Uma Abordagem de Especificação e Imposição de Restrições de Reconfiguração em Redes Programáveis

Antônio Tadeu Azevedo Gomes[1, 2], Luiz Fernando Gomes Soares[2]

[1] Coordenação de Sistemas e Redes – LNCC
Av. Getúlio Vargas, 333 – CEP 25.651-075 – Petrópolis – RJ – Brasil

[2] Departamento de Informática – PUC-Rio
R. Marquês de São Vicente, 225 – CEP 22.453-900 – Rio de Janeiro – RJ – Brasil

atagomes@lncc.br, lfgs@inf.puc-rio.br

Abstract. *This paper describes a process for the automatic synthesis of configurations and reconfiguration controlling mechanisms for programmable network architectures. The process is centred around principles from Software Architecture and Component-Based Development. This paper shows how this process is applied from an architecture style-based specification language and a component-based programming environment that can enforce reconfiguration constraints through reflective facilities.*

Resumo. *Este artigo descreve um processo de síntese automática de configurações e de mecanismos de controle de reconfigurações para arquiteturas de rede programável. O processo é centrado em princípios de Arquitetura de Software e de Desenvolvimento Baseado em Componentes. Este artigo mostra como esse processo é aplicado a partir de uma linguagem de especificação baseada em estilos arquiteturais e de um ambiente de programação baseado em componentes que permite impor restrições de reconfiguração via facilidades de reflexão.*

1. Introdução

No cenário atual do setor de telecomunicações percebe-se uma tendência crescente no uso de sistemas de comunicação que permitam a criação rápida e de baixo custo de serviços. Na busca por arquiteturas de rede que respondam a essa tendência, vários grupos têm centrado seus esforços em pesquisas na área de 'redes programáveis' (Campbell et al., 1999). O surgimento da tecnologia de 'processamento de rede' (Network Processing Forum, 2001) no mercado de equipamentos de telecomunicações abriu ainda maior espaço para pesquisas nessa área.

Nesse contexto, é imprescindível que o processo de criação de serviços seja bem estruturado e, o quanto possível, sistemático. Este trabalho insere-se dentro de um projeto em desenvolvimento no Laboratório TeleMídia da PUC-Rio[1] cuja meta é definir um ambiente de criação de serviços independente da arquitetura de rede programável em uso. O projeto adota uma abordagem em que técnicas de Arquitetura de Software e de Desenvolvimento Baseado em Componentes são aplicadas consistentemente e de modo ubíquo, desde especificações de alto nível de serviços até a implementação de software básico em NPUs (*Network Processing Units*).

O principal objetivo deste artigo é apresentar um processo, desenvolvido no projeto citado acima, que permite a síntese automática de configurações e de mecanismos de controle de reconfigurações para diferentes arquiteturas de rede programável, a partir de especificações arquiteturais de alto nível. Esse processo

---

[1] Subprojeto GIGA "Engenharia de Serviços e Aplicações com QoS", CPqD/RNP – FUNTTEL.

envolve dois elementos principais: (i) ambientes de programação baseados em componentes de software e (ii) uma linguagem de especificação de arquitetura. O processo de síntese em si, conforme proposto no projeto, é independente do ambiente de programação utilizado. Neste texto, porém, o processo é descrito em termos de um ambiente de programação específico, aplicado no contexto do projeto mencionado.

O ambiente de programação utilizado é o NetKit (Coulson et al., 2003), um conjunto de componentes especializado na configuração de redes programáveis. O NetKit não pressupõe nenhuma arquitetura ou NPU específica, logo é perfeitamente adequado ao objetivo deste trabalho. Além disso, o modelo de componentes subjacente ao NetKit oferece facilidades de reflexão (Maes, 1987) que permitem a inspeção e adaptação de sistemas de modo genérico e organizado, separando claramente a operação de um sistema e a gerência de reconfiguração do mesmo. Por fim, o NetKit emprega o conceito de 'framework de componentes' (Szyperski, 2002) para permitir a definição de composições de software que seguem regras de configuração de domínio específico.

A linguagem de especificação empregada é LindaX (Gomes et al., 2003), um formato declarativo baseado em XML que permite descrever a arquitetura de sistemas de comunicação bem como restrições de reconfiguração nesses sistemas. Descrições em LindaX podem ser refinadas em especificações em diferentes linguagens formais ou podem ser usadas na síntese de configurações para diversas plataformas.

O processo de síntese proposto é aplicado na construção sistemática de frameworks de componentes para o NetKit a partir de descrições em LindaX. Para esse fim, foi desenvolvido um mecanismo genérico de imposição de regras de configuração para o NetKit. O uso desse mecanismo em frameworks de componentes possibilita um alto reuso de projeto e implementação entre frameworks de componentes distintos. Além disso, esse mecanismo confere aos frameworks de componentes uma alta manutenibilidade – alterações nas regras de configuração desses frameworks podem ser feitas em tempo de execução por intermédio desse mecanismo. Dois outros mecanismos complementares – de configuração transacional e configuração em lote – também foram desenvolvidos como parte deste trabalho. Esses mecanismos permitem, respectivamente, manter a consistência de uma configuração frente a múltiplas reconfigurações e facilitar a tarefa de projetistas na implantação de novos serviços.

O restante do artigo está estruturado da seguinte forma. A Seção 2 apresenta o ambiente de programação do NetKit. O mecanismo genérico de imposição de regras de configuração e os mecanismos complementares de configuração transacional e em lote são introduzidos na Seção 3. A Seção 4 ilustra o processo de construção sistemática de frameworks de componentes para o NetKit a partir de LindaX. O ambiente experimental utilizado como prova de conceito da abordagem proposta é apresentado na Seção 5. Alguns trabalhos ligados a este artigo são relacionados na Seção 6, enquanto a Seção 7 é reservada a conclusões e trabalhos futuros.

