4
Especializações de LindaX

Este capítulo tem por objetivo demonstrar a exeqüibilidade da abordagem de descrição de arquitetura de sistemas de comunicação proposta nesta tese. Para isso, são apresentados dois estudos de caso de especializações de DSLs LindaX. A Seção 4.1 apresenta a DSL LindaRouter. Essa DSL permite a especificação da arquitetura do "plano de dados" de roteadores programáveis (englobando, principalmente, roteadores baseados em PCs e NPUs). A Seção 4.2 apresenta a DSL LindaQoS, que permite a especificação de arquiteturas complexas de provisão de QoS. Para ambas as DSLs, seguiu-se a seguinte estrutura de apresentação. Primeiramente, são levantados alguns aspectos gerais dos domínios que cada uma delas se propõe a tratar. Em seguida, são ilustrados os estilos LindaX aos quais as DSLs estão ligadas. Alguns desses estilos têm uma especificação de vocabulário consideravelmente extensa. Nesses casos, partes das especificações foram omitidas, simplificadas ou substituídas por diagramas.[1] Por fim, é descrita a semântica das DSLs, mantida pelas ferramentas extensoras correspondentes. Essa descrição é feita basicamente por meio de exemplos de especificação de arquiteturas com relativo grau de complexidade em cada um dos domínios abrangidos pelas DSLs.

É importante salientar que não é o objetivo deste capítulo apresentar a fundo as arquiteturas sobre as quais as DSLs LindaRouter e LindaQoS são definidas. O objetivo foi explorar domínios distintos, mas ainda assim relacionados, na concepção de DSLs e ferramentas extensoras. A partir de LindaRouter e LindaQoS conseguiu-se ilustrar tanto facilidades quanto fraquezas da abordagem proposta, da forma como ela é concebida atualmente. Em especial, ambos os estudos de caso foram fundamentais no estabelecimento de algumas metas futuras acerca do trabalho, particularmente a integração entre DSLs distintas, conforme apresentado no Capítulo 7.

[1]As especificações completas podem ser encontradas no Apêndice B.

PUC-Rio - Certificação Digital Nº 0115599/CA

4.1.
A DSL LindaRouter

A DSL LindaRouter oferece uma semântica própria para a especificação da arquitetura de encaminhamento de pacotes provido por um roteador baseado em software. Embora a estrutura dessa DSL seja genérica o suficiente também para representar a arquitetura de hardware de roteadores, privilegiou-se manter o foco nos aspectos de software desses equipamentos.

A DSL LindaRouter se baseia no detalhamento do trabalho apresentado por Coulson et al. (2003), que ilustra, sob o prisma de tecnologias de CBSD, uma visão bastante geral da estrutura de software do plano de dados de um roteador. Contudo, Coulson et al. não consideram uma questão fundamental, que é o impacto da arquitetura do hardware utilizado na adaptabilidade do roteador. Para melhor definir a DSL proposta nesta seção, é feito primeiramente um resumo das diferentes arquiteturas existentes de hardware de roteador, bem como é apontado de que forma as entidades de software típicas de um roteador se encaixam nessas arquiteturas e o grau de adaptabilidade que esses arranjos de hardware e software permitem.

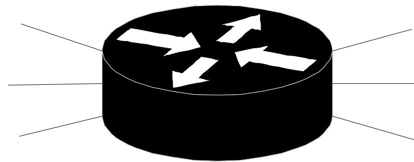
4.1.1.
Arquitetura geral de um roteador

Os elementos principais que constituem uma arquitetura típica de roteador são os módulos de E/S, encaminhamento, sinalização e gerência. A distribuição desses módulos por um roteador pode ser representada genericamente como na Figura 4.1. A forma como cada um dos módulos constituintes é implementado (por hardware ou software) e a função real que eles exercem são diretamente ligadas à arquitetura de hardware subjacente, que por sua vez tem influência direta na capacidade de adaptação do roteador. Duas variantes principais de arquitetura de hardware de roteador, bem como as características de seus módulos constituintes, são resumidamente descritas a seguir.

Em um roteador baseado em PC, os módulos de E/S constituem-se nas interfaces de rede e nos *drivers* desses dispositivos, sendo em geral responsáveis somente pelas funções de nível físico e de enlace na nomenclatura OSI. O módulo de encaminhamento, nesse caso, implementa todas as funções de nível de rede,

PUC-Rio - Certificação Digital Nº 0115599/CA

com exceção das de sinalização e gerência. Esse módulo é executado na CPU do hardware hospedeiro do roteador, juntamente com outras aplicações e serviços, além dos próprios *drivers* de dispositivo que controlam as interfaces de rede. A ligação entre esses módulos é obtida por meio de um barramento, que também é compartilhado por todo o restante do sistema computacional. Essa arquitetura é a que oferece o maior grau de adaptabilidade, uma vez que todas as funções principais do serviço estão concentradas na CPU, sendo, portanto, facilmente reprogramáveis. Contudo, é também a arquitetura em que é mais difícil obter-se alto desempenho e garantias determinísticas de retardo e vazão, devido ao compartilhamento extensivo de recursos, em especial do barramento.



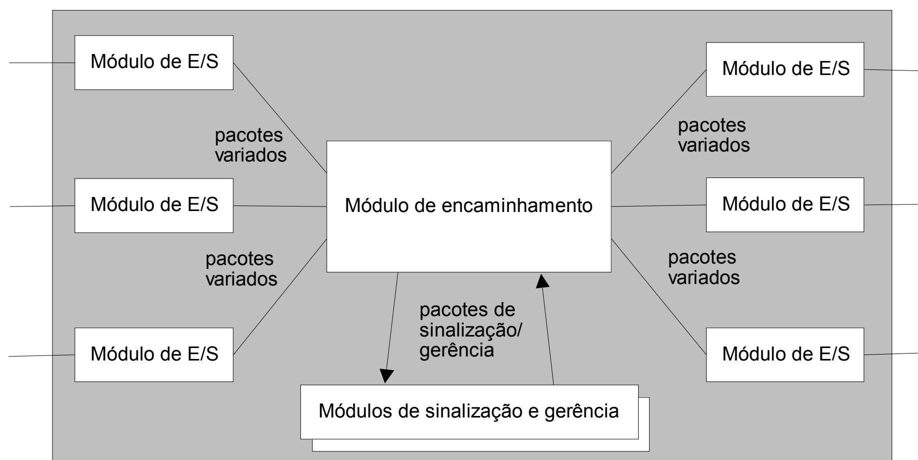


Figura 4.1. Arquitetura geral de um roteador.

Em um roteador baseado em matriz de comutação, o gargalo introduzido pelo barramento é removido. O módulo de encaminhamento constitui-se no próprio hardware da matriz de comutação. Processadores de propósito geral ou baseados em circuitos integrados específicos (*Application-Specific Integrated Circuits* – ASICs) são geralmente acoplados às interfaces de rede, de forma que os módulos de E/S podem executar, além das funções típicas dos *drivers* de dispositivos, boa parte também das funções de nível de rede, como classificação e escalonamento. Nesse caso, somente funções de sinalização e gerência são processadas na CPU do hardware hospedeiro do roteador. O grau de

PUC-Rio - Certificação Digital Nº 0115599/CA

adaptabilidade oferecido por essa arquitetura de roteador, bem como o desempenho final obtido na mesma, depende fundamentalmente do tipo de processador acoplado às interfaces de rede: processadores de propósito geral são extensivamente programáveis, porém têm desempenho aquém dos processadores baseados em ASICs, que por sua vez têm uma programabilidade mais restrita.

As unidades de processamento de rede (*Network Processing Units* – NPUs) (Network Processing Forum, 2001) surgiram como solução de compromisso entre os extremos citados acima. Em geral, uma NPU oferece um arranjo de processadores com diferentes características e funções, interligados por esquemas hierárquicos de memória e barramentos internos altamente eficientes. Como exemplo, alguns desses processadores podem ser responsáveis exclusivamente pelas funções principais de classificação e escalonamento (*fast path*). Outros podem ser dedicados a funções específicas, como verificação de redundância cíclica (*Cyclic Redundancy Check* – CRC) e cálculo de dispersão (*hash*) para consulta a tabelas de encaminhamento. Outros ainda lidam com aspectos menos críticos (*slow path*), mas que demandam mesmo assim alto desempenho, como processamento de campos opcionais, mascaramento de endereços e técnicas de tradução de protocolos.

