado, ele envia um alerta a um elemento de mais alto nível de abstração, que pode ser até mesmo uma aplicação usuária, para tomar ações nesse sentido. O sintonizador pode também inferir violações no contrato de serviço em seu próprio nível de abstração, isoladas do nível primitivo, através do monitor associado, o que também dispara os mecanismos de sintonização anteriormente descritos.

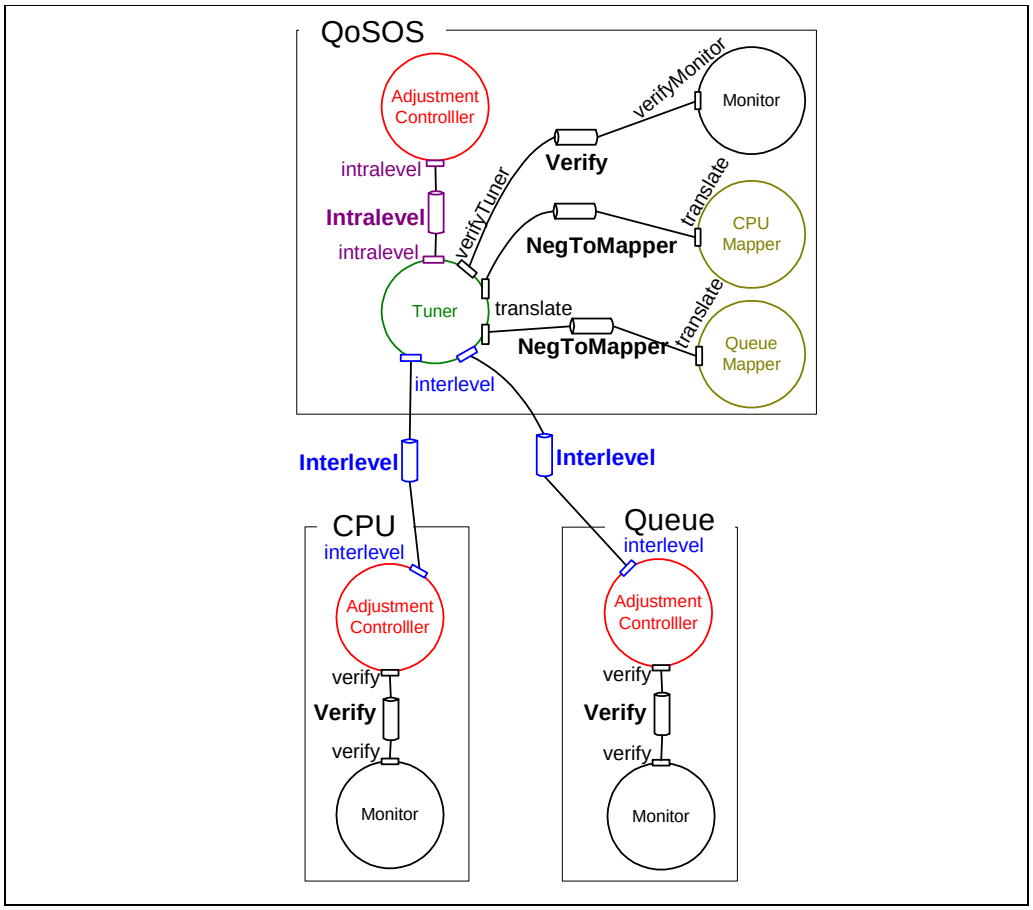


Figura 4.10 – Sintonização de QoS na arquitetura QoSOS

A Figura 4.11 apresenta a especificação em LindaQoS dos mecanismos de sintonização previamente descritos. Além da descrição dos subsistemas CPU, Queue e QoSOS, a seção MtHierarchy faz a ligação entre os subsistemas, associando os controladores de ajuste dos subsistemas primitivos ao componente Tuner.

```
MtSystem CPU
   Instances:
      AdjustmentController : AdjCtrl();
      Monitor : Mon();
   Attachments:
      AdjustmentController <=> Monitor;
EndMtSystem

MtSystem Queue
   Instances:
      AdjustmentController : AdjCtrl();
      Monitor : Mon();
   Attachments:
      AdjustmentController <=> Monitor;
EndMtSystem
```

```
MtSystem QoSOS
   Instances:
      AdjustmentController : AdjCtrl();
      Tuner : QoSTun();
      Monitor : Mon();
      CPUMapper : QosMap();
      QueueMapper : QoSMap();
   Attachments:
      AdjustmentController <=> Tuner;
      Tuner <=> Monitor;
      Tuner <=> CPUMapper;
      Tuner <=> QueueMapper;
EndMtSystem

MtHierarchy Mt_QoSOS
   QoSOS.Tuner <=> CPU.AdjustmentController;
   QoSOS.Tuner <=> Queue.AdjustmentController;
EndMtHierarchy
```

Figura 4.11 – Especificação em LindaQoS da sintonização na arquitetura QoSOS