2. O modelo de componentes do NetKit

O NetKit consiste em uma biblioteca de componentes especializada para redes programáveis. Essa biblioteca abrange todos os níveis presentes em uma arquitetura de rede programável, desde as funções de processamento de pacotes no 'plano de dados' – como classificadores, marcadores e escalonadores – até as funções de mais alto nível de sinalização e coordenação. Por meio das facilidades oferecidas pelo seu modelo de componentes, o OpenCOMv2, o NetKit permite uma alta flexibilidade na implantação,

instanciação e reconfiguração dessas funções em NPUs. Pelo fato dos mecanismos propostos neste artigo se apoiarem diretamente nos serviços OpenCOMv2, esta seção será mais dedicada à apresentação do modelo de componentes do que do NetKit em si.

O modelo OpenCOMv2 define seis conceitos principais (vide Figura 1): componentes, interfaces, receptáculos, cápsulas, associações locais e metamodelos.

Um componente OpenCOMv2 define um tipo para módulos de software univocamente identificado por um GUID (*General Universal IDentifier*). Componentes oferecem serviços a outros componentes por meio de interfaces e requisitam serviços a interfaces de outros componentes por meio de receptáculos. Interfaces são imutáveis e fortemente tipadas – o tipo de uma interface também é identificado por um GUID – podendo admitir três modos de interação: (i) invocações de operações; (ii) fluxos contínuos; e (iii) sinais. Receptáculos tornam explícitas possíveis dependências entre componentes. Assim, quando um componente é instanciado é possível determinar a partir de seus receptáculos se outras instâncias de componentes ou, mais precisamente, se interfaces de determinado tipo sendo oferecidas por outras instâncias, devem estar presentes para que a instância recém-criada possa funcionar corretamente.

Cápsulas delineiam escopos de carga (instanciação) e de associação local (conexão entre receptáculos e interfaces) para componentes. O objetivo principal das cápsulas é uniformizar a implantação de componentes em diferentes plataformas. Ao prover uma interface única de acesso a serviços de carga e associação de componentes, a cápsula esconde as peculiaridades de cada plataforma. O conceito de cápsula introduz assim uma separação de papéis entre projetistas. Projetistas 'de implantação' fazem a ponte entre o ambiente concreto de cada plataforma e o modelo de componentes, implementando os serviços de carga e associação nessas plataformas. Projetistas 'de sistemas' são isolados das peculiaridades de cada plataforma por meio desses serviços.

O nível de distribuição física de uma cápsula é arbitrário e dependente de plataforma. Em casos extremos pode-se considerar até mesmo uma cápsula como abrangendo diferentes máquinas ligadas em rede. Porém, o propósito original das cápsulas é que elas sejam 'fortemente acopladas', ou seja, o conceito de 'associação local' assume conexões entre receptáculos e interfaces que sejam obrigatoriamente confiáveis, determinísticas e de baixo retardo, como no caso de NPUs, de processos comunicantes em uma mesma máquina ou de redes embarcadas. O objetivo é viabilizar o uso de associações locais em comunicações que demandam alta vazão, como no

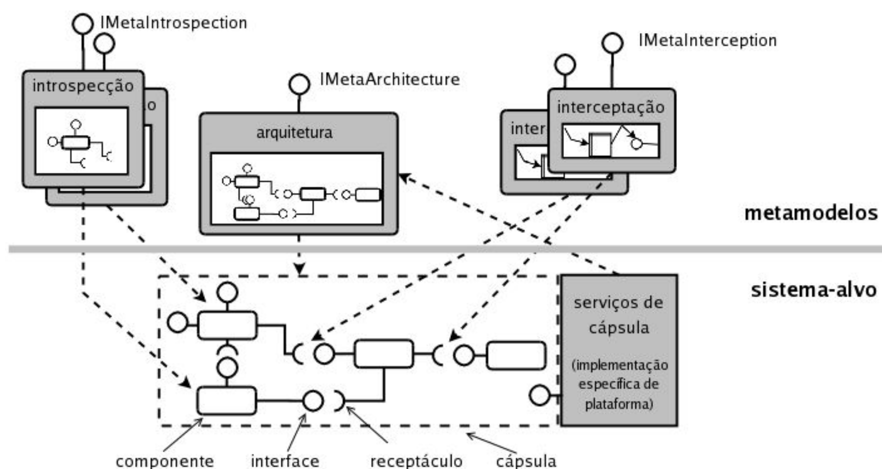

Figura 1. O modelo de componentes OpenCOMv2.

encaminhamento de pacotes no plano de dados. Associações remotas ou com semântica agregada (por exemplo, uma associação multiponto) são construídas no OpenCOMv2 sob a forma de composições (possivelmente distribuídas) de componentes.

Metamodelos são mecanismos de reflexão (Maes, 1987) – implementados no OpenCOMv2 como componentes – que permitem gerir, inspecionar e adaptar metadados de uma configuração. Metamodelos mantêm uma relação causal entre metadados e configurações, de modo que mudanças nos metadados – em geral ocorridas em face de operações requisitadas nas interfaces dos metamodelos (as 'metainterfaces') – se refletem automaticamente em mudanças na configuração e vice-versa. Os metamodelos independem de um suporte nativo a reflexão nas linguagens de programação usadas na confecção de componentes. Contudo, essa independência acarreta limitações na abordagem de controle de reconfiguração proposta neste artigo, quando linguagens não-reflexivas são utilizadas (vide Seção 7).

O OpenCOMv2 provê três metamodelos principais (vide Figura 1). O metamodelo de introspecção dá suporte a inspeção e adaptação limitada (isto é, exposição e oclusão) de interfaces e receptáculos. O metamodelo de arquitetura representa a topologia de uma configuração como um grafo que pode ser inspecionado ou alterado. A invocação dos serviços de carga e associação local das cápsulas também se reflete em alterações nesse grafo. Finalmente, o metamodelo de interceptação permite a associação de 'interceptadores' a operações, fluxos ou sinais de uma interface particular. Interceptadores são pequenos módulos de software (não necessariamente componentes) que podem ser executados antes, durante ou após invocações em uma interface. Outros metamodelos complementares vêm sendo introduzidos no OpenCOMv2, como o metamodelo de recursos (Durán-Limón, 2001), que permite gerir a alocação de recursos (tempo de CPU, espaço de memória etc) a configurações.