As estruturas típicas de um roteador apresentadas acima implicam em uma série de observações que influem nas decisões de projeto acerca da DSL LindaRouter. A primeira dessas observações é a existência de uma gama imensa de possíveis configurações de software no plano de dados de um roteador. Dessa forma, deve ser estabelecido na DSL o mínimo possível de regras obrigatórias a serem seguidas por todas essas configurações.

Contudo, cada arquitetura de hardware particular impõe regras de localidade para entidades de software específicas – por exemplo, uma entidade classificadora de pacotes em um roteador baseado em NPU tem que estar executando em um processador específico interno a essa unidade. É interessante que essas regras sejam refletidas também em LindaRouter. Para complicar esse cenário, NPUs podem não estar necessariamente instaladas em uma arquitetura de hardware dotada de matriz de comutação[2], conforme foi apresentado anteriormente. Desse modo, as restrições impostas a entidades executando na NPU podem não ser

[2]Exemplos de NPUs preparadas para serem ligadas ao barramento de PCs podem ser encontrados em (Radisys Corporation, 2004).

PUC-Rio - Certificação Digital Nº 0115599/CA

adequadas ao roteador como um todo e este pode ter restrições globais distintas dependendo de sua arquitetura.

O que se percebe das observações acima é que diferentes configurações de software, seguindo restrições de hardware distintas, podem coexistir no plano de dados de um mesmo roteador. Isso se reflete na composição dos estilos LindaX com os quais a DSL LindaRouter mantém corrrespondência.

4.1.2. Estilos de LindaRouter

Os estilos para arquiteturas de roteadores baseados em software são organizados em uma estrutura hierárquica de herança. A Figura 4.2 ilustra essa estrutura, bem como apresenta diagramaticamente o vocabulário e restrições de cada um dos estilos.[3] Dois desses estilos – `GenericForwarder` e

PUC-Rio - Certificação Digital Nº 0115599/CA

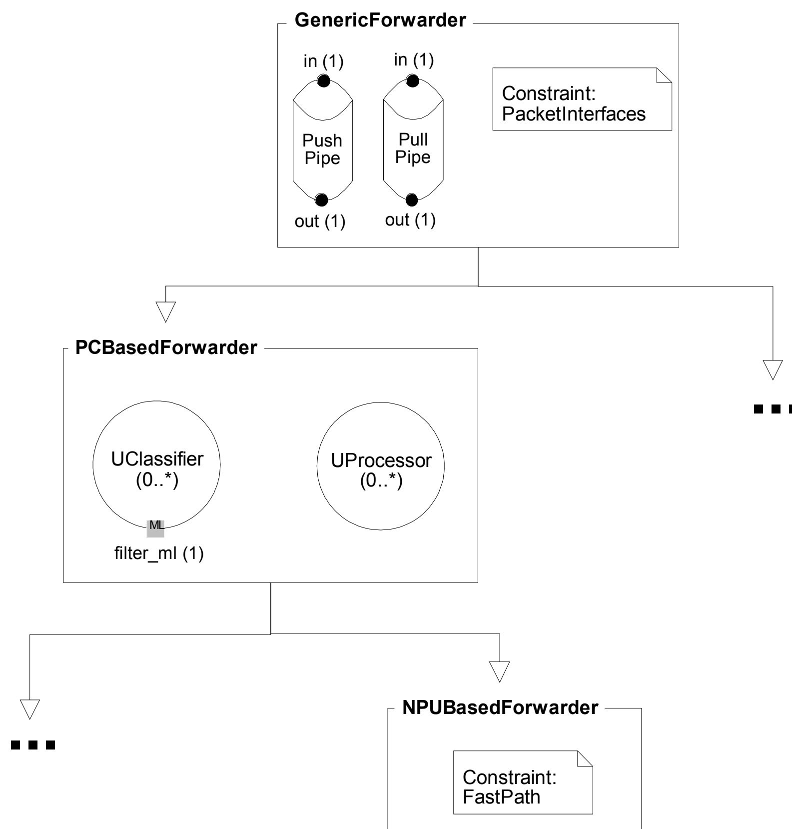


Figura 4.2. Estrutura de herança de estilos de roteador em LindaRouter.

[3] Na notação usada nas Figuras 4.2 e 4.9, a propriedade `Cardinality` para componentes, *pipes*, portas e pontos de acesso é representada entre parênteses, próximo ao nome do elemento.

`NPUBasedForwarder` – já foram introduzidos no Capítulo 3, sendo revisitados mais detalhadamente nesta seção. Obviamente, essa estrutura de estilos pode – e deve – ser continuamente estendida para atender a outras possíveis arquiteturas de roteadores baseados em software.

O estilo-pai `GenericForwarder` define um vocabulário e conjunto de regras mínimos para uma arquitetura de software genérica de roteador. Configurações que seguem esse estilo podem ser compostas de entidades de software que implementam funções arbitrárias de processamento de pacotes e que trocam pacotes entre si por meio de interfaces bem determinadas. Isso se reflete no estilo pela definição de um vocabulário composto de dois tipos de *pipes* (e de interfaces correspondentes). O tipo de *pipe* `PushPipe` define interfaces (tipo `PPacketPush`) às quais um pacote pode ser entregue por uma entidade de software para que seja tratado por outra entidade. O tipo de *pipe* `PullPipe` define interfaces (tipo `PPacketPull`) às quais um pacote pode ser solicitado por uma entidade de software para que ela trate o pacote. A combinação entre interfaces dos tipos acima sendo providas ou requisitadas por uma entidade de software determina a atividade ou passividade dessa entidade. A Figura 4.3 ilustra combinações possíveis dessas interfaces.

O estilo `GenericForwarder` define uma única restrição `PacketInterfaces`, pela qual todas as entidades de software no plano de dados de um roteador devem

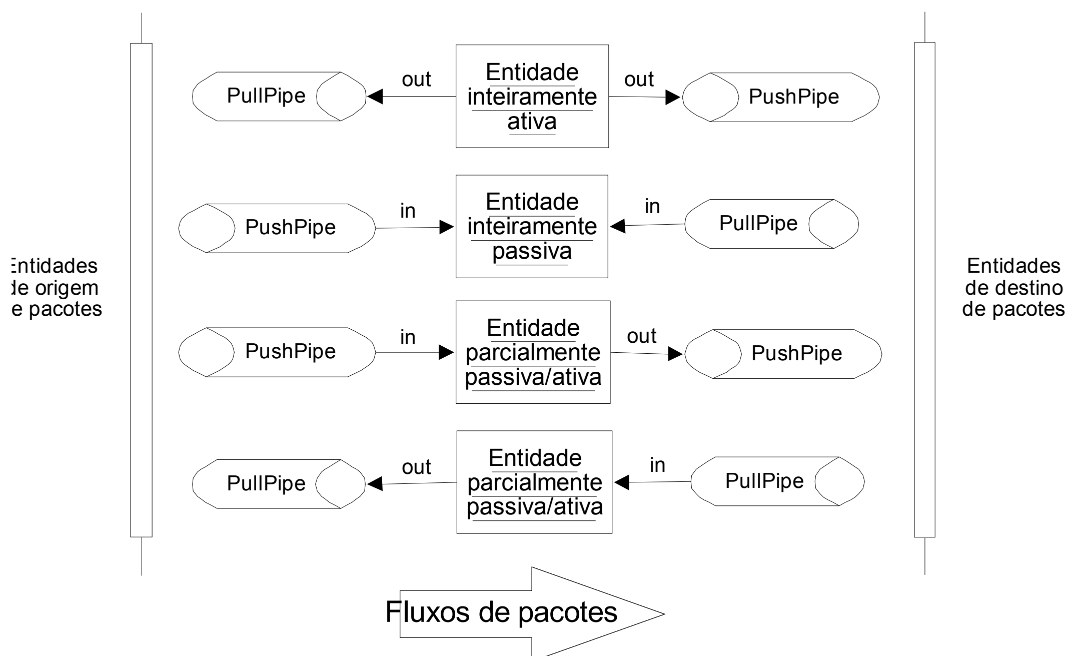


Figura 4.3. Combinações de interfaces *push* e *pull*.

