Uma Proposta para Reconfiguração Consistente no Nível Arquitetural

Jonivan Lisbôa[1], Orlando Loques[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Rua Passo da Pátria 153, Bloco E – 24210-240 – Niterói – RJ – Brasil
{jlisboa,loques}@ic.uff.br

Resumo. *Em aplicações adaptativas, a manutenção da consistência após uma adaptação está associada à manutenção de seus requisitos funcionais e não-funcionais, respeitando-se as dependências entre seus componentes. A consistência possui três aspectos: estado dos componentes, comunicação e configuração da aplicação. Este trabalho explora o último deles, propondo um modelo para manutenção da consistência da configuração no nível arquitetural, através do tratamento de falhas nas atividades básicas de configuração de componentes. O objetivo principal é evitar incoerências entre o modelo de alto nível utilizado e a configuração real, e assim não comprometer o uso de arquiteturas de software no projeto de aplicações.*

Abstract. *In adaptive applications, the maintenance of consistency after an adaptation is related to maintenance of both functional and non-functional requirements, respecting dependencies among their components. Consistency has three aspects: component state, communication, and application configuration. This work explores the last one, proposing a model for configuration's consistency maintenance at architectural level, by treating failures in basic component configuration activities. The main goal is to avoid incoherences between high level model and actual configuration, and so do not compromise the utilization of software architectures in application design.*

1. Introdução

Diversas aplicações computacionais modernas são projetadas para adaptar automaticamente sua configuração em tempo de execução, visando adquirir características como alta disponibilidade e tolerância a falhas. O desenvolvimento de aplicações adaptativas cresceu com o lançamento da idéia de Computação Autônoma (*Autonomic Computing*) [Kephart e Chess 2003], uma iniciativa que visa criar sistemas computacionais com a capacidade de auto-gerenciamento, sem a intervenção direta do ser humano. Este, como projetista, passa a ter o papel de definir políticas e regras gerais que servem como entrada para o processo de auto-gerenciamento, que envolve: auto-configuração (configuração automática de componentes), auto-correção (detecção automática de falhas), auto-otimização (controle de alocação de recursos para funcionamento ótimo) e auto-proteção (proteção contra ataques).

Adaptações na configuração ocorrem através da execução seqüencial de operações sobre o ciclo de vida de componentes – ativação, desativação e troca de ligações. Desde trabalhos primordiais sobre reconfiguração dinâmica [Kramer e Magee

1985], a manutenção da consistência da aplicação após uma adaptação é um problema relevante. A consistência está relacionada ao atendimento de todos os requisitos funcionais e não-funcionais definidos para a aplicação, respeitando-se as dependências existentes entre os componentes envolvidos na adaptação [Kon e Campbell 2000]. Em linhas gerais, o problema pode ser analisado sob três aspectos:

- A consistência de estado relaciona-se à manutenção do estado de computação apresentado pela aplicação no momento da adaptação;
- A consistência de comunicação relaciona-se com o tratamento de possíveis mensagens em trânsito entre componentes envolvidos na adaptação, sem prejuízo para a funcionalidade da aplicação;
- A consistência de configuração está ligada à manutenção de propriedades invariantes da aplicação como um todo. Após uma adaptação, o conjunto de componentes configurados deve continuar a atender adequadamente os requisitos definidos para a aplicação. Deve-se também manter a coerência entre o modelo de alto nível e a configuração real de componentes.

Em diversas abordagens citadas mais adiante, a consistência da configuração é um aspecto pouco explorado, por não existir na maioria delas a preocupação com a visão arquitetural da aplicação. Julgamos que isso é uma ferramenta importante para a obtenção de benefícios como análise comportamental em alto nível e separação entre interesses funcionais e não-funcionais. Esse fato serve como motivação para a proposta deste trabalho: apresentar um modelo para implementação de reconfiguração consistente no nível arquitetural. O modelo proposto apóia-se na detecção e tratamento de falhas nas atividades relacionadas à configuração de componentes, que possuem potencial para gerar inconsistências no modelo de alto nível. Espera-se assim manter a equivalência entre o modelo arquitetural utilizado e a configuração real, não se comprometendo o uso de arquiteturas de software e seus benefícios na fase de projeto da aplicação.

A seqüência deste artigo contém: a Seção 2, que apresenta paradigmas considerados para a elaboração da proposta; a Seção 3, que apresenta trabalhos correlatos sobre adaptação dinâmica; a Seção 4, que apresenta o modelo proposto; a Seção 5, que apresenta um exemplo para validação; e a Seção 6, que apresenta uma análise da proposta, conclusões e direções futuras.

2. Paradigmas considerados

O desenvolvimento de aplicações adaptativas vem convergindo para a utilização de determinadas técnicas e paradigmas, que foram considerados para o modelo proposto neste trabalho:

- Modelos de componentes que servem como uma abstração para descrever recursos de hardware e software e permitem o desenvolvimento de aplicações através da composição de blocos independentes. O desenvolvimento baseado em componentes é derivado da pesquisa e utilização na indústria da tecnologia de orientação a objetos, e ganhou força com a disseminação de plataformas como CORBA Component Model [OMG 2008], Enterprise Java Beans [Sun 2008], COM+/.Net [Microsoft 2008], dentre outras. Tais plataformas são responsáveis

por fornecer serviços básicos de gerenciamento de componentes, como controle de ciclo de vida, persistência, nomes e diretório, transações, introspecção, adaptação, e outros;