2.1. Frameworks de componentes

O NetKit aplica o conceito de framework de componentes (Szyperski, 2002) na definição de arquiteturas de rede programável. Tradicionalmente, frameworks são vistos como esqueletos para a construção de software que permitem o reuso de módulos e de decisões de projeto. Porém, frameworks no NetKit não são um conceito de projeto somente. Eles são representados explicitamente em tempo de execução por 'componentes controladores' que restringem o modo como os componentes de uma configuração podem se comportar e interagir. Esse controle é baseado em regras específicas de domínio que não podem ser, em geral, descritas simplesmente pelo uso de receptáculos e metamodelos. Por exemplo, um framework de protocolos pode ser construído por meio de controladores que detêm conhecimento específico das maneiras pelas quais as entidades de protocolo de uma arquitetura em camadas podem interagir.

Como regra geral, funcionalidades de controle específicas de domínio são construídas sobre receptáculos e metamodelos. Considere novamente o exemplo do framework de protocolos mencionado acima. Imaginando que um projetista resolva substituir um componente em uma determinada camada da pilha de protocolos, os controladores do framework podem basear-se no grafo mantido pelo metamodelo de arquitetura para manipular o arranjo topológico da pilha – por exemplo, restringindo conexões diretas entre o novo componente e componentes em camadas não-adjacentes. Esses controladores podem também usar o metamodelo de introspecção para verificar se o novo componente oferece uma interface de passagem de pacotes requerida naquela camada. O metamodelo de interceptação pode ser aplicado durante o processo de

substituição do componente para reduzir ao mínimo a interrupção dos fluxos de dados processados pela pilha – por exemplo, retendo pacotes junto a um receptáculo momentaneamente desconectado. Por fim, o metamodelo de recursos pode ser acionado para alocar recursos de CPU (*threads*) e memória (*buffers*) ao novo componente.

3. O componente controlador genérico

Apesar de propiciarem um certo nível de reuso, frameworks de componentes agregam conhecimento de domínio específico e, por isso, demandam mecanismos de controle de reconfiguração também específicos, que em geral não podem ser reusados entre diferentes domínios. A manutenção de frameworks – por exemplo, atualizações nas regras de configuração – é um ponto ainda mais crítico no projeto desses artefatos.

Um grau maior de reuso pode ser conseguido partindo do princípio de que, apesar das restrições de reconfiguração serem específicas de domínio, a forma como elas são verificadas é normalmente a mesma. Isto é, em face de um evento de reconfiguração – por exemplo, a instanciação de um componente ou uma nova associação entre instâncias – o controlador responsável pela configuração em questão é acionado. A partir daí, regras específicas de domínio são aplicadas para validar a reconfiguração desejada.

O processo de síntese proposto neste artigo se apóia em um componente controlador de reconfiguração 'genérico', a partir do qual frameworks de componentes distintos podem ser construídos por meio da atribuição de regras de configuração diferentes a esse componente. O controlador genérico é capaz de detectar os vários eventos de reconfiguração possíveis no OpenCOMv2 – instanciação e remoção de componentes, associação e dissociação entre componentes instanciados, realocação de recursos – por meio das facilidades de inspeção e adaptação oferecidas pelos metamodelos. Com base na configuração corrente, nos eventos de reconfiguração e nas regras de configuração vigentes, o controlador pode admitir ou recusar reconfigurações.

O controlador genérico oferece duas interfaces principais (vide Figura 2). A interface ICFControl permite gerenciar a semântica do framework de componentes em tempo de execução através de operações de inserção, remoção e substituição de regras de configuração. A interface ICFValidation oferece uma única operação de verificação dessas regras. Quando invocada essa operação dispara uma série de operações de inspeção (por meio de receptáculos apropriados) nos metamodelos. Na implementação atual, o controlador genérico só inspeciona os metamodelos de introspecção e arquitetura. Contudo, esses metamodelos já conferem a esse componente uma boa abrangência de eventos de reconfiguração reconhecidos.

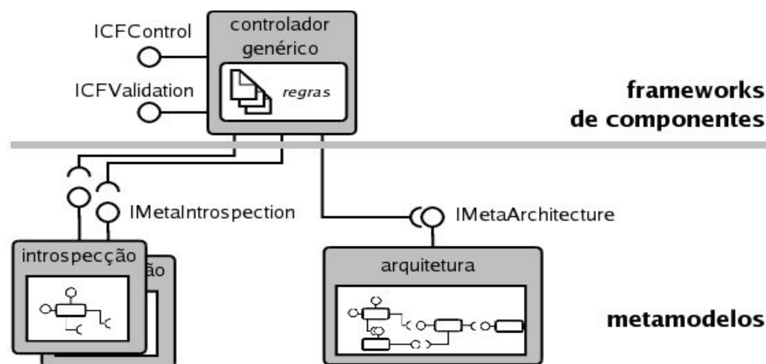

Figura 2. O componente controlador genérico.

3.1. Transações de reconfiguração

O controlador genérico pode usar os serviços providos por um ‘componente transacional’ para efetivar ou abortar reconfigurações. Esse componente permite manter a consistência de uma configuração em face de múltiplos eventos de reconfiguração que, se fossem tratados de modo isolado pelo controlador genérico, poderiam ser incorretamente invalidados – por exemplo, a instanciação de um componente seguida de sua associação com outras instâncias. Para isso, o componente transacional oferece um esquema de transação que permite tratar eventos relacionados como um único evento ‘atômico’. No início de uma transação de reconfiguração, esse componente gera uma cópia dos metadados dos metamodelos – novamente, na implementação atual desse componente somente os metadados de introspecção e arquitetura são tratados – e aplica as operações de reconfiguração sobre essa cópia. Ao final da transação, as restrições de reconfiguração são verificadas sobre a cópia alterada e, se elas forem satisfeitas, a cópia alterada substitui os metadados originais, efetivando a reconfiguração. Senão, a cópia alterada é descartada e a reconfiguração abortada.[2] É interessante notar que, com transações, as restrições de reconfiguração podem ser verificadas uma única vez para um conjunto de eventos de reconfiguração relacionados. Desse modo, em cenários envolvendo muitas restrições e múltiplos eventos, o tempo total de reconfiguração pode até mesmo ser reduzido, embora esse aspecto não seja avaliado neste artigo.

