5

Ferramentas para Adaptação Dinâmica

Como visto no capítulo 3, diversas técnicas têm sido propostas para auxiliar o desenvolvimento de aplicações adaptáveis, em especial para aplicações baseadas em componentes. Da mesma forma que existem diversas técnicas de adaptação, também existem diferentes necessidades de adaptação para diferentes tipos de aplicações. Técnicas de adaptação devem levar em consideração as necessidades de cada aplicação, como disponibilidade, eficiência, evolução, entre outras. Inclusive, numa mesma aplicação, pode haver a necessidade de aplicar técnicas distintas na flexibilização de diferentes partes do sistema. Portanto, um conjunto de ferramentas integrado, capaz de fornecer abstrações e mecanismos que permitam utilizar diferentes técnicas de adaptação dinâmica de aplicações, se mostra adequado como mecanismo de suporte ao desenvolvimento de aplicações adaptáveis.

No capítulo anterior foram apresentados os recursos do LOAF para construção de componentes dinamicamente adaptáveis. Neste capítulo, são apresentadas as ferramentas do LOAF que são utilizadas pelos clientes ou pelos administradores do sistema para adaptar dinamicamente as aplicações através da manipulação de seus componentes. As ferramentas de adaptação apresentadas são duas: uma ferramenta para controlar alterações nos componentes LuaCCM através do conceito de papéis, como será visto na seção 5.1 e uma ferramenta de configuração de componentes CCM apresentada na seção 5.2.

Todas as ferramentas de adaptação do LOAF são construídas em Lua, através dos mecanismos de extensão da linguagem. Através de abstrações e da própria simplicidade da linguagem Lua, essas ferramentas facilitam a utilização dos recursos reflexivos do modelo LuaCCM, assim como a manipulação de componentes CCM em geral. Essas ferramentas de adaptação permitem criar aplicações compostas de componentes CCM e realizar alterações em ponto pequeno através da adaptação do conjunto de componentes dinâmicos LuaCCM que compõem a aplicação, assim como fazer alterações em ponto grande através de reconfigurações.

5.1
Papéis e Protocolos

As interfaces de adaptação do LuaCCM são um mecanismo relativamente básico de adaptação, já que não definem abstrações que permitam introduzir alterações de forma mais estruturada. Nesse sentido, em [14] são propostas duas abstrações utilizadas para controlar alterações de comportamento em componentes de software num *middleware* orientado a eventos denominado Comet. Essas abstrações são os conceitos de *protocolo* e *papel*.

Um papel é uma abstração utilizada para definir o conjunto de alterações às quais um componente deve ser submetido para que este possa oferecer um novo comportamento (ou "desempenhar um novo papel"). Em particular, um papel define essas alterações através de um conjunto de novos conectores e a semântica relacionada a esses conectores através de código executável. Já um protocolo é uma abstração utilizada para aplicar efetivamente novos papéis a um ou mais componentes do sistema, de tal forma que o sistema como um todo seja adaptado a uma nova configuração. Isso é feito através de roteiros (*scripts*) que estabelecem novas conexões entre os componentes (*e.g.*, utilizando as características impostas por um papel) e disparam eventos ou requisições. Adicionalmente, um protocolo pode possuir um estado interno que pode ser acessado dentro dos roteiros do protocolo.

Esses dois conceitos podem ser adaptados ao modelo CCM através dos recursos reflexivos do LuaCCM. Um papel é basicamente responsável por definir uma alteração em um componente, para que este possa participar de um novo tipo de interação (*i.e.*, realizando um novo papel). No modelo CCM essa alteração é caracterizada pela mudança da estrutura do componente, ou seja, definição de novas portas do componente ou alteração da semântica (*i.e.*, implementação) das portas existentes, de forma que novas interações possam ser feitas através dessas portas. A interface de adaptação do LuaCCM permite adicionar portas fornecendo código que define a semântica das interações através dessas portas, assim como também permite alterar a semântica das portas existentes através do uso de interceptadores. Com base nesses recursos, é definida uma ferramenta baseada no conceito de papel, onde é possível definir portas e interceptadores a serem aplicados a um componente. A figura 5.1 ilustra a estrutura de definição de um papel.