- Conceitos de Arquiteturas de Software [Shaw e Garlan 1996] para modelar em alto nível o comportamento de uma aplicação, em função de seus componentes e as interações existentes entre eles. Isso permite análises comportamentais na fase de projeto e separação clara entre interesses funcionais e não-funcionais, com a utilização de contratos entre componentes [Meyer 1992, Beugnard et al. 1999] para descrever configurações possíveis para a aplicação levando-se em conta requisitos de qualidade de serviço, e contratos de adaptação para definir regras de transição entre configurações;
- Configuração de recursos e adaptação da configuração em tempo de execução, através de *frameworks* programáveis que reúnem modelos arquiteturais e mecanismos de gerenciamento de componentes. A adaptação dinâmica apóia-se na capacidade dos *frameworks* de avaliar as condições do ambiente de execução de uma aplicação e confrontá-las com os requisitos funcionais e, principalmente, não-funcionais definidos para ela. Os *frameworks* de adaptação atuam sobre a plataforma de gerenciamento de componentes, de modo a controlar de forma dinâmica o seu ciclo de vida (ativação, desativação) e as ligações entre eles, permitindo a conexão e desconexão em tempo de execução.

3. Trabalhos relacionados

Diversos trabalhos apresentam *frameworks* para suporte ao gerenciamento automático de configuração. Dentre eles, podemos destacar: QuO [Loyall et al. 1998], Rainbow [Garlan et al. 2004], Q-CAD [Capra et al. 2005], Plastik [Batista et al. 2005], CR-RIO [Cardoso et al. 2006]. O funcionamento dos *frameworks* baseia-se no esquema "sentir-planejar-agir" dos sistemas de controle para automatizar a adaptação em aplicações baseadas em componentes, e apresentam como elementos principais:

- Um modelo para descrição em alto nível de tipos de componentes, configurações possíveis para a aplicação, condições para ativação de cada configuração e regras de transição entre configurações;
- Um sistema de informação de contexto, com monitoração de propriedades do ambiente através de sensores ligados aos componentes, e armazenamento em bases de dados apropriadas;
- Um gerenciador de configuração que toma decisões de acordo com o contexto observado e as condições previstas nos contratos, fazendo a aplicação reagir a variações observadas no ambiente de execução, através da intervenção no ciclo de vida de componentes e nas ligações entre eles.

Inicialmente, visa-se aplicar o modelo proposto neste trabalho no *framework* CR-RIO, que foi desenvolvido e vem sendo continuamente aprimorado pelo grupo de pesquisa ao qual os autores pertencem. Como diferencial em relação aos outros, o CR-RIO possibilita a obtenção de uma visão arquitetural da aplicação, com a descrição de arquiteturas e contratos. Mediante uma linguagem declarativa (CBabel), são descritos

tipos de componentes e instâncias, ligações permitidas entre eles e possibilidades de configuração considerando-se requisitos de qualidade.

A Figura 1 apresenta o esquema do *framework* CR-RIO, e seus elementos: sensores de monitoração de propriedades (*Resource Agents*); sistema de informação de contexto; registro de componentes validados, que serve como base de dados para obtenção dinâmica de referências a componentes; configurador, que atua sobre o ciclo de vida dos componentes executando os comandos de configuração; e o próprio elemento gerenciador. Alguns detalhes sobre sintaxe e semântica dos comandos da linguagem podem ser encontrados no exemplo descrito na Seção 5.

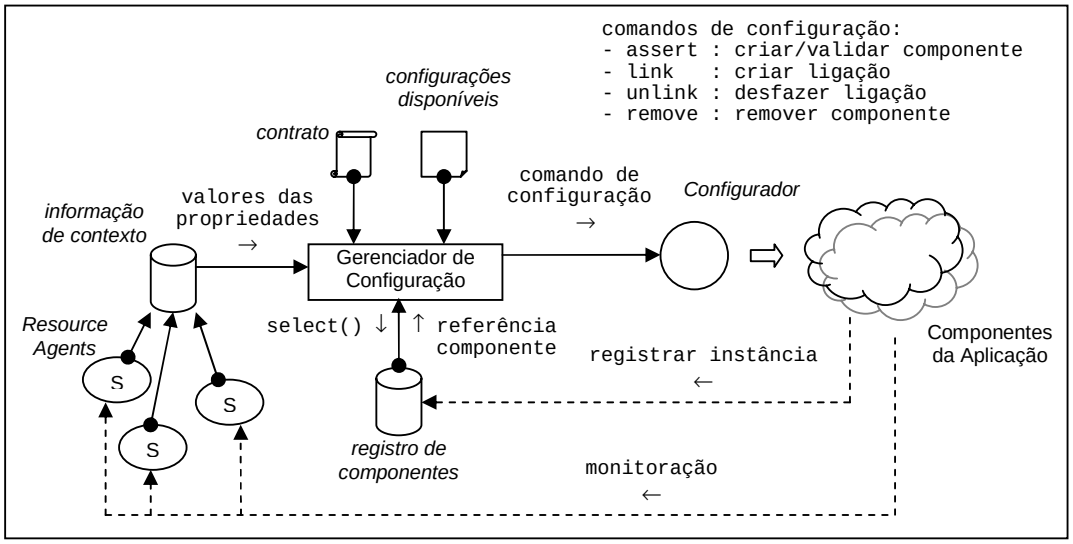

Figura 1. Esquema do *framework* CR-RIO

Um fato relevante observado nos trabalhos destacados é que as alterações na aplicação são impostas sem que seja verificada a ocorrência de falhas na configuração dos componentes. Isso possui potencial para causar inconsistências entre a representação de alto nível da aplicação (arquitetura) e a sua configuração real, pois falhas na ativação de um componente ou na substituição de alguma ligação farão com que a configuração real apresente um estado errôneo e não-sinalizado no nível mais alto. Para resolver esse problema, a idéia principal do modelo proposto é obter um retorno sobre o sucesso de cada atividade básica sobre o ciclo de vida dos componentes envolvidos em uma adaptação, para manter a coerência entre a representação de alto nível da aplicação e sua configuração real.