PUC-Rio - Certificação Digital Nº 0115599/CA

obrigatoriamente implementar as interfaces PPacketPull e PPacketPush segundo, no mínimo, uma das combinações descritas na Figura 4.3. A Figura 4.4 apresenta, na notação sem *tags* XML, a descrição completa desse estilo.

```
1  Style GenericForwarder {
2    InterfaceType PPacketPush( behv ) { Behaviour = #behv; }
3    InterfaceType PPacketPull( behv ) { Behaviour = #behv; }
4
5    PipeType PushPipe( behv ) {
6      Behaviour = #behv;
7      AccessPoint in {
8        Cardinality = 1;
9        Type = #PPacketPush; Direction = "in";
10     }
11     AccessPoint out {
12       Cardinality = 1;
13       Type = #PPacketPush; Direction = "out";
14     }
15   }
16
17   PipeType PullPipe( behv ) {
18     Behaviour = #behv;
19     AccessPoint in {
20       Cardinality = 1;
21       Type = #PPacketPull; Direction = "in";
22     }
23     AccessPoint out {
24       Cardinality = 1;
25       Type = #PPacketPull; Direction = "out";
26     }
27   }
28
29   FolPredicate PacketInterfaces {
30     forall c: Components(); {
31       exists p1: Ports( #c );
32              p2: Ports( #c ); {
33         forall s1: Signatures( #p1 );
34                s2: Signatures( #p2 ); {
35           [#s1 != #s2] and
36           ([PropertyValue( #s1,"Type" ) == #PPacketPush] and
37            [PropertyValue( #s1,"Direction" ) == "out"]   and
38            [PropertyValue( #s2,"Type" ) == #PPacketPull] and
39            [PropertyValue( #s2,"Direction" ) == "out"])
40           or
41           ([PropertyValue( #s1,"Type" ) == #PPacketPull] and
42            [PropertyValue( #s1,"Direction" ) == "in"]    and
43            [PropertyValue( #s2,"Type" ) == #PPacketPush] and
44            [PropertyValue( #s2,"Direction" ) == "in"])
45           or
46           ([PropertyValue( #s1,"Type" ) == #PPacketPush] and
47            [PropertyValue( #s1,"Direction" ) == "in"]    and
48            [PropertyValue( #s2,"Type" ) == #PPacketPush] and
49            [PropertyValue( #s2,"Direction" ) == "out"])
50           or
51           ([PropertyValue( #s1,"Type" ) == #PPacketPull] and
52            [PropertyValue( #s1,"Direction" ) == "out"]   and
53            [PropertyValue( #s2,"Type" ) == #PPacketPull] and
54            [PropertyValue( #s2,"Direction" ) == "in"])
55         }
56       }
57     }
58   }
59 }
```

Figura 4.4. Estilo GenericForwarder.

PUC-Rio - Certificação Digital Nº 0115599/CA

Note que o estilo `GenericForwarder` não define tipo algum de componente. Além de servir como estilo-pai de todos os outros estilos de LindaRouter, esse estilo pode ser usado também como uma espécie de “cola” entre configurações de partes distintas de um roteador. Essa forma particular de se representar a ligação entre configurações decorre da ausência em LindaX de um suporte a estilos recursivos (vide discussão a respeito no Capítulo 7).

O estilo `PCBasedForwarder` (vide Figura 4.5) herda o vocabulário e a restrição do estilo `GenericForwarder`, estendendo-o com a definição de três tipos. O tipo de interface `IMetaFilter` permite configurar filtros de pacotes em componentes do tipo `UClassifier`. Por “filtro de pacotes” entenda-se qualquer padrão de dados, reconhecido em pacotes, a partir do qual informações acerca do tratamento dos pacotes possam ser extraídas. Assim, o tipo `UClassifier` prescreve não só a estrutura de classificadores, como também a de marcadores para ações de policiamento e a de componentes responsáveis pelas decisões de encaminhamento. Componentes do tipo `UProcessor` são entidades que atuam sobre os resultados de uma filtragem de pacotes. Essa definição é propositalmente vaga para incluir uma miríade de funções de tratamento de pacotes, como moldagem de tráfego, modificação de fluxos (mascaramento de endereços, tradução de protocolos), escalonamento etc. Note que não é incluída descrição de porta alguma de passagem de pacotes em ambos os tipos de componentes definidos nesse estilo. A obrigatoriedade de implementação dessas portas já é especificada na restrição do estilo-pai `GenericForwarder`, descrito acima.

PUC-Rio - Certificação Digital Nº 0115599/CA

```
1  Style PCBasedForwarder {
2    Superstyle = #GenericForwarder;
3
4    InterfaceType IMetaFilter( behv ) { Behaviour = #behv; }
5
6    ComponentType UClassifier( behv ) {
7      Behaviour = #behv;
8      Port filter_ml {
9        Level = "meta";
10       Cardinality = 1;
11       Type = #IMetaFilter;
12     }
13   }
14
15   ComponentType UProcessor( behv ) {
16     Behaviour = #behv;
17   }
18 }
```

Figura 4.5. Estilo `PCBasedForwarder`.

O estilo `NPUBasedForwarder` é sub-estilo de `PCBasedForwarder`, estendendo-o para arquiteturas de software de roteadores baseados em NPUs. A

Figura 4.6 ilustra esse estilo. A única extensão feita a `PCBasedForwarder` é a restrição `FastPath`, que define que todo componente dos tipos `UClassifier` e `UProcessor` deve ter sua execução obrigatoriamente atrelada a um processador específico de manipulação de pacotes dentro da NPU. Como o esquema de endereçamento de memória associado a esse processador é dependente de plataforma, optou-se pela parametrização desse endereçamento na restrição.

```
1  Style NPUBasedForwarder {
2    Superstyle = #PCBasedForwader;
3
4    FolPredicate FastPath( addr ) {
5      forall t: Tasks( Components( #UClassifier ) union
6                       Components( #UProcessor ) ); {
7        [PropertyValue( #t,"Address" ) == #addr]
8      }
9    }
10 }
```

Figura 4.6. Estilo `NPUBasedForwarder`.

4.1.3. Semântica de LindaRouter

As ferramentas extensoras associadas à DSL LindaRouter detêm pouquíssima informação acerca da semântica de configurações. Isso decorre, em parte, da forma concisa como o vocabulário dos estilos de LindaRouter é definido. Devido à grande quantidade de possíveis configurações do plano de dados de um roteador, privilegiou-se, no caso de LindaRouter, que o máximo possível de semântica fosse obtido a partir da sintaxe da DSL. Isso dá uma maior flexibilidade à linguagem, porém às custas de uma sintaxe menos concisa. Essa decisão de projeto leva em conta o tamanho relativamente reduzido que uma configuração típica de roteador possui. Conforme será visto, isso contrasta fortemente com os estilos e ferramentas extensoras ligados a LindaQoS, que privilegiam sintaxes mais concisas, porém com menor flexibilidade de configuração.

O exemplo de configuração (baseado em (Ueyama et al., 2003)) a ser utilizado nesta seção para apresentar a semântica de LindaRouter é representado diagramaticamente na Figura 4.7. Essa figura ilustra uma configuração típica do software de encaminhamento IP de um roteador usando uma NPU em um sistema com barramento convencional. Essa configuração consiste em vários componentes executando em processadores dedicados à manipulação de pacotes na NPU. Esses componentes são responsáveis pelas funções de classificação, encaminhamento e