Para detectar eventos de reconfiguração o componente transacional atua como um ‘procurador’ dos metamodelos (vide Figura 3). O metamodelo de interceptação é acionado para que as invocações aos outros metamodelos sejam delegadas às interfaces correspondentes no componente transacional. Esse método é transparente para quem requisita as reconfigurações, o que facilita a sua adoção em sistemas pré-existentes. No entanto, o tratamento de múltiplas reconfigurações como um evento atômico depende do componente transacional ser informado sobre o início e fim das transações, por meio das operações oferecidas na interface ICFTransaction. Isso pode ser obtido por meio da

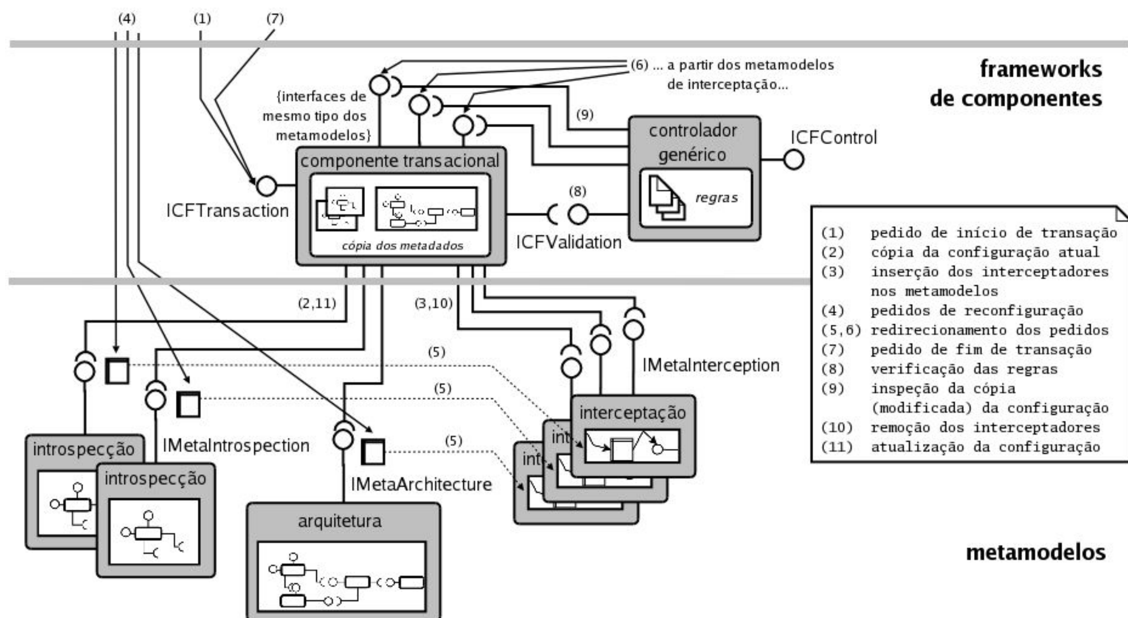

Figura 3. O serviço de reconfiguração transacional.

[2] O esquema descrito baseia-se na técnica de *deferred update*. Nada impede, porém, que outras técnicas de manutenção de atomicidade sejam usadas em futuras versões do componente transacional.

intervenção explícita de um componente externo (alternativa adotada na implementação atual) ou, assumindo-se que reconfigurações associadas ocorram em lote, por meio da detecção automática de início e fim de transação por parte do componente transacional.

3.2. *Scripts* de verificação

O controlador genérico hospeda um interpretador de *scripts* que permitem a descrição algorítmica das regras de configuração. A interface ICFVALIDATION do controlador genérico é acionada na ocorrência de eventos de reconfiguração, para que os *scripts* nele residentes sejam executados, com vistas a validar a reconfiguração solicitada.

Lua (Ierusalimschy et al., 2003) foi a linguagem de *script* usada neste trabalho para descrever as regras de configuração. Lua foi escolhida pela sua simplicidade e leveza. O interpretador Lua é menor que o de outras linguagens de *script* conhecidas (Perl, Tcl). Ainda assim, Lua possui facilidades de descrição e manipulação eficiente de dados, que são particularmente úteis em configurações em lote (vide Seção 3.3).

Scripts de verificação são codificados como funções Lua que devolvem valores booleanos. Quando o controlador genérico é acionado para verificar uma configuração, ele executa os *scripts* e admite a configuração somente se todos devolverem o valor "*true*". A Figura 4 ilustra um *script* que verifica a existência de uma instância do tipo uQoSNegGUID ao qual todas as instâncias do tipo uAdmCtrlGUID na configuração devem estar conectadas, por meio de interfaces do tipo pIntraLevelGUID.

Para validar reconfigurações, os *scripts* Lua usam uma API disponibilizada pelo controlador genérico que oferece as seguintes operações principais de inspeção:

- enumInstances: enumera instâncias de componentes em uma configuração;
- enumPorts: enumera interfaces e receptáculos de uma instância específica;
- getType: devolve o GUID de uma instância, interface ou receptáculo específico;
- enumBindings: enumera associações entre instâncias em uma configuração; e
- getBinding: devolve referência a uma associação específica.

```
function evalRuleX()
 local uQoSNegGUID     = "412ece3a-0f39-45cf-86d2-fbf27f365e13"
 local uAdmCtrlGUID    = "eb582791-4841-4bad-8722-b31b08e6145d"
 local pIntraLevelGUID = "2b383111-61b9-47de-ab09-f7fdecb60f65"

 local existsExp_1 = false
 for _, c in ipairs( netkit.enumInstances( uQoSNegGUID ) ) do
  local forAllExp_1 = true
  for _, n in ipairs( netkit.enumInstances( uAdmCtrlGUID ) ) do
   local existsExp_2 = false
   for _, pc in ipairs( netkit.enumPorts( c ) ) do
    local existsExp_3 = false
    for _, pn in ipairs( netkit.enumPorts( n ) ) do
     local boolExp =
      table.foreachi(
       netkit.enumBindings(),
       function( _,e )
        if utils.isEqual( e, netkit.getBinding( c, pc, n, pn ) ) then return true end
       end ) and
      utils.isEqual( netkit.getType( pn ), pIntraLevelGUID )
     if boolExp then existsExp_3 = true; break end
    end -- for
    if existsExp_3 then existsExp_2 = true; break end
   end -- for
   if not existsExp_2 then forAllExp_1 = false; break end
  end -- for
  if forAllExp_1 then existsExp_1 = true; break end
 end -- for
 return existsExp_1
end -- function
```

Figura 4. Exemplo de *script* de verificação.