4. Proposta para Reconfiguração Consistente

O modelo proposto neste trabalho realiza o tratamento de possíveis fontes de inconsistência no processo de adaptação dinâmica, através de detecção de falhas ocorridas no nível mais baixo da execução da adaptação. As falhas detectadas são tratadas no nível arquitetural pelo mecanismo de gerenciamento de configuração, segundo algoritmo detalhado mais adiante. Para este trabalho, assume-se que existe um *framework* programável capaz de monitorar as condições do ambiente e tomar decisões

sobre adaptações dinâmicas na configuração da aplicação – no caso, o CR-RIO e um contrato para gerenciamento da configuração.

4.1. Idéia básica da proposta

Uma forma de se garantir uma reconfiguração consistente é fazer com que a atividade de adaptação possua um caráter transacional atômico: a seqüência de atividades básicas para se alterar a configuração da aplicação deve ser executada completamente para que a alteração seja bem sucedida. Segundo [Romanovsky 2001], o modelo tradicional de transações [Gray e Renter 1993, Lynch et al. 1993] possui algumas falhas que motivaram sua extensão, ou aprimoramento. Uma delas é a não existência de mecanismos aceitáveis de recuperação frente a situações contornáveis como, por exemplo, as exceções de software que podem acontecer na execução de uma adaptação. Uma forma de realizar tal recuperação é relaxar a propriedade de atomicidade de uma transação, de modo que seja possível tratar falhas que a invalidariam.

A proposta deste trabalho oferece um mecanismo para relaxamento da atomicidade na atividade de adaptação. As operações básicas de configuração de componentes são organizadas em um nível mais interno de execução, e são executadas de forma gradual. Caso não ocorra falha na execução de uma operação, ela é adicionada a um histórico de operações bem sucedidas. Uma falha na execução é sinalizada através do disparo de uma exceção gerada pelo configurador de componentes e capturada para tratamento pelo gerenciador de configuração.

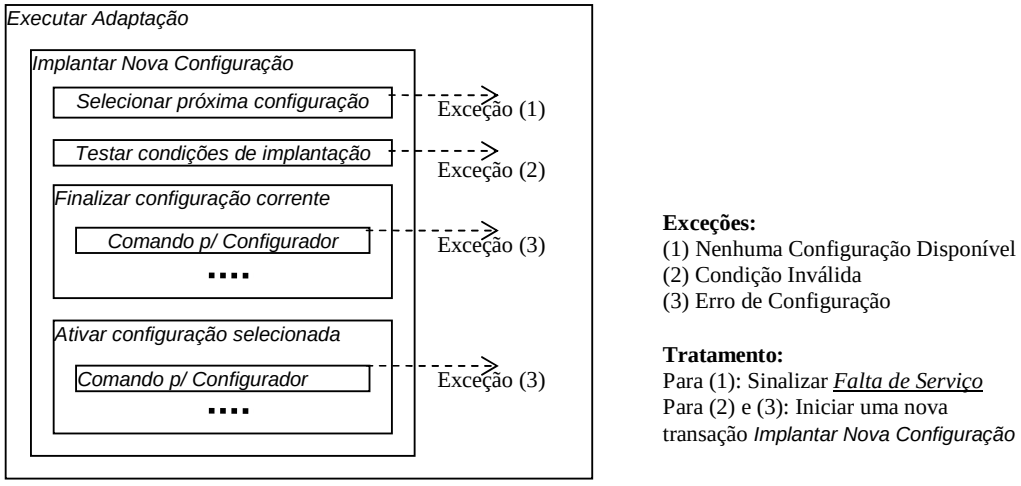

Figura 2. Organização das transações no modelo de adaptação proposto

A Figura 2 apresenta a organização em níveis das transações, representadas por retângulos, as possíveis exceções e o respectivo tratamento. A transação mais externa (*Executar Adaptação*) é iniciada nas situações em que uma adaptação é requerida: início da aplicação (primeira configuração); ocorrência de falha em algum componente da configuração corrente; ou, a configuração corrente deixa de atender aos requisitos de qualidade de serviço definidos em contrato, devido a variações observadas em propriedades monitoradas nos componentes configurados. A transação engloba uma outra (*Implantar Nova Configuração*), estruturada para executar uma seqüência de passos – também transações – para implantar uma configuração: (i) selecionar uma nova

configuração dentre as disponíveis; (ii) testar condições de implantação da nova configuração; (iii) finalizar a configuração corrente; (iv) ativar a nova configuração.