PUC-Rio - Certificação Digital Nº 0115599/CA

PUC-Rio - Certificação Digital Nº 0115599/CA

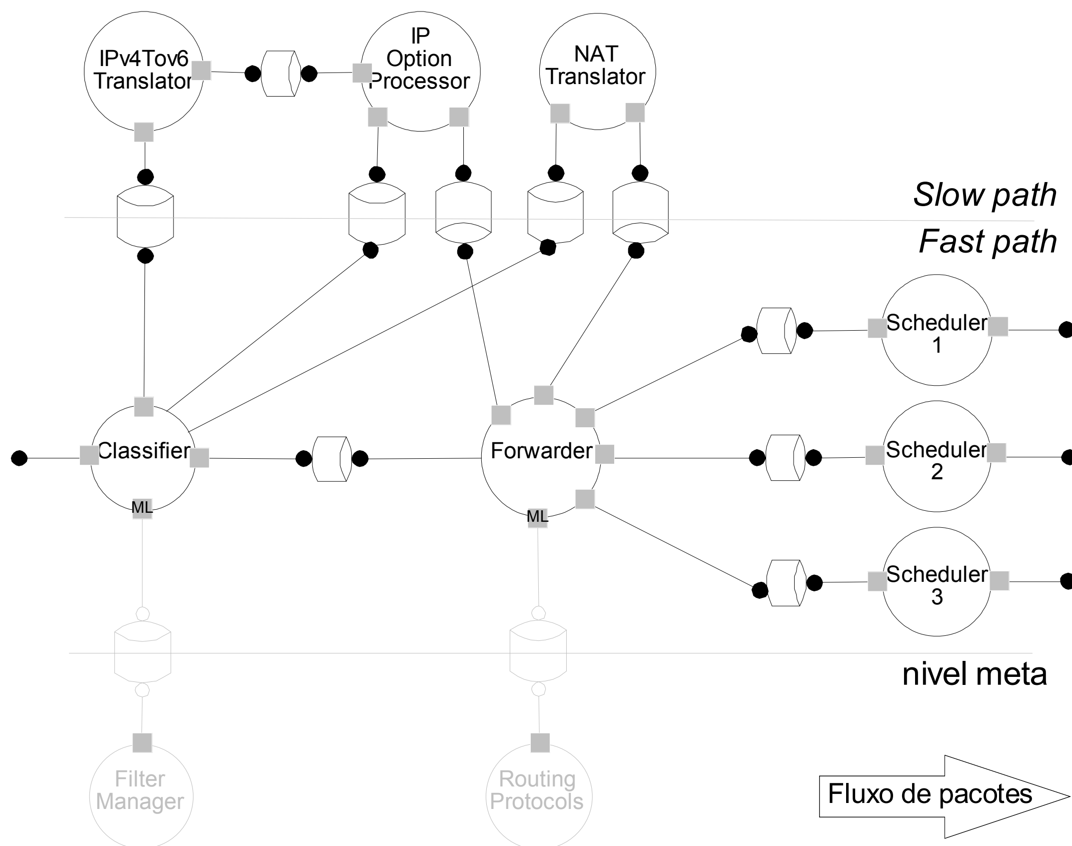


Figura 4.7. Configuração de encaminhamento IP em um roteador.

escalonamento. Outros componentes, executando em outro processador na NPU, porém fora do *fast path*, compõem o resto da configuração do plano de dados. Esse segundo grupo de componentes é responsável pelo processamento do campo de opções dos pacotes IP, pelo mascaramento de endereços IP (*Network Address Translation* – NAT) e pela tradução entre versões do protocolo IP. A configuração é complementada por metacomponentes executando fora da NPU (por exemplo, na CPU do sistema computacional onde se hospeda o roteador), e que são responsáveis pela configuração dos filtros de pacotes e da tabela de encaminhamento.

A Figura 4.8 apresenta a descrição, em LindaRouter, da arquitetura correspondente à Figura 4.7. A descrição inclui somente os componentes atuantes no plano de dados (*slow path* e *fast path*) do roteador. Considerações acerca da semântica dessa descrição são feitas a seguir.

Primeiro, a ordem na declaração dos componentes acoplados a um *pipe* em LindaRouter tem significado semântico para as ferramentas extensoras ligadas a

essa DSL. Essas ferramentas interpretam essa ordem como uma indicação que o primeiro componente requer dos componentes seguintes (relembrando que um *pipe* pode ser multiponto) uma interface do tipo referenciado pelo tipo de *pipe* de interligação. Assim, na descrição do sistema `FastPath` (que representa a configuração dos componentes no *fast path* – linhas 1 a 20) o componente `classifier` requer do componente `forwarder` uma interface de envio (`PPacketPush`) de pacotes (linha 10). Já os componentes `scheduler1`, `scheduler2` e `scheduler3` requerem do componente `forwarder` interfaces de solicitação (`PPacketPull`) de pacotes (linhas 11 a 13). Note, portanto, que o componente `forwarder` é totalmente passivo no exemplo (vide Figura 4.3). É interessante observar também que os tipos de *pipe* são parametrizados segundo diferentes argumentos em cada uma das declarações de sistema que constituem a configuração. Isso porque a real implementação do *pipe* vai depender fundamentalmente do ambiente de execução de cada parte da configuração. Desse modo, os *pipes* que interligam componentes no *fast path* devem ter uma implementação muito mais eficiente que aqueles que ligam componentes no *fast path* a componentes no *slow path*. Por exemplo, um simples desvio incondicional entre componentes localizados em um mesmo processador de manipulação de pacotes pode ser usado no *fast path*. Já no *slow path*, implementações de mecanismos de passagem de pacotes baseadas em memória compartilhada podem ser necessárias.

Em segundo lugar, mapeamentos de constituintes de uma configuração em LindaRouter são interpretados pelas ferramentas extensoras como exteriorizações de portas que requerem ou oferecem interfaces do tipo `PPacketPush` ou `PPacketPull`. Dessa forma, na descrição do sistema `ForwarderSys` (que representa a configuração completa do software de encaminhamento IP – linhas 36 a 62), o *pipe* `cls_to_ipv` declara (linhas 46 a 48), de fato, uma ligação entre uma porta de requisição de serviço *push* no componente `classifier` (residente no *fast path*) e uma porta equivalente de provisão no componente `ipv4tov6` (residente no *slow path*, que é descrito pelo sistema `SlowPath` – linhas 22 a 34).

Finalmente, é importante destacar que as ferramentas extensoras de LindaRouter não inferem a existência de constituintes implícitos em uma descrição de configuração. Isto é, todos os componentes que se quer presentes em uma configuração devem ser explicitamente declarados em LindaRouter.

PUC-Rio - Certificação Digital Nº 0115599/CA

```
1  System FastPath( cls_impl,fwd_impl,sch_impl,push_impl ) {
2    Style = #NPUBasedForwarder;
3
4    Instance classifier { Type = #UClassifier( #cls_impl ); }
5    Instance forwarder  { Type = #UClassifier( #fwd_impl ); }
6    Instance scheduler1 { Type = #UProcessor( #sch_impl ); }
7    Instance scheduler2 { Type = #UProcessor( #sch_impl ); }
8    Instance scheduler3 { Type = #UProcessor( #sch_impl ); }
9
10   Pipe[#classifier,#forwarder] { Type = #PushPipe( #push_impl ); }
11   Pipe[#scheduler1,#forwarder] { Type = #PullPipe( #pull_impl ); }
12   Pipe[#scheduler2,#forwarder] { Type = #PullPipe( #pull_impl ); }
13   Pipe[#scheduler3,#forwarder] { Type = #PullPipe( #pull_impl ); }
14
15   Mapping cls: #classifier;
16   Mapping fwd: #forwarder;
17   Mapping sch1: #scheduler1;
18   Mapping sch2: #scheduler2;
19   Mapping sch3: #scheduler3;
20 }
21
22 System SlowPath( ipv_impl,ipo_impl,nat_impl,push_impl ) {
23   Style = #PCBasedForwarder;
24
25   Instance ipv4tov6 { Type = #UProcessor( ipv_impl ); }
26   Instance ipoption { Type = #UProcessor( ipo_impl ); }
27   Instance nattrans { Type = #UProcessor( nat_impl ); }
28
29   Pipe[#ipv4tov6,#ipoption] { Type = #PushPipe( #push_impl ); }
30
31   Mapping ipv: #ipv4toipv6;
32   Mapping ipo: #ipoption;
33   Mapping nat: #nattrans;
34 }
35
36 System ForwarderSys( xplane_push_impl ) {
37   Style = #GenericForwarder;
38
39   Include fp: #FastPath;
40   Include sp: #SlowPath;
41
42   Pipe cls_to_ipv[#FastPath.cls,#SlowPath.ipv] {
43     Type = #PushPipe( #xplane_push_impl );
44   }
45   Pipe cls_to_ipo[#FastPath.cls,#SlowPath.ipo] {
46     Type = #PushPipe( #xplane_push_impl );
47   }
48   Pipe cls_to_nat[#FastPath.cls,#SlowPath.nat] {
49     Type = #PushPipe( #xplane_push_impl );
50   }
51   Pipe ipo_to_fwd[#SlowPath.ipo,#FastPath.fwd] {
52     Type = #PushPipe( #xplane_push_impl );
53   }
54   Pipe nat_to_fwd[#SlowPath.nat,#FastPath.fwd] {
55     Type = #PushPipe( #xplane_push_impl );
56   }
57
58   Mapping cls: #FastPath.cls;
59   Mapping sch1: #FastPath.sch1;
60   Mapping sch2: #FastPath.sch2;
61   Mapping sch3: #FastPath.sch3;
62 }
```

Figura 4.8. Descrição de arquitetura em LindaRouter de um roteador baseado em NPU.