A API acima usa os receptáculos do controlador genérico para inspecionar os metadados de introspecção e arquitetura. Os metadados inspecionados podem ser aqueles efetivamente geridos pelos metamodelos (Figura 2) ou cópias dos mesmos, nos casos em que o serviço provido pelo componente transacional é usado (Figura 3).

3.3. O componente configurador

A configuração de uma arquitetura complexa por meio de chamadas diretas às operações dos metamodelos pode exigir do projetista um esforço grande de programação. Além de essas operações deverem ser chamadas em uma ordem bem definida (uma associação entre instâncias de componentes só pode ser criada após a instanciação das mesmas!), o projetista deverá preocupar-se com questões ligadas a iniciação e término de transações, gerência de referências a instâncias e associações locais, entre outras. Este artigo propõe um 'componente configurador' que facilita a tarefa dos projetistas no que se refere a essas questões. Esse componente permite configurar uma arquitetura a partir de uma descrição declarativa da mesma. Em geral, o configurador atua em consonância com o controlador genérico e o componente transacional (vide Figura 5). No entanto, o configurador pode ser usado na ausência de ambos, caso em que o configurador ignoraria os serviços de transação e requisitaria reconfigurações diretamente aos metamodelos.

O configurador provê a interface ICFConfiguration, cujas operações permitem a declaração arquitetural de configurações. O formato de declaração entendido pelo configurador é baseado nas facilidades de descrição de dados de Lua. Novamente, o objetivo foi tornar o configurador o mais leve e eficiente possível (o *parser* Lua é muito menor e mais rápido do que *parsers* para outros formatos declarativos, como XML).

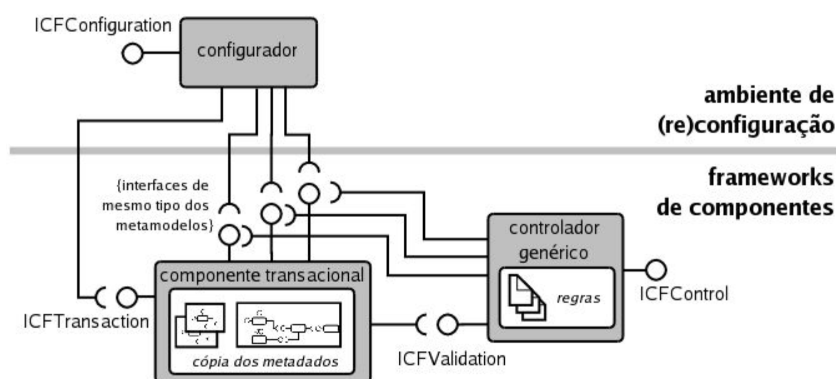

Figura 5. Componente configurador.

4. O ambiente de especificação LindaX

Tanto os *scripts* de verificação quanto as declarações arquiteturais em Lua podem ser sintetizados a partir de especificações de mais alto nível em LindaX. LindaX é apoiada extensivamente no conceito de 'estilos arquiteturais' (Monroe et al., 1997) como estratégia para permitir o reuso de especificações arquiteturais. Estilos arquiteturais são idiomas que caracterizam um conjunto de arquiteturas compartilhando determinadas propriedades estruturais e semânticas. Em LindaX, tais propriedades são descritas por meio de um 'vocabulário' e um conjunto de 'restrições'. O vocabulário define os tipos de componentes, conectores (em LindaX, denominados *pipes*) e interfaces existentes nas arquiteturas que seguem o estilo. As restrições definem as regras de instanciação e

ligação de componentes e *pipes* que devem ser obedecidas por essas arquiteturas. Em LindaX, isso permite que as características arquiteturais dos sistemas que seguem o estilo possam ser prescritas (em tempo de projeto) e impostas em face de reconfigurações (em tempo de execução).

A Figura 6 ilustra um exemplo de estilo LindaX. Para privilegiar a compreensão dos exemplos com LindaX, uma notação abreviada, sem marcações XML, é usada neste artigo, de modo que somente informações relevantes são apresentadas.[3] O estilo ilustrado define uma família de arquiteturas centralizadas de negociação de QoS. Nessas arquiteturas, componentes-satélite – os controladores de admissão (tipo UADMCTRL) – são responsáveis por receber pedidos de estabelecimento de contrato de serviço e encaminhá-los a um componente central – o negociador de QoS (tipo UQOSNEG) – por meio de *pipes* específicos para tal (tipo INTRALEVELPIPE). O negociador tem como função identificar os recursos envolvidos na provisão do serviço requisitado e dividir a responsabilidade dessa provisão entre eles. Esse estilo é uma simplificação de um dentre vários estilos definidos em LindaX para a especificação de arquiteturas hierárquicas de provisão de QoS. Detalhes desses estilos podem ser encontrados em Gomes et al (2003).