4.2. Execução da adaptação

A Figura 3 mostra um pseudocódigo para a transação do 2º. Nível (*Implantar Nova Configuração*), que contém a sequência de execução das transações mais internas (3º. Nível) que a compõem. A transação de 1º. Nível (*Executar Adaptação*) será ativada pela *thread* principal de execução do gerenciador de configuração, que monitora periodicamente a validade do contrato de adaptação. O pseudocódigo será útil para o entendimento dos exemplos apresentados na Seção 5. No código, **C** representa a configuração corrente, e **C´** representa uma nova configuração.

```
Transação do 2º. Nível
  ImplantarNovaConfiguração()
  Início
    C´ = SelecionarConfiguração()
    TestarPerfil(C´)
    FinalizarConfiguração(C,C´)
    AtivarConfiguração(C´)
  Fim

Transações do 3º. Nível:
  TestarPerfil(Conf) : testa condições iniciais da configuração indicada

  SelecionarConfiguração() : retorna próxima configuração disponível no contrato

  AtivarConfiguração(Conf)
  Início
    Enquanto houver comandos na descrição da configuração indicada
      Obter próximo comando
      Configurador executa comando
      Colocar comando no Histórico de Operações Bem Sucedidas
    Fim-Enquanto
  Fim

  FinalizarConfiguração(Conf1,Conf2)
  Início
    Obter operações úteis a Conf2
    Enquanto houver comandos no Histórico de Operações Bem Sucedidas
              - Obter próximo comando
              - Se comando não for útil a Conf2,
                    então Configurador desfaz comando
    Fim-Enquanto
  Fim
```

Figura 3. Pseudocódigo das transações de 2º. e 3º. níveis do modelo proposto

Os dados do contrato de adaptação vigente servem de entrada para o gerenciamento da configuração. Dele são extraídos a lista de configurações disponíveis, com as instruções e condições para a implantação de cada uma delas. Tais dados são úteis na execução das transações *Selecionar Próxima Configuração* e *Testar Condições de Implantação*. As transações *Finalizar Configuração Corrente* e *Ativar Configuração Selecionada* envolvem a execução das instruções obtidas na descrição da configuração selecionada. Na finalização da configuração corrente, as operações básicas realizadas são desfeitas, mediante consulta ao histórico de operações realizadas com sucesso. Assume-se que a semântica da linguagem de descrição permita associar as instruções de configuração aos pares, com uma possuindo efeito contrário à outra, e que o configurador seja capaz de identificar isso. Por exemplo, no CR-RIO podem ser definidos os pares {`assert`; `remove`} e {`link`; `unlink`}. A utilização do histórico pode ser otimizada para que se desfaçam somente operações não necessárias à nova

configuração a ser implantada. Assim, são preservados componentes e ligações que serão úteis à nova configuração. Para auxiliar isso, define-se uma operação ***util*(C)**, que retorna a lista de operações necessárias à uma configuração C que já foram realizadas sem falha e estão presentes no histórico de operações concluídas.

5. Validação da proposta

Para validar a proposta deste trabalho, será descrito um exemplo aplicável ao *framework* CR-RIO. O exemplo consiste em uma aplicação cliente que utiliza um serviço oferecido por um provedor de informação (no exemplo, cotação da Bolsa de Valores), configurando-se uma arquitetura cliente-servidor típica. Para o exemplo, serão considerados dois provedores diferentes, e cada um deles pode oferecer dois tipos de serviço: com criptografia e sem criptografia. Cliente e provedor podem interagir através de conexão de rede fixa ou sem fio. Por questões de segurança, é desejo dos usuários da aplicação que a informação proveniente do serviço utilizado seja criptografada caso seja utilizada a conexão sem fio. Nessa situação, deve ser utilizado um componente para decodificar a informação antes que ela seja utilizada pelo cliente.

A Figura 4 ilustra o cenário obtido. Na Figura 4a, é mostrada a arquitetura genérica, com a aplicação cliente (AP) ligada ao provedor de serviço (PS) através de alguma conexão de rede (C). A Figura 4b ilustra as possíveis configurações para a aplicação, considerando-se as instâncias reais dos componentes disponíveis para configuração: os dois provedores (PS1 e PS2) e os tipos de serviço oferecidos (N para normal, K para criptografada), as conexões de rede (F para fixa, W para sem fio), e o componente decodificador (DK) no caso de utilização do serviço com criptografia.

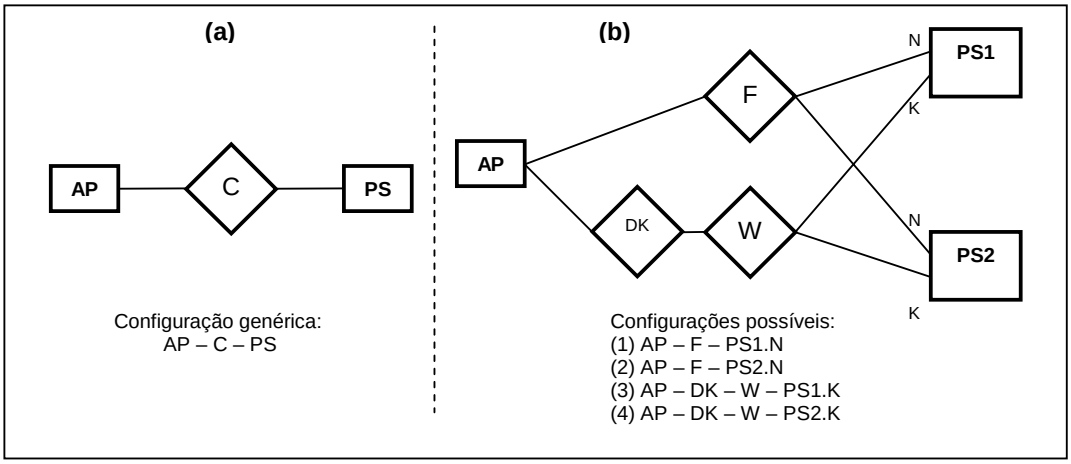

Figura 4. Exemplo de aplicação cliente-servidor: (a) Arquitetura genérica; (b) Configurações possíveis com componentes reais