PUC-Rio - Certificação Digital Nº 0115599/CA

4.2. A DSL LindaQoS

A DSL LindaQoS oferece uma semântica própria para a especificação de arquiteturas de provisão de QoS em sistemas de comunicação. A característica “fim-a-fim” dos mecanismos de provisão de QoS os torna particularmente interessantes para avaliar a expressividade do arcabouço sintático de LindaX.

Em (Gomes, 1999; Gomes et al., 2001a) é ilustrado, por meio de um conjunto de *frameworks* de software, que um modelo de provisão de QoS para um sistema de comunicação pode ser estruturado hierarquicamente, onde cada nível é responsável por prover garantias de QoS relativamente a uma determinada parte do sistema.[4] Essa mesma estrutura pode se repetir internamente em cada um desses níveis. Dessa forma, um mesmo modelo de provisão de QoS pode ser aplicado ubiquamente em todo o sistema, de modo recursivo. Os estilos associados à DSL LindaQoS são representações semi-formais dos *frameworks* de provisão de QoS apresentados em (Gomes, 1999; Gomes et al., 2001a). Antes de se apresentar a estrutura da DSL LindaQoS e dos estilos correspondentes, é feita uma apresentação concisa da arquitetura hierárquica de provisão de QoS definida por esses *frameworks*.

4.2.1. Arquiteturas de provisão de QoS

Os *frameworks* de provisão de QoS modelam mecanismos que atuam, segundo a nomenclatura do modelo SCM, como metassistemas responsáveis pela gerência de QoS em um sistema de comunicação. Esses mecanismos agem em duas fases distintas: *negociação* e *sintonização de QoS*.

A fase de negociação envolve mecanismos responsáveis pela admissão de novas atividades de computação e comunicação em um sistema. A reserva e alocação de recursos são os resultados principais dessa fase. Para isso, há um mecanismo de controle de admissão que permite estabelecer ‘acordos de serviço’ entre usuários e fornecedores de serviços (seguindo a nomenclatura adotada no Capítulo 2). O mecanismo de controle de admissão exerce, portanto, o papel de

[4]Note que a noção de “nível” em uma arquitetura hierárquica de provisão de QoS pode não ter relação alguma com o conceito de nível/camada do modelo OSI. Nessa arquitetura, um nível pode se referir, por exemplo, a mecanismos de provisão de QoS atuando no sistema operacional, ou mesmo internamente a um processo.

PUC-Rio - Certificação Digital Nº 0115599/CA

uma interface de solicitação de serviços que possibilita ao usuário definir parâmetros para a QoS desejada. Outro mecanismo importante nessa fase é o de negociação de QoS. Esse mecanismo tem como função identificar quais partes do sistema estarão envolvidas na provisão do serviço requisitado e definir, para cada uma dessas partes, qual a responsabilidade parcial de cada uma delas na provisão da QoS fim-a-fim requisitada pelo usuário. Nesse processo, é necessária também a atuação de um mecanismo que permita mapear a visão de QoS do usuário – isto é, os parâmetros de QoS passados pelo usuário ao fornecedor – na visão de QoS de cada uma das partes do sistema.

É interessante notar o caráter recursivo do processo de estabelecimento de acordos de serviço: os mecanismos de negociação atuam de fato como usuários dos mecanismos de provisão de QoS presentes em cada uma das partes que compõem o sistema. O último passo de recursão desse processo ocorre quando os mecanismos de controle de admissão atuam diretamente na alocação de recursos computacionais e de comunicação – em termos de modelo SCM, na criação de folhas na árvore de recursos virtuais (Capítulo 2).

A fase de sintonização provê mecanismos responsáveis por monitorar o uso de recursos por parte de um usuário após o estabelecimento de um acordo de serviço. Em caso de violações no acordo (um fluxo consumindo mais largura de banda que o acordado pelo usuário, por exemplo), determinadas ações podem ser tomadas pelos mecanismos de sintonização. Dentre essas ações, inclui-se notificação, readequação do usuário (por exemplo, redução de prioridade de uma *thread* mal-comportada, moldagem de um fluxo não-conforme), reorquestração de recursos (quando o fornecedor é responsável por uma "falsa" violação) e interrupção do serviço. No caso da readequação de usuários e da reorquestração de recursos, o processo é similar àquele de negociação de QoS, podendo ocorrer, porém, durante a provisão do serviço e sem intervenção ou notificação direta do usuário.

Nesta tese, somente os mecanismos atuantes durante a fase de negociação de QoS foram vislumbrados pela DSL LindaQoS. A especificação de estilos e ferramentas extensoras para os mecanismos atuantes na fase de sintonização foi deixada como trabalho futuro. Note, contudo, que o princípio de hierarquização de mecanismos se aplica nessa fase de modo muito similar à fase de negociação. Deduz-se, portanto, que a especificação de mecanismos de sintonização em muito se assemelhará à apresentada nesta seção.

PUC-Rio - Certificação Digital Nº 0115599/CA

A natureza recursiva das arquiteturas de provisão de QoS tem impacto direto na concepção da DSL LindaQoS, que privilegia justamente especificações aninhadas dos mecanismos presentes nessas arquiteturas. Contudo, cada um desses níveis de aninhamento pode seguir um padrão arquitetural diferente – por exemplo, em um determinado nível o mecanismo de negociação pode ser distribuído, enquanto no nível imediamente abaixo pode haver um metacomponente que centraliza a negociação. Essa variedade de possíveis configurações se reflete na estrutura dos estilos ligados à LindaQoS.

4.2.2. Estilos de LindaQoS

Quatro estilos foram definidos para a especificação de arquiteturas hierárquicas de negociação de QoS: `LowestNQoS`, `CentralizedNQoS`, `DistributedNQoS` e `HierarchyNQoS`. Ao contrário dos estilos de LindaRouter, os estilos apresentados nesta seção não apresentam relação de herança alguma entre si. Além disso, não é prevista neste trabalho a criação de novos estilos para LindaQoS. A Figura 4.9 ilustra de modo diagramático o vocabulário e restrições que compõem cada um desses estilos.

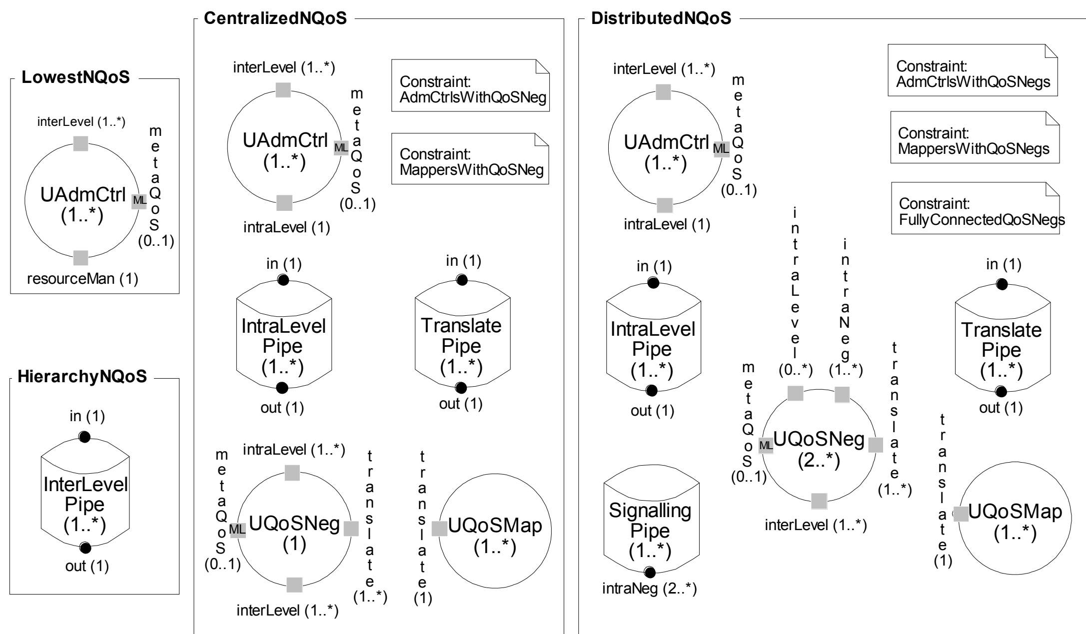


Figura 4.9. Notação diagramática dos estilos de negociação em LindaQoS.