O exemplo ilustra algumas características importantes dos estilos LindaX. A primeira delas é a possibilidade de uso de diferentes notações para a representação das restrições. Na cláusula CONSTRAINTS o arquiteto de software informa as restrições do estilo bem como as notações utilizadas. Na versão atual da linguagem, essas restrições podem ser descritas usando somente lógica de primeira ordem (cláusula FOLPREDICATE). No ambiente de especificação LindaX é provida uma ferramenta que permite a síntese de *scripts* de verificação a partir dos predicados lógicos de primeira ordem e a

```
Style CentralizedNQoS {
 InterfaceType PIntraLevel( impl ) { Implementation = #impl; }
 InterfaceType PInterLevel( impl ) { Implementation = #impl; }

 ComponentType UQoSNeg( impl ) {
  Implementation = #impl;
  Cardinality    = 1;
  Port intraLevel { Cardinality = { 1 .. }; Type = #PIntraLevel; Direction = "in"; }
  Port interLevel { Cardinality = { 1 .. }; Type = #PInterLevel; Direction = "out"; }
 }

 ComponentType UAdmCtrl( impl ) {
  Implementation = #impl;
  Cardinality    = { 1 ..  };
  Port interLevel { Cardinality = { 1 .. }; Type = #PInterLevel; Direction = "in"; }
  Port intraLevel { Cardinality = 1;        Type = #PIntraLevel; Direction = "out"; }
 }

 PipeType IntraLevelPipe( impl ) {
  Implementation = #impl;
  Cardinality    = { 1 ..  };
  AccessPoint in  { Cardinality = 1; Type = #PIntraLevel; Direction = "in"; }
  AccessPoint out { Cardinality = 1; Type = #PIntraLevel; Direction = "out"; }
 }

 Constraints {
  FolPredicate AdmCtrlsWithQoSNeg {
   exists n: set Components( #UQoSNeg ) {
    forall c: set Components( #UAdmCtrl ) {
     exists pc: set Ports( #c ); pn: set Ports( #n ) {
      Pipe( #c,#pc,#n,#pn ) in Pipes() and PropertyValue( #s,"Type" ) == #PIntraLevel
     }
    }
   }
  }
 }
}
```

Figura 6. Exemplo de especificação de estilo em LindaX.

[3] Tradutores dessa notação para XML são oferecidas no ambiente de especificação LindaX.

instalação desses *scripts* no controlador genérico. A Figura 4 apresenta justamente o resultado da tradução da especificação do predicado lógico da Figura 6. Outra característica é a parametrização do vocabulário. Essa característica permite, por exemplo, o reuso de uma mesma especificação de estilo na geração de código para diferentes plataformas, embora este artigo foque somente no OpenCOMv2 como plataforma-alvo. No exemplo, a ferramenta de síntese de *scripts* associa o atributo parametrizado IMPLEMENTATION a GUIDs de componentes ou interfaces, que são descritos separadamente dos estilos em arquivos de parametrização que alimentam a ferramenta.

É importante destacar que estilos LindaX descrevem características arquiteturais de configurações, e não configurações em si. Além de estilos, LindaX oferece uma "metalinguagem" para descrições de configuração específicas de domínio. A semântica de uma descrição de configuração em LindaX é determinada por um *plug-in* específico de domínio, acoplado às ferramentas de síntese presentes no ambiente de especificação. Cada *plug-in* está associado a um conjunto de estilos LindaX, e cada descrição de configuração referencia um estilo, de modo que a ferramenta de síntese pode determinar o *plug-in* correto a ser usado a partir dessa referência. O tratamento de configurações em LindaX sob a perspectiva de domínios específicos (ao modo de DSLs – *Domain-Specific Languages*) permite uma maior concisão nas descrições do que ADLs (*Architecture Description Languages*), conforme demonstrado em Soares-Neto et al. (2003).

A Figura 7 ilustra um exemplo simplificado de descrição de configuração LindaX de um sistema de negociação de QoS, associada ao estilo da Figura 6. Como se pode notar, várias informações importantes acerca da configuração não estão presentes na sintaxe – por exemplo, os tipos de *pipe* e de interface usados na ligação entre os componentes OUTERCOMP1 e OUTERCOMP2 a CENTRALCOMP (cláusulas PIPE). Tais informações são 'inferidas' por um *plug-in* específico de estilo e passadas à ferramenta de síntese, que pode, por exemplo, traduzir a descrição de configuração LindaX em uma declaração arquitetural em Lua e repassar essa declaração ao componente configurador (Figura 5). Novamente, é importante destacar que, embora o foco deste artigo seja em configurações OpenCOMv2, o ambiente de especificação é suficientemente genérico para permitir o reuso de descrições de configuração – conquanto elas estejam devidamente parametrizadas – na síntese para outras plataformas. Detalhes do formato de descrição arquitetural de LindaX também são encontrados em Gomes et al. (2003).

```
System sys_example( ac_impl,neg_impl ) {
 Style = #CentralizedNQoS;

 Instance outerComp1  { Type = #UAdmCtrl( #ac_impl ); }
 Instance outerComp2  { Type = #UAdmCtrl( #ac_impl ); }
 Instance centralComp { Type = #UQoSNeg( #neg_impl ); }

 Pipe [#outerComp1, #centralComp];
 Pipe [#outerComp2, #centralComp];
}
```

Figura 7. Exemplo de descrição de configuração em LindaX.

5. Implantação em unidades de processamento de rede

Os ambientes de experimentação principais deste trabalho foram as NPUs IXP da Intel®. A Figura 8 apresenta a arquitetura geral de um roteador baseado na unidade IXP1200. O roteador consiste em um PC ligado à unidade IXP1200 por meio de um barramento PCI. A unidade IXP1200 contém um processador central StrongARM, um arranjo de processadores RISC – as 'micromáquinas' – e um conjunto de processadores dedicados. O StrongARM é responsável pelo tratamento de pacotes no plano de

controle e por outras funções de gerência. As micromáquinas são responsáveis pelo encaminhamento de pacotes no plano de dados. Os processadores dedicados (não mostrados na figura) executam tarefas específicas como verificação de redundância cíclica, cálculo de dispersão (*hash*) etc. Por fim, uma arquitetura hierárquica de memória compartilhada (também não mostrada) interliga os processadores da unidade.

Na implementação atual do OpenCOMv2 sobre o IXP1200, uma única cápsula engloba todo o roteador, incluindo o PC hospedeiro (vide Figura 8). Desse modo, o projetista de sistemas não precisa estar ciente das idiossincrasias da unidade, uma vez que elas são abstraídas pelo OpenCOMv2. Os componentes projetados para executar no StrongARM e nas micromáquinas são carregados nesses processadores e interligados por associações “locais” de modo transparente para o projetista. Detalhes do mapeamento do OpenCOMv2 em unidades IXP encontram-se em Coulson et al. (2003).