5.1. Descrição no *framework* CR-RIO

O *framework* CR-RIO utiliza a linguagem CBabel para descrever arquiteturas e contratos. Para o exemplo proposto, são apresentados a seguir na Figura 5: as descrições da arquitetura básica da aplicação e os elementos do contrato para gerenciamento da configuração – configurações reais disponíveis e possibilidades de transição entre elas.

```
1.    module ClientServer {
2.      module Client { out port getQuote }
3.      module Provider { in port getQuoteN
4.                        in port getQuoteK }
5.      connector NetworkConnection: external;
6.      connector Decrypt: external;
7.      assert Client as AP;
8.      assert Provider as PS;
9.      assert NetworkConnection as C;
10.     link AP to PS by C
11.   }
12.   category ProviderNFP {
13.     responseTime: dynamic decreasing numeric ms; }
14.   category NetworkNFP {
15.     technology: static enum (fixed, wireless); }
16.   contract {
17.     service { assert Provider as PS1 at provider1.com monitoring ProviderNFP;
18.               assert Provider as PS2 at provider2.com monitoring ProviderNFP;
19.     } initProvider;
20.     negotiation { initProvider -> no-service }
21.   } startUpcontract;
22.   contract {
23.     _provider: Provider;
24.     profile { ProviderNFP.responseTime < 50; } providerSelection;
25.     profile { NetworkNFP.technology = fixed; } fNet;
26.     profile { NetworkNFP.technology = wireless; } wNet;
27.     service { assert Client as AP at <clientHost>;
28.               assert NetworkConnection as F with fNet;
29.               _provider = select Provider with providerSelection;
30.               link AP.getQuote to _provider.getQuoteN by F;
31.     } fixedNetwork;
32.     service { assert Client as AP at <clientHost>;
33.               assert NetworkConnection as W with wNet;
34.               assert Decrypt as DK at <clientHost>;
35.               _provider = select Provider with providerSelection;
36.               link AP.getQuote to _provider.getQuoteK by DK, W;
37.     } wirelessNetwork;
38.     negotiation { not fixedNetwork -> (wirelessNetwork -> no-service)
39.                   wirelessNetwork -> fixedNetwork
40.                   not wirelessNetwork -> no-service
41.     }
42.   } adaptationContract;
```

Figura 5. Descrições da arquitetura básica e contratos para o exemplo citado

Explicações sobre as descrições apresentadas:

- Descrição da Arquitetura Básica (linhas de 1 a 11): define os tipos de componente (módulos) utilizados: aplicacão cliente (`Client`, linha 2) e provedor (`Provider`, linhas 3-4). A porta `getQuote` indica a operação de solicitação da cotação da bolsa realizada pelo cliente, e as portas `getQuoteN` e `getQuoteK` indicam as operações disponibilizadas no provedor: sem criptografia e com criptografia. Os tipos de conectores `NetworkConnection` (conexão de rede, linha 5) e `Decrypt` (decriptação, linha 6) são declarados com externos, ou seja, providos pela plataforma de suporte à execução da aplicação. Nas linhas de 7 a 9 são descritas validações (`assert`) de instâncias genéricas dos componentes da aplicação (`AP`, `PS` e `C`), e na linha 10 é definida a restrição básica para composição da aplicação: o cliente liga-se ao provedor através de alguma conexão de rede disponível;

- Categorias (linhas de 12 a 15): indicam a lista de propriedades não-funcionais de interesse em um determinado recurso ou componente. A categoria `ProviderNFP` apresenta a propriedade tempo de resposta (`responseTime`) para um provedor de serviço. A declaração `dynamic` indica que a propriedade

possui valor dinâmico, e deverá ser monitorada em tempo de execução, e `decreasing` indica que menores valores são mais desejáveis na monitoração. A categoria `NetworkNFP` apresenta a propriedade tecnologia (`technology`) utilizada para uma conexão de rede, como sendo uma enumeração com dois valores possíveis: `fixed` e `wireless`. A propriedade é estática (`static`), o que indica que seu valor será atribuído e não sofrerá alteração.