PUC-Rio - Certificação Digital Nº 0115599/CA

O estilo `LowestNQoS` descreve os níveis inferiores de uma arquitetura hierárquica de negociação de QoS, onde o controle de admissão é feito sobre recursos diretamente. Nesses níveis os mecanismos de controle de admissão (tipo de componente `UAdmCtrl`) são os únicos metacomponentes presentes. Eles fazem a ligação entre o metasserviço e o serviço principal, por meio da porta `resourceMan`. Nesse estilo não são definidas restrições de configuração.

É importante notar o papel crucial que esse estilo tem em relação à integração de LindaQoS e LindaRouter. A porta `resourceMan` está sempre associada a uma porta ML que provê serviços de gerência de recursos ou dá acesso a mecanismos de adaptação no serviço principal. No último caso, essa porta ML será, tipicamente, aquela provida por componentes do tipo `UClassifier`, ilustrado na Figura 4.5 (vide linha 13 nessa figura). Contudo, as ferramentas de refinamento e síntese (vide Capítulo 5) não contemplam, em sua implementação atual, a integração automática de descrições em LindaQoS e LindaRouter. Essa questão é levantada novamente no Capítulo 7.

O estilo `CentralizedNQoS` descreve os níveis de uma arquitetura de negociação em que o mecanismo de negociação é centralizado em um único metacomponente (tipo `UQoSNeg`). A Figura 4.10 ilustra as restrições desse estilo.

Arquiteturas de QoS baseadas no serviço de gerência de políticas COPS (Durham et al., 2000) são um bom exemplo de onde esse estilo pode ser aplicado. Nessas arquiteturas, o metacomponente central de negociação pode receber – através de *pipes* do tipo `IntraLevelPipe` – pedidos de novos acordos de serviço de um ou mais metacomponentes de controle de admissão (tipo `UAdmCtrl`) e encaminhar esses pedidos – através de *pipes* do tipo `InterLevelPipe` – a metacomponentes de controle de admissão responsáveis por outras partes do sistema. A cada um desses metacomponentes de controle de admissão "externos" está associado um metacomponente de mapeamento (tipo `UQoSMap`) responsável pela tradução das visões de QoS entre os dois níveis do metassistema. Esse metacomponente se liga ao metacomponente de negociação por meio de *pipes* do tipo `TranslatePipe`. Esse estilo define restrições em relação à topologia de ligação entre os metacomponentes de controle de admissão e de negociação (que deve seguir a forma de uma estrela tendo como centro o metacomponente de negociação) e entre os metacomponentes de negociação e de mapeamento.

O estilo `DistributedNQoS` descreve os níveis de uma arquitetura de negociação em que o mecanismo de negociação é distribuído entre vários

PUC-Rio - Certificação Digital Nº 0115599/CA

```
1  Style CentralizedNQoS {
2    ... // vocabulário do estilo
3
4    // Todo metacomponente de mapeamento
5    // deve estar ligado a um metacomponente de negociação central.
6    FolPredicate MappersWithQoSNeg {
7      exists n : Components( #UQoSNeg ); {
8        forall c : Components( #UQoSMap ); {
9          exists pn : Ports( #n );
10                pc : Ports( #c ); {
11           exists s : Signatures( #pc ); {
12             [Pipe( #n,#pn,#c,#pc ) in Pipes()] and
13             [PropertyValue( #s,"Type" ) == #PTranslate]
14           }
15         }
16       }
17     }
18   }
19
20   // Todo metacomponente de controle de admissão
21   // deve estar ligado a um metacomponente de negociação central.
22   FolPredicate AdmCtrlsWithQoSNeg {
23     exists n : Components( #UQoSNeg ); {
24       forall c : Components( #UAdmCtrl ); {
25         exists pc : Ports( #c );
26                pn : Ports( #n ); {
27           exists s : Signatures( #pn ); {
28             [Pipe( #c,#pc,#n,#pn ) in Pipes()] and
29             [PropertyValue( #s,"Type") == #PIntraLevel]
30           }
31         }
32       }
33     }
34   }
35 }
```

Figura 4.10. Estilo `CentralizedNQoS`.

metacomponentes, como em sistemas de sinalização baseados no protocolo RSVP (Braden et al., 1997), por exemplo. Um desses metacomponentes de negociação – tipicamente, mas não necessariamente, residente em um sistema final – pode receber pedidos de novos acordos de serviço de metacomponentes de controle de admissão, distribuir esses pedidos a outros metacomponentes de negociação por meio de *pipes* do tipo `SignallingPipe` (que descreve o protocolo de sinalização) e encaminhar esses pedidos (após o devido mapeamento entre as visões de QoS) a metacomponentes de controle de admissão responsáveis por outras partes do sistema, de modo similar ao descrito no estilo `CentralizedNQoS`. Esse estilo também define restrições em relação à topologia de ligação entre os metacomponentes de controle de admissão e negociação (que deve seguir a forma de um grafo conexo) e entre os metacomponentes de negociação e de mapeamento. A Figura 4.11 apresenta a descrição das restrições desse estilo.

Finalmente, o estilo `HierarchyNQoS` provê a ligação entre níveis de uma arquitetura hierárquica de provisão de QoS. Esse estilo define o tipo de *pipe* `InterLevelPipe`, que regula a interação entre metacomponentes de negociação e

PUC-Rio - Certificação Digital Nº 0115599/CA

```
1  Style DistributedNQoS {
2    ... // vocabulário do estilo
3
4    // Todo metacomponente de mapeamento
5    // deve estar ligado a um metacomponente de negociação.
6    FolPredicate MappersWithQoSNegs {
7      forall c : Components( #UQoSMap ); {
8        exists n : Components( #UQoSNeg ); {
9          exists pn : Ports( #n );
10                pc : Ports( #c ); {
11           exists s : Signature( #pc ); {
12             [Pipe( #n,#pn,#c,#pc ) in Pipes()] and
13             [PropertyValue( #s,"Type" ) == #PTranslate]
14           }
15         }
16       }
17     }
18   }
19
20   // Todo metacomponente de controle de admissão
21   // deve estar ligado a um metacomponente de negociação.
22   FolPredicate AdmCtrlsWithQoSNegs {
23     forall c : Components( #UAdmCtrl ); {
24       exists n : Components( #UQoSNeg ); {
25         exists pc : Ports( #c );
26                pn : Ports( #n ); {
27           exists s : Signature( #pn ); {
28             [Pipe( #c,#pc,#n,#pn ) in Pipes()] and
29             [PropertyValue( #s,"Type" ) == #PIntraLevel]
30           }
31         }
32       }
33     }
34   }
35
36   // Restrição de grafo conexo, utilizando pipes do tipo
37   // SignallingPipe, entre metacomponentes de negociação.
38   // SE o grafo de componentes é conexo, haverá sempre um conjunto
39   // com todos os componentes na configuração, ordenado do componente
40   // com menos associações ao componente com mais associações,
41   // que obedece a esse predicado.
42   FolPredicate FullyConnectedQoSNegs {
43     exists s : SequenceSet( Components( #UQoSNeg ) )
44                 where
45                   [Cardinality( #s ) > 1] and
46                   [Cardinality( #s ) ==
47                      Cardinality( Components( #UQoSNeg ) )]; {
48       forall i : { 1..Cardinality( #s )-1 }; {
49         exists j : { #i+1..Cardinality( #s ) }; {
50           exists pni : Ports( At( #s,#i ) );
51                  pnj : Ports( At( #s,#j ) ); {
52             [Pipe( At( #s,#i ),#pni,At( #s,#j ),#pnj )
53                in Pipes( #SignallingPipe )]
54           }
55         }
56       }
57     }
58   }
59 }
```

Figura 4.11. Estilo `DistributedNQoS`.

de controle de admissão presentes em níveis distintos da arquitetura. Nesse estilo não são definidas restrições de configuração. Esse estilo tem função similar ao estilo `GenericForwarder`, definido para LindaRouter, de suprir a ausência em LindaX de um suporte a estilos recursivos.