O controlador genérico e o componente transacional propostos neste trabalho são sempre instanciados no PC hospedeiro, junto aos metamodelos e serviços de cápsula do OpenCOMv2.[4] Pedidos de carga e associação local entre componentes (tanto no StrongARM quanto nas micromáquinas) passam necessariamente por esses módulos. A forma como os pedidos de reconfiguração são entregues a esses módulos depende do paradigma de rede programável usado no sistema. Em um ambiente de gerência de reconfiguração local (abordagem adotada na implementação atual) o controlador genérico e o componente transacional são acessíveis por meio de aplicações residentes no PC hospedeiro. Em um ambiente de rede ativa, pacotes de dados podem também carregar código de invocação de pedidos de reconfiguração. Em um ambiente de sinalização aberta, protocolos de sinalização específicos são necessários para esse fim.

Experimentos envolvendo a implementação de componentes OpenCOMv2 para as arquiteturas IntServ/ RSVP e DiffServ/ COPS-PR no roteador IXP1200 vêm sendo conduzidos no Laboratório TeleMídia da PUC-Rio, como parte do ambiente de testes dos mecanismos propostos neste trabalho. Ambas as arquiteturas foram modeladas em LindaX a partir dos estilos arquiteturais de provisão de QoS citados na Seção 4, que envolvem a gerência de reconfiguração de componentes atuando no plano de controle (sinalização), e de outros estilos arquiteturais para o plano de dados de roteadores programáveis, que gerenciam a reconfiguração de marcadores, classificadores,

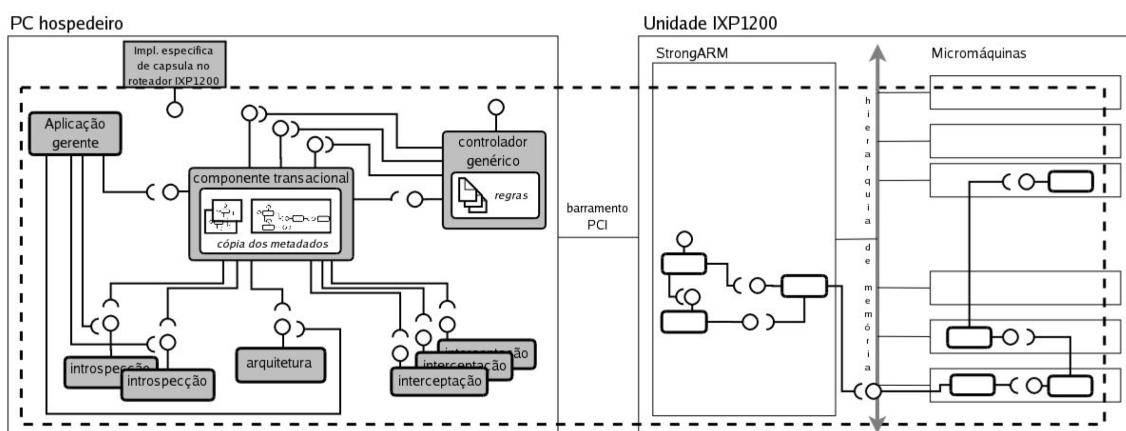

Figura 8. Implementação em um roteador IXP1200.

[4] A princípio, o componente configurador pode ser instanciado tanto local quanto remotamente – por exemplo, na mesma máquina onde o ambiente de especificação LindaX está sendo executado.

escalonadores etc. Os componentes do plano de dados, que povoam as micromáquinas, já são oferecidos pelo NetKit. Os componentes do plano de controle, que residem no StrongARM, foram obtidos a partir da recodificação de módulos de software de um trabalho anterior da PUC-Rio (Mota et al., 2001) com base no modelo OpenCOMv2.

6. Trabalhos relacionados

Diversas soluções baseadas em princípios de Engenharia de Software vêm sendo desenvolvidas na área de sistemas de comunicação reconfiguráveis. Essas soluções encontram-se distribuídas ao longo de um espectro que abrange desde especificações de alto nível de serviços até ambientes de configuração e programação especializados.

Dietrich e Hubaux (2002) apresentam um estudo detalhado da aplicação de formalismos na especificação de serviços e levantam a necessidade de integração gradual de linguagens formais no desenvolvimento de sistemas de comunicação. Eles propõem então um framework de monitoramento de propriedades, em tempo de execução, para sistemas implementados sobre CORBA. A especialização desse framework para um certo sistema se dá por meio da definição de um modelo formal para o sistema, baseado em eventos de interação entre seus objetos. Geram-se, então, interceptadores de interações que indicam a um objeto 'observador' os eventos ocorridos. O observador pode, assim, verificar propriedades do sistema a partir de predicados em uma lógica temporal definida sobre esses eventos. Em contraste com o presente trabalho, Dietrich e Hubaux privilegiam os aspectos comportamentais de um sistema. Já as abordagens baseadas em ADLs focam em aspectos estruturais – Medvidovic e Taylor (2000) fazem um apanhado desses trabalhos –, porém são poucas as ADLs que lidam com a síntese automática de sistemas reconfiguráveis. XelHa (Durán-Limón, 2001) é um das que mais se aproxima da abordagem proposta neste artigo, porém sem suporte a estilos arquiteturais ou a atualizações de regras de configuração em tempo de execução.

Na linha de ambientes de geração e programação de aplicações em redes programáveis, três trabalhos merecem destaque. Batory e Geraci (1997) propõem uma ferramenta que permite a síntese de código a partir de descrições em uma linguagem de domínio específico. A abordagem de LindaX é menos "granular", na medida em que uma biblioteca de componentes pré-existentes é usada e os únicos trechos de código que são realmente sintetizados são os *scripts* de verificação. Reid et al. (2000) propõem uma linguagem de configuração que permite a síntese de sistemas a partir de módulos pré-existentes codificados em C. A abordagem proposta no presente trabalho é mais genérica na medida em que componentes codificados em diferentes linguagens podem ser usados. Da Silva et al. (1998) propõem uma abordagem similar à apresentada neste artigo, definindo uma linguagem de configuração própria para redes ativas. Essa linguagem, no entanto, não lida com reconfigurações dinâmicas; reconfigurações são obtidas através da modificação e resubmissão de descrições de configuração. Na abordagem do presente trabalho, o suporte a restrições de reconfiguração provido pelos estilos LindaX possibilita o planejamento de reconfigurações bem controladas.