- Contrato para validação dos provedores (linhas de 16 a 21): serve para adicionar referências dos provedores ao registro de componentes disponíveis, já que a inicialização deles é independente. O serviço `initProvider` apresenta as declarações para validação das duas instâncias, `PS1` e `PS2`, nos seus respectivos endereços, indicando a monitoração das propriedades definidas na categoria `ProviderNFP` – nesse caso, o tempo de resposta. No contrato, a cláusula `negotiation` representa uma máquina de estados. A configuração do serviço `initProvider` será tentada, e caso não seja bem sucedida, acontecerá uma transição para o estado de serviço indisponível (`no-service`), sinalizando que não foi possível a validação. Este contrato é independente do contrato de gerenciamento de configuração, e não será tratado no exemplo.
- Contrato de gerenciamento da configuração (linhas de 22 a 42): Nele, são definidos dois serviços, que descrevem as possibilidades de configuração: `fixedNetwork` (linhas 27-31) utiliza conexão de rede fixa, e `wirelessNetwork` (linhas 32-37) utiliza conexão sem fio com criptografia. Nos serviços, são indicadas as seguintes atividades:
    - Validação do cliente (linhas 27 e 32) e do conector de rede (linhas 28 e 33), de acordo com os respectivos perfis: tecnologia fixa (`fNet`, linha 25) e tecnologia sem fio (`wNet`, linha 26)
    - A seleção dinâmica (`select`) de uma instância de provedor (linhas 29 e 35) segundo o perfil `providerSelection` (linha 24), que define que o tempo de resposta monitorado deve ser inferior a 50ms. A instância selecionada é armazenada na variável `_provider`;
    - A ligação (`link`) entre o cliente e o provedor, utilizando os conectores adequados (linhas 30 e 36);
    - Diferenças entre os dois serviços: (i) utilização do conector `F` para rede fixa e `W` para sem fio; (ii) ligações entre as portas de cliente e provedor: sem criptografia (`getQuoteN`) no serviço de rede fixa, e com criptografia (`getQuoteK`) no serviço de rede sem fio; (iii) utilização do conector de decriptação (`DK`) no serviço de rede sem fio;
- A cláusula de negociação (linhas de 38 a 40) mostra que a configuração `fixedNetwork` é preferencial. Transições para `wirelessNetwork` e `no-service` não são automáticas, ou seja, somente acontecerão se a primeira não estiver disponível. Este fato, indicado pela declaração `not`, ocorre do mesmo modo na transição de `wirelessNetwork` para `no-service`. A transição de `wirelessNetwork` para `fixedNetwork` é automática – acontecerá caso a primeira esteja ativa e a segunda torne-se disponível.

5.2. Aplicação do contrato de adaptação

São apresentadas quatro situações para ilustrar a aplicação do contrato de gerenciamento de configuração descrito para o exemplo. O objetivo é demonstrar o tratamento de falhas na adaptação, e também a utilização otimizada do histórico de operações concluídas. Será assumido que as referências dos provedores de serviço já estão devidamente adicionadas ao registro de componentes. Nas descrições das situações, **C** representa a configuração corrente, e **C'** representa uma nova configuração selecionada.

As situações acontecem na seguinte ordem:

1º) Imposição da configuração `fixedNetwork` como primeira configuração;
2º) Mudança no provedor, mantendo-se a conexão de rede fixa;
3º) Mudança no provedor, porém com falha na conexão de rede fixa;
4º) Mudança de conexão de rede, de sem fio para fixa.

Para cada situação, são mostrados: (i) a informação de contexto com as propriedades monitoradas; (ii) a execução do algoritmo mostrado na Seção 4.2; (iii) o estado do histórico de operações concluídas, atualizado a cada operação executada; (iv) a ocorrência de falhas de configuração e respectivo tratamento. Para simplificar, a operação `link` foi resumida, omitindo-se as portas envolvidas na ligação.

Situação 1: Imposição da primeira configuração

Contexto:

Configuração corrente: C = *null* (nenhuma configuração ativa)

Propriedades: `PS1.ProviderNFP.responseTime = 30` ←
`PS2.ProviderNFP.responseTime = 35`

Execução da transação *Implantar Nova Configuração*:

1. *Selecionar próxima configuração* (selecionada: C' = `fixedNetwork`)
2. *Testar condições iniciais* (perfil `fNet`: OK)
3. *Finalizar configuração corrente*: *nada a fazer*
4. *Ativar configuração selecionada*:

`assert Client as AP at <clientHost>` → OK (cliente validado)
histórico: `OP1 - assert Client as AP at <clientHost>`

`assert NetworkConnection as F with fNet` → OK (conexão validada)
histórico:
```
OP1 - assert Client as AP at <clientHost>
OP2 - assert NetworkConnection as F with fNet
```

```
_provider = select Provider with providerSelection
```
instância selecionada: `PS1` (menor tempo de resposta)

`link AP to _provider by F` : OK (ligação criada: `AP–F–PS1`)
histórico:
```
OP1 - assert Client as AP at <clientHost>
OP2 - assert NetworkConnection as F with fNet
OP3 - link AP.getQuote to PS1.getQuoteN by F
```

Conclusão: Configuração `fixedNetwork` ativada com sucesso(`AP–F–PS1`)

Situação 2: Mudança de provedor (seleção dinâmica)

Contexto:

Configuração corrente: C = `fixedNetwork (AP–F–PS1)`

Propriedades: `PS1.ProviderNFP.responseTime = 30`
`PS2.ProviderNFP.responseTime = 25` ←

Histórico:
```
OP1 - assert Client as AP at <clientHost>
OP2 - assert NetworkConnection as F with fNet
OP3 - link AP.getQuote to PS1.getQuoteN by F
```

Execução da transação *Implantar Nova Configuração*:

1. *Selecionar próxima configuração* (selecionada: C' = `fixedNetwork`)
2. *Testar condições iniciais* (perfil `fNet`: OK)
3. *Finalizar configuração corrente*

   Verificar operações úteis no histórico: *util* (C') = { `OP1`, `OP2` }

   Finalização: Desfazer `OP3` :`unlink AP from PS1`

   Não desfazer `OP2`

   Não desfazer `OP1`
4. *Ativar configuração selecionada*:

   `assert Client as AP at <clientHost>` → OK (aproveitada)

   histórico: `OP1 - assert Client as AP at <clientHost>`

   `assert NetworkConnection as F with fNet` → OK (aproveitada)

   histórico:
   ```
   OP1 - assert Client as AP at <clientHost>
   OP2 - assert NetworkConnection as F with fNet
   ```
   ```
   _provider = select Provider with providerSelection
   ```

   instância selecionada: `PS2` (menor tempo de resposta)