PUC-Rio - Certificação Digital Nº 0115599/CA

4.2.3. Semântica de LindaQoS

Pelo fato dos estilos de LindaQoS possuírem vocabulários bastante extensos e detalhados, pode-se manter nas ferramentas extensoras associadas a cada estilo uma quantidade considerável de informação acerca da semântica de configurações nessa DSL. Isso permite descrições mais concisas de configurações, porém às custas de uma menor flexibilidade da linguagem. Essa decisão de projeto leva em conta o tamanho relativamente extenso de uma descrição típica de arquitetura de provisão de QoS, que pode envolver configurações em diversos níveis de aninhamento e com graus variados de distribuição.

A configuração a ser utilizada nesta seção para ilustrar o uso de LindaQoS é representada diagramaticamente na Figura 4.12 (por questão de legibilidade, a notação para representação de portas e pontos de acesso não é usada na figura).

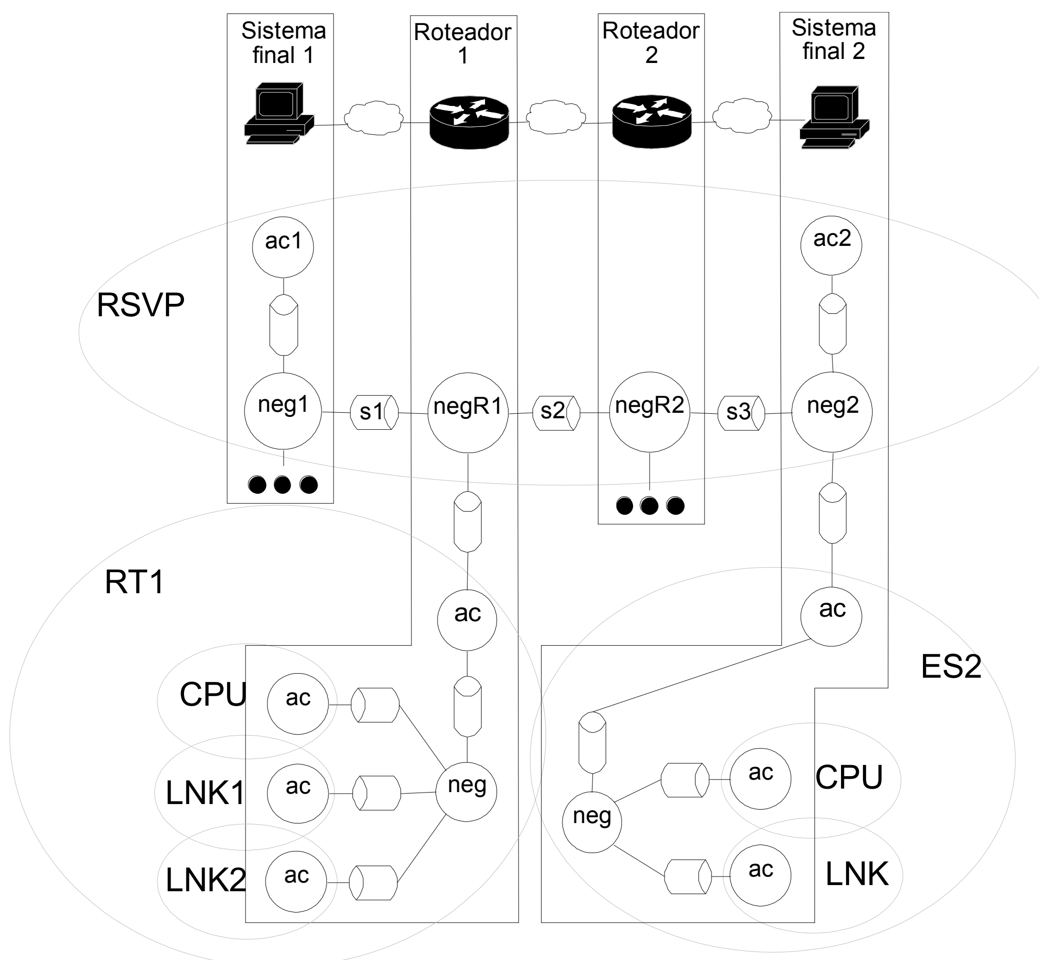


Figura 4.12. Configuração de uma arquitetura de provisão de QoS distribuída.

PUC-Rio - Certificação Digital Nº 0115599/CA

A figura acima ilustra uma configuração de negociação de QoS envolvendo dois sistemas finais e dois roteadores. A negociação de QoS em seu nível mais alto ocorre de forma distribuída, utilizando o protocolo RSVP como plataforma de sinalização. Os metacomponentes de negociação que fazem uso do RSVP coordenam os mecanismos atuantes no sistema operacional de cada uma das estações e roteadores. O sistema operacional de cada uma dessas máquinas tem um subsistema de negociação de QoS próprio, que gerencia dois tipos de recursos principais: tempo de CPU e largura de banda. Para não "poluir" excessivamente a figura, os metacomponentes de mapeamento que se acoplam aos metacomponentes de negociação não foram representados. Omitiu-se também a configuração dos mecanismos que atuam no primeiro sistema final e no segundo roteador, haja visto que ela é idêntica a dos outros sistemas representados.

As Figuras 4.13 e 4.14 apresentam a descrição, em LindaQoS, da arquitetura correspondente à Figura 4.12. A Figura 4.13 apresenta a descrição de *templates* para mecanismos de negociação em sistemas finais e roteadores. A Figura 4.14 descreve instâncias desses *templates* no contexto da arquitetura de sinalização ilustrada. Considerações sobre a semântica dessa descrição são feitas a seguir.

Primeiro, grande parte dos *pipes* descritos em ambas as figuras não são adornados com propriedade alguma. Em particular, o tipo de cada um desses *pipes* (que seria, a princípio, referenciado por meio de uma propriedade `Type`) pode ser inferido pelas ferramentas extensoras associadas a LindaQoS, a partir dos tipos dos componentes que se acoplam ao *pipe*. O mesmo pode ser dito sobre as portas que os componentes usam nesses acoplamentos. Logo, a referência aos tipos desses *pipes* é desnecessária em um primeiro momento. Além disso, os tipos desses *pipes* são definidos nos estilos de LindaQoS para refletir a restrição, imposta pelo modelo da linguagem, que componentes devem obrigatoriamente se comunicar por meio de *pipes*, haja vista que a semântica desses *pipes* é a de uma simples requisição de operação. No caso das ferramentas de síntese, pode-se assumir que a implementação desses *pipes* é a mais simples existente em uma plataforma de programação – uma chamada de operação em uma interface de componente, por exemplo. Porém, no caso de refinamento para uma ADL específica, a referência ao tipo do *pipe* pode se tornar necessária.