7. Conclusões e trabalhos futuros

Este artigo apresentou uma abordagem de especificação e síntese de configurações e de mecanismos de controle de reconfigurações para redes programáveis baseada no ambiente de programação NetKit e na linguagem LindaX. O trabalho apóia-se em um controlador de reconfiguração genérico, a partir do qual vários frameworks de

componentes para o NetKit podem ser construídos. A semântica específica de cada framework é definida por *scripts* que são gerados a partir de restrições de reconfiguração descritas em LindaX. A generalidade do controlador de reconfiguração aumenta o reuso entre frameworks distintos. Além disso, os *scripts* podem ser atualizados em tempo de execução, o que confere aos frameworks uma alta manutenibilidade.

Neste artigo foi proposto também um mecanismo de reconfiguração transacional oferecido junto ao controlador genérico. Esse mecanismo permite reduzir o tempo de reconfiguração bem como a manutenção da consistência de uma configuração em face de múltiplos eventos de reconfiguração relacionados.

Por fim, foi apresentado um mecanismo que instancia configurações a partir de descrições de configuração em LindaX. Esse processo reduz o esforço do projetista de sistema no que se refere à correta ordenação de operações de reconfiguração, gerência de referências a instâncias e associações locais etc. Além disso, uma vez que configurações podem ser validadas pelo controlador genérico, propriedades dessas configurações podem ser verificadas (com certo formalismo) em tempo de execução.

Embora o foco deste artigo tenha estado invariavelmente no NetKit, é importante destacar a abrangência da abordagem proposta. O ambiente de especificação LindaX admite a geração de código para diferentes plataformas e a integração com diferentes linguagens formais. Como prova de conceito, tradutores LindaX específicos para Java e para a ADL Wright (Allen, 1997) estão sendo desenvolvidos.

Três pontos principais têm sido estudados como trabalho futuro. Primeiro, os componentes de controle foram implementados em C++, o que limita a aplicação dos mesmos em cenários de reconfiguração genéricos. Estuda-se o uso de Java – que dá suporte nativo a reflexão – na implementação desses componentes, de modo que outros metamodelos específicos (já existentes ou a surgir) possam ser usados sem que seja necessário recompilar os componentes de controle. Em segundo lugar, pretende-se investigar o uso de heurísticas que detectam o início e término de reconfigurações em lote automaticamente. Isso permitiria o desenvolvimento de um serviço transacional de reconfiguração totalmente transparente, facilitando o projeto dos frameworks. Por fim, o processo de controle de reconfiguração apresentado neste artigo assume reconfigurações localizadas, isto é, adaptações distribuídas (intercápsulas) não são contempladas. É previsto no projeto o desenvolvimento de mecanismos de controle de reconfigurações distribuídas e sua aplicação na reconfiguração de protocolos de sinalização em redes programáveis (na implementação atual do ambiente de experimentação, reconfigurações ocorrem de modo independente em cada nó IntServ/ DiffServ).

Referências

Allen, R. (1997). *A formal approach to software architecture*. Pittsburgh, EUA. 248p. Tese de doutorado – School of Computer Science, Carnegie Mellon University.

Batory, D.; Geraci, B.J. (1997). Composition validation and subjectivity in GenVoca generators. *IEEE Transactions on Software Engineering*, v. 23, n. 2, p. 67-82.

Campbell, A.T. et al. (1999). A survey of programmable networks. *ACM SIGCOMM computer communications review*, v. 29, n. 2, p. 7-23.

Coulson, G. et al. (2003). NETKIT: a software component-based approach to programmable networking. *ACM SIGCOMM Computer Communications review*, v. 33, n. 5, p. 55-66.

Da Silva, S.; Florissi, D.; Yemini, Y. (1998). Composing active services in NetScript. In: DARPA Active Networks Workshop. Tucson, EUA. *Proceedings...*

Dietrich, F.; Hubaux, J-P. (2002). Formal methods for communication services: Meeting the industry expectations. *Computer Networks*, v. 38, n. 1, p. 109-126.

Durán-Limón, H.A. (2001). *A resource management framework for reflective multimedia middleware*. Lancaster, Inglaterra. 233p. Tese de doutorado – Computing Department, Lancaster University.

Gomes, A.T.A.; Coulson, G.; Blair, G.S.; Soares, L.F.G. (2003). A component-based approach to the creation and deployment of network services in the programmable Internet. *Monografia em Ciências da Computação MCC-42/03*, PUC-Rio. <http://www.telemidia.puc-rio.br/products/lindax/>. Acesso em: 8 out. 2004.

Ierusalimschy, R.; Figueiredo, L.H.; Celes, W. Lua 5.0 Reference Manual. (2003). *Relatório Monografia em Ciências da Computação MCC-14/03*, PUC-Rio. <http://www.lua.org/manual/5.0/>. Acesso em: 6 out. 2004.

Maes, P. (1987). Concepts and experiments in computational reflection. In: Annual Conference on O-O Programming Systems, Languages and Applications. Orlando, EUA. *Proceedings...* p. 147-155.

Medvidovic, N.; Taylor, R. (2000) A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*. v. 26, n. 1.

Mota, O.T.J. et al. (2001). Uma arquitetura adaptável para provisão de QoS na Internet. In: XIX Simpósio Brasileiro de Redes de Computadores. Florianópolis. *Anais...* p. 622-637.

Monroe, R. T. et al. (1997). Architectural styles, design patterns, and objects. *IEEE Software*, v. 14, n. 1, p. 43-42.

Network Processing Forum. (2001). *History & milestones*. <http://www.npforum.org/about/>. Acesso em: 27 set. 2004.

Reid, A. et al. (2000). Knit: Component composition for systems software. In: IV Symposium on Operating Systems Design and Implementation. San Diego, EUA. *Proceedings...* p. 347-360.

Soares-Neto, C. S.; Moreno, M. F.; Gomes, A. T. A.; Soares, L. F. G. (2003). Descrição arquitetural da provisão de QoS para suporte a aplicações multimídia. In: IX SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB. *Anais...* Salvador, Brasil.

Szyperski, C. (2002). *Component Software:* beyond object-oriented programming. 2. ed. New York, EUA: Addison-Wesley. 589p.