   `link AP to _provider by F` : OK (ligação criada: `AP–F–PS2`)

   histórico:
   ```
   OP1 - assert Client as AP at <clientHost>
   OP2 - assert NetworkConnection as F with fNet
   OP3 - link AP.getQuote to PS2.getQuoteN by F
   ```

Conclusão: Configuração `fixedNetwork` ativada com sucesso (`AP–F–PS2`)

Situação 3: Mudança de provedor com falha na conexão após teste inicial

Contexto:

Configuração corrente: C = `fixedNetwork (AP–F–PS2)`

Propriedades:
```
PS1.ProviderNFP.responseTime = 20 ←
PS2.ProviderNFP.responseTime = 25
```

Histórico:
```
OP1 - assert Client as AP at <clientHost>
OP2 - assert NetworkConnection as F with fNet
OP3 - link AP.getQuotes to PS2.getQuoteN by F
```

Execução da transação *Implantar Nova Configuração*:

1. *Selecionar próxima configuração* (selecionada: C' = `fixedNetwork`)
2. *Testar condições iniciais* (perfil `fNet`: OK)
3. *Finalizar configuração corrente*

   Verificar operações úteis no histórico: *util* (C') = { `OP1`, `OP2` }

   Finalização: Desfazer `OP3` :`unlink AP from PS2`

   Não desfazer `OP2`

   Não desfazer `OP1`
4. *Ativar configuração selecionada*:

   `assert Client as AP at <clientHost>` → OK (aproveitada)

   histórico: `OP1 - assert Client as AP at <clientHost>`

   `assert NetworkConnection as F with fNet` → FALHOU!

   Tratamento: Abortar transação corrente

   Iniciar nova transação *Implantar Nova Configuração*

Execução da nova transação *Implantar Nova Configuração*:

1. *Selecionar próxima configuração* (selecionada: C' = `wirelessNetwork`)
2. *Testar condições iniciais* (perfil `wNet`: OK)
3. *Finalizar configuração corrente*

   Verificar operações úteis no histórico: *util* (C') = { `OP1` }

   Finalização: Não desfazer `OP1`
4. *Ativar configuração selecionada*:

   `assert Client as AP at <clientHost>` → OK (aproveitada)

   histórico: `OP1 - assert Client as AP at <clientHost>`

   `assert NetworkConnection as W with wNet` → OK (conexão validada)

   histórico:
   ```
   OP1 - assert Client as AP at <clientHost>
   OP2 - assert NetworkConnection as W with wNet
   ```
   `assert Decrypt as DK at <clientHost>` → OK (conector validado)

   histórico:
   ```
   OP1 - assert Client as AP at <clientHost>
   OP2 - assert NetworkConnection as W with wNet
   OP3 - assert Decrypt as DK at <clientHost>
   ```
   `_provider = select Provider with providerSelection`

   instância selecionada: `PS1` (menor tempo de resposta)

   `link AP to _provider by DK, W` : OK (ligação criada: `AP–DK-W–PS1`)

   histórico:
   ```
   OP1 - assert Client as AP at <clientHost>
   OP2 - assert NetworkConnection as W with wNet
   OP3 - assert Decrypt as DK at <clientHost>
   OP4 - link AP.getQuote to PS1.getQuoteK by DK, W
   ```

Conclusão: Configuração `wirelessNetwork` ativada com sucesso (`AP–DK-W–PS1`)

Situação 4: Mudança de conexão de rede (conexão fixa disponível)

Contexto:

Configuração corrente: C = `wirelessNetwork (AP–DK-W–PS1)`

Propriedades: `PS1.ProviderNFP.responseTime = 20 ←`
`PS2.ProviderNFP.responseTime = 25`

Histórico:
```
OP1 - assert Client as AP at <clientHost>
OP2 - assert NetworkConnection as W with wNet
OP3 - assert Decrypt as DK at <clientHost>
OP4 - link AP.getQuote to PS1.getQuoteK by DK, W
```

Execução da transação *Implantar Nova Configuração*:

1. *Selecionar próxima configuração* (selecionada: C' = `fixedNetwork`)
2. *Testar condições iniciais* (perfil `fNet`: OK)
3. *Finalizar configuração corrente*

   Verificar operações úteis no histórico: *util* (C') = { `OP1` }

   Finalização: Desfazer `OP4` : `unlink AP from PS1`

   Desfazer `OP3` : `remove DK from <clientHost>`

   Desfazer `OP2` : `remove W`

   Não desfazer `OP1`
4. *Ativar configuração selecionada*:

   `assert Client as AP at <clientHost>` → OK (aproveitada)

   histórico: `OP1 - assert Client as AP at <clientHost>`

   `assert NetworkConnection as F with fNet` → OK (conexão validada)

   histórico:
   ```
   OP1 - assert Client as AP at <clientHost>
   OP2 - assert NetworkConnection as F with fNet
   ```
   `_provider = select Provider with providerSelection`

   instância selecionada: `PS1` (menor tempo de resposta)

`link AP to _provider by F` : OK (ligação criada: `AP–F–PS1`)

histórico:
```
OP1 - assert Client as AP at <clientHost>
OP2 - assert NetworkConnection as F with fNet
OP3 - link AP.getQuote to PS1.getQuoteN by F
```

Conclusão: Configuração `fixedNetwork` ativada com sucesso (`AP–F–PS1`)

6. Análise do exemplo e conclusões

As situações descritas na Seção 5 cobrem as possibilidades para adaptação citadas na Seção 4.1: primeira configuração (Situação 1), falha em componentes (Situação 3) e invalidação de requisitos de qualidade (Situações 2 e 4). O agrupamento de transações em níveis permite interromper uma transação mais interna e tratar a interrupção – no caso, uma exceção de software – sem prejuízo para a transação que a envolve. Assim, é possível manter a propriedade de atomicidade da transação mais externa.