Exceções à observação acima são os *pipes* `s1`, `s2` e `s3` da Figura 4.14 (linhas 18 a 20). Esses *pipes* são do tipo `SignallingPipe`, que representa o protocolo de sinalização utilizado na comunicação entre metacomponentes de negociação. No

PUC-Rio - Certificação Digital Nº 0115599/CA

caso desses *pipes*, a implementação associada não será tão trivial quanto uma chamada de operação. Por isso, a ferramenta extensora associada ao estilo `DistributedNQoS` obriga a definição da propriedade `Type` para *pipes* usados na ligação entre dois ou mais metacomponentes de negociação.

```
1  SystemTemplate CPUManager( ac_impl ) {
2    Style = #LowestNQoS;
3
4    Instance ac { Type = #UAdmCtrl( #ac_impl ); }
5
6    Mapping ac : #ac;
7  }
8
9  SystemTemplate LinkManager( ac_impl ) {
10   Style = #LowestNQoS;
11
12   Instance ac{ Type = #UAdmCtrl( #ac_impl ); }
13
14   Mapping ac : #ac;
15 }
16
17 SystemTemplate OS( ac_impl,neg_impl,map1_impl,map2_impl ) {
18   Style = #CentralizedNQoS;
19
20   Instance ac { Type = #UAdmCtrl( #ac_impl ); }
21   Instance neg { Type = #UQoSNeg( #neg_impl );
22                 Mappers = { { #map1_impl },{ #map2_impl } }; }
23
24   Pipe[#ac,#neg];
25
26   Mapping ac : #ac;
27   Mapping neg : #neg;
28 }
29
30 SystemTemplate Router( cpu_ac_impl,lnk1_ac_impl,lnk2_ac_impl,
31                        os_ac_impl,os_neg_impl,
32                        os_map1_impl,os_map2_impl ) {
33   Style = #HierarchyNQoS;
34
35   Instance cpu  { Type = #CPUManager( #cpu_ac_impl ); }
36   Instance lnk1 { Type = #LinkManager( #lnk1_ac_impl ); }
37   Instance lnk2 { Type = #LinkManager( #lnk2_ac_impl ); }
38   Instance os   { Type = #OS( #os_ac_impl,#os_neg_impl,
39                               #os_map1_impl,#os_map2_impl ); }
40
41   Pipe[#os.neg,#cpu.ac];
42   Pipe[#os.neg,#lnk1.ac];
43   Pipe[#os.neg,#lnk2.ac];
44
45   Mapping ac : #os.ac;
46 }
47
48 SystemTemplate EndSys( cpu_ac_impl, lnk_ac_impl,
49                        os_ac_impl, os_neg_impl,
50                        os_map1_impl,os_map2_impl ) {
51   Style = #HierarchyNQoS;
52
53   Instance cpu { Type = #CPUManager( #cpu_ac_impl ); }
54   Instance lnk { Type = #LinkManager( #lnk_ac_impl ); }
55   Instance os  { Type = #OS( #os_ac_impl,#os_neg_impl,
56                              #os_map1_impl,#os_map2_impl ); }
57
58   Pipe[#os.neg,#cpu.ac];
59   Pipe[#os.neg,#lnk.ac];
60
61   Mapping ac : #os.ac;
62 }
```

Figura 4.13. Descrição de *templates* para sistemas finais e roteadores em LindaQoS.

PUC-Rio - Certificação Digital Nº 0115599/CA

```
1  System RSVPNet( ac_impl,neg_impl,negR_impl,
2                  map1_impl,map2_impl,sig_impl ) {
3    Style = #DistributedNQoS;
4
5    Instance ac1   { Type = #UAdmCtrl( #ac_impl ); }
6    Instance ac2   { Type = #UAdmCtrl( #ac_impl ); }
7    Instance neg1  { Type = #UQoSNeg( #neg_impl );
8                     Mappers = { { #map1_impl },{ #map2_impl } }; }
9    Instance neg2  { Type = #UQoSNeg( #neg_impl ); }
10                    Mappers = { { #map1_impl },{ #map2_impl } }; }
11   Instance negR1 { Type = #UQoSNeg( #negR_impl ); }
12                    Mappers = { { #map1_impl },{ #map2_impl } }; }
13   Instance negR2 { Type = #UQoSNeg( #negR_impl ); }
14                    Mappers = { { #map1_impl },{ #map2_impl } }; }
15
16   Pipe[#ac1,#neg1];
17   Pipe[#ac2,#neg2];
18   Pipe s1[#neg1,#negR1]  { Type = #SignallingPipe( #sig_impl ); }
19   Pipe s2[#neg2,#negR2]  { Type = #SignallingPipe( #sig_impl ); }
20   Pipe s3[#negR1,#negR2] { Type = #SignallingPipe( #sig_impl ); }
21
22   Mapping ac1 : #ac1;
23   Mapping ac2 : #ac2;
24   Mapping neg1 : #neg1;
25   Mapping neg2 : #neg2;
26   Mapping negR1 : #negR1;
27   Mapping negR2 : #negR2;
28 }
29
30 System QoSSystem( es1_cpu_ac_impl,es1_lnk_ac_impl,
31                   es1_os_ac_impl,es1_os_neg_impl,
32                   es2_cpu_ac_impl,es2_lnk_ac_impl,
33                   es2_os_ac_impl, es2_os_neg_impl,
34                   rt1_cpu_ac_impl,rt1_lnk1_ac_impl,
35                   rt1_lnk2_ac_impl,rt1_os_ac_impl,rt1_os_neg_impl,
36                   rt2_cpu_ac_impl,rt2_lnk1_ac_impl,
37                   rt2_lnk2_ac_impl,rt2_os_ac_impl, rt2_os_neg_impl ) {
38   Style = #HierarchyNQoS;
39
40   Instance es1 { Type = #EndSys( #es1_cpu_ac_impl,#es1_lnk_ac_impl,
41                                  #es1_os_ac_impl,#es1_os_neg_impl ); }
42   Instance es2 { Type = #EndSys( #es2_cpu_ac_impl,#es2_lnk_ac_impl,
43                                  #es2_os_ac_impl,#es2_os_neg_impl ); }
44   Instance rt1 { Type = #Router( #rt1_cpu_ac_impl,#rt1_lnk1_ac_impl,
45                                  #rt1_lnk2_ac_impl,#rt1_os_ac_impl,
46                                  #rt1_os_neg_impl ); }
47   Instance rt2 { Type = #Router( #rt2_cpu_ac_impl,#rt2_lnk1_ac_impl,
48                                  #rt2_lnk2_ac_impl,#rt2_os_ac_impl,
49                                  #rt2_os_neg_impl ); }
50
51   Include rsvp: #RSVPNet;
52
53   Pipe[#rsvp.neg1,#es1.ac];
54   Pipe[#rsvp.neg2,#es2.ac];
55   Pipe[#rsvp.negR1,#rt1.ac];
56   Pipe[#rsvp.negR2,#rt2.ac];
57 }
```

Figura 4.14. Descrição de arquitetura em LindaQoS de um sistema de sinalização RSVP.

A segunda consideração acerca de descrições em LindaQoS diz respeito à exteriorização de constituintes de configurações. As ferramentas extensoras interpretam as cláusulas `Mapping` como exteriorizações de portas acopláveis aos pontos de acesso de *pipes* do tipo `InterLevelPipe`. Nas arquiteturas descritas em LindaQoS, somente as portas de nível base `interLevel` dos metacomponentes de controle de admissão e de negociação são exteriorizáveis.

PUC-Rio - Certificação Digital Nº 0115599/CA

Finalmente, as ferramentas extensoras de LindaQoS associadas aos estilos `CentralizedNQoS` e `DistributedNQoS` inferem a existência de metacomponentes de mapeamento em configurações baseadas nesses estilos. A propriedade `Mappers` é usada junto a descrições de instâncias do tipo `UQoSNeg` para indicar que um ou mais metacomponentes de mapeamento estão ligados ao metacomponente de negociação correspondente por meio de *pipes* do tipo `TranslatePipe`. Essa propriedade define uma lista cuja cardinalidade indica exatamente quantos metacomponentes de mapeamento estão presentes na configuração. Cada elemento dessa lista representa o conjunto de parâmetros passados como argumentos para o tipo `UQoSMap`.

4.3. Sumário

Este capítulo apresentou duas DSLs especializadas a partir do arcabouço sintático de LindaX. Embora o objetivo tenha sido de fato demonstrar a exeqüibilidade da abordagem proposta, as DSLs citadas constituem-se também em duas contribuições secundárias da tese presente. Uma consideração importante é o quão difícil pode ser para um projetista de DSL especializar o arcabouço sintático de LindaX em outros domínios. Se por um lado a especificação do vocabulário e das restrições arquiteturais para uma nova DSL não apresenta maiores dificuldades, quando comparado a outras linguagens com suporte a estilos (Allen, 1997; Monroe, 1998), por outro lado a implementação de uma ferramenta extensora específica para essa DSL pode ser relativamente complexa. Um *framework* básico para a implementação dessas ferramentas é oferecido no ambiente LindaStudio (vide próximo capítulo). Fora isso, o ambiente não oferece suporte para automatizar (mesmo que parcialmente) novas ferramentas extensoras a partir de especificações de estilos. Essa questão é discutida novamente no Capítulo 7.

PUC-Rio - Certificação Digital Nº 0115599/CA