Com a captura de falhas a cada execução de uma operação básica, é possível interromper a transação de implantação de uma configuração (2º. Nível), e desfazer as operações já realizadas, caso não sejam úteis à nova configuração. O histórico de operações concluídas serve a esse propósito, funcionando como uma pilha. Como uma nova transação de 2º. Nível é iniciada a cada falha, garante-se que a transação mais externa (1º. Nível) só poderá terminar de duas maneiras: ou uma nova configuração é implantada com sucesso, ou é sinalizada a impossibilidade de implantar qualquer configuração, com a ocorrência da Exceção 1 (Nenhuma Configuração Disponível). Assim a atomicidade da adaptação é garantida, e logo, a manutenção da consistência.

O modelo de adaptação proposto é aplicável a qualquer tipo de arquitetura que possa ser descrita segundo o modelo de componentes de CR-RIO. Incluem-se aí os estilos arquiteturais mais comuns, como o cliente-servidor ilustrado no exemplo, e suas variações como P2P, em que componentes são clientes e servidores ao mesmo tempo,e *pipeline*, em que a saída de um componente é ligada à entrada de outro. O ponto chave é ter disponíveis as descrições arquiteturais e contratos de qualidade de serviço. É possível a utilização de contratos dinâmicos, que se apóiam em serviços de descoberta de componentes e técnicas de otimização e inteligência computacional, por exemplo.

Dois pontos importantes não foram cobertos pelo escopo deste artigo: a possibilidade de execução concorrente dos passos da adaptação, e a possibilidade de falha e reinício do gerenciador de configuração. Para fins de simplificação adotou-se a execução sequencial da adaptação e assumiu-se a robustez do gerenciador de configuração. Tais pontos, porém, merecem destaque em um trabalho mais amplo.

Direções futuras para o trabalho apontam para a aplicação em uma extensão do *framework* CR-RIO para configuração de serviços Web [W3C 2008] e tecnologias associadas: descrição de serviços com WSDL (*Web Service Description Language*), descoberta de recursos com UDDI (*Universal Description Discovery and Integration*) e interação entre componentes através de SOAP (*Simple Object Access Protocol*).

7. Referências

Batista, T., Joolia, A. e Coulson, G. (2005). “Managing Dynamic Reconfiguration in Component-Based Systems”. *Lecture Notes in Computer Science, Vol. 3527/2005*, Springer.

Beugnard, A., Jézéquel, J.-M., Plouzeau, N. e Watkins, D. (1999). “Making Components Contract Aware”. *IEEE Computer, 32(7):38-45*.

Capra, L., Zachariadis, L. e Mascolo, C. (2005). “Q-CAD: QoS Context-Aware Discovery Framework for Adaptive Mobile Systems”. Em *IEEE International Conference on Pervasive Services 2005*. Santorini (Grécia), julho.

Cardoso, L., Sztajnberg, A. e Loques, O. (2006). “Self-adaptive Applications Using ADL Contracts”. Em *SelfMan 2006 – 2 nd . IEEE International Workshop on Self-Managed Networks, Systems and Services*. Dublin (Irlanda), junho.

Garlan, D., Cheng, S., Huang, A., Schmerl, B. e Steenkiste, P. (2004) “Rainbow: Architecture Based Self-Adaptation with Reusable Infrastructure”. *IEEE Computer, 18(10): 46-54*.

Gray, J. e Renter, A. (1993). *Transaction Processing: Concepts and Techniques*. Kaufman Publishers.

Kephart, J. O. e Chess, D. M. (2003). “The Vision of Autonomic Computing”. *IEEE Computer, 36(1):41-50*.

Kon, F. e Campbell, R. H. (2000). “Dependence Management in Component-Based Distributed Systems”. *IEEE Concurrency, 8(1):26-36*.

Kramer, J. e Magee. J. (1985). “Dynamic Configuration for Distributed Systems”, *IEEE Transactions on Software Engineering, SE-11(4):424-436*.

Loyall, J. P., Schantz, R. E., Zinky, J. A. e Bakken, D. E. (1998). “Specifying and Measuring Quality of Service in Distributed Object Systems”. Em *The 1st IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*. Kyoto (Japão), abril.

Lynch, N. A., Merrit, M., Weihl, W. E. e Fekcte, A. (1993). *Atomic Transactions*, Morgan Kaufman.

Meyer, B. (1992). “Appliying ‘Design by Contract’”. *IEEE Computer, 25(10):40-51*.

Microsoft Corporation (acessado em abril de 2008). *Microsoft's Developer News (MSDN) Library*. http://msdn.microsoft.com/library/default.asp

OMG - Object Management Group (acessado em abril de 2008). *Catalog of OMG Specifications*. http://www.omg.org/technology/documents/spec_catalog.htm

Romanovsky, A. (2001). “Coordinated Atomic Actions: How To Remain ACID in the Modern World”, *Software Engineering Notes 26(2):66-68*, ACM SIGSOFT.

Shaw, M. e Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*, 1 st . ed., Prentice Hall.

Sun Microsystems (acessado em abril de 2008). *Developer Services: Information about Java technology*. http://developer.java.sun.com

W3C - World Wide Web Consortium (acessado em abril de 2008). *Web Services Architecture.* http://www.w3.org/TR/ws-arch/