Depois de definido, um papel pode ser aplicado a um ou mais componentes através da operação assign, que recebe o componente a ser alterado como parâmetro. Quando o papel é aplicado a um componente LuaCCM, todas as portas definidas no papel são adicionadas à estrutura do componente e novos segmentos são definidos de acordo com os trechos de código Lua fornecidos na definição do papel. Todas essas alterações realizadas pelo papel são feitas através da interface de adaptação do LuaCCM. Assim como um papel pode ser aplicado a um componente, ele também pode ser removido através da operação unassign.

Como é possível aplicar papéis em componentes e estabelecer novas

```
new_role = loaf.Role {
  -- Definicao de novas portas
  provides = {
    nova_faceta = {
      interface = "::InterfaceDaFaceta",
      code = [[{
        operacao = function(self)
          -- Implementacao da operacao
        end,
      }]],
    },
  },
  uses = {
    novo_receptaculo = {
      interface = "::InterfaceDoReceptaculo",
      multiple = true,
    },
  },
  emits = {
    novo_emissor = {
      event = "::EventoEmitido",
    },
  },
  publishes = {
    novo_publicador = {
      event = "::EventoPublicado",
    },
  },
  consumes = {
    novo_consumidor = {
      event = "::EventoConsumido",
      code = [[{
        push_EventoConsumido = function(self, event)
          -- Implementacao do tratador de eventos
        end
      }]],
    },
  },
  -- Interceptacao de portas
  before = {
    porta_existente = {
      code = [[function(self, request)
          -- Interceptacao antes da chamada.
      end]],
    },
  },
  after = {
    porta_existente = {
      code = [[function(self, request)
          -- Interceptacao apos a chamada.
      end]],
    },
  },
}
```

Figura 5.1: Exemplo de definição de papel no LOAF.

conexões entre eles através de *scripts* Lua, estes são capazes de representar naturalmente a abstração de protocolos, podendo inclusive representar um estado interno do protocolo através de variáveis locais do *script*. Por essa razão, o LOAF não necessita definir nenhum suporte específico para o conceito de protocolo. Além disso, a ferramenta de manipuladores descrita na próxima seção simplifica consideravelmente a implementação de protocolos em Lua.

5.2 Manipuladores

Uma das motivações por trás do desenvolvimento da linguagem Lua é a necessidade de linguagens para configuração de sistemas [33]. Nesse sentido, diversos trabalhos mostram como as características da linguagem se adequam a esse propósito, inclusive na configuração dinâmica de sistemas de objetos CORBA [36]. Com a proposta do modelo componentes CORBA com mecanismos para definição de dependências entre componentes, as tarefas de composição, configuração e gerência de dependências dos componentes de uma aplicação se tornam mais fáceis. Entretanto, as interfaces de configuração do modelo CCM ainda apresentam certo grau de dificuldade e limitações.

Uma das dificuldades na manipulação de conexões CCM é a utilização de diversas operações com assinaturas diferentes, dependendo do tipo de conector associado. Por exemplo, para estabelecer uma conexão através de um publicador de eventos é necessário utilizar uma operação como subscribe, já no caso de um emissor de eventos a operação utilizada é similar a connect. Em outras palavras, as interfaces de manipulação, apesar de bem especificadas e funcionais, são razoavelmente extensas e complexas (vários nomes de operações diferentes). Isso faz com que tanto o desenvolvedor do componente quanto o desenvolvedor de aplicações tenham que se familiarizar com os termos e as regras que regem as interfaces de manipulação de conexões CCM.

Para facilitar essa manipulação, o LOAF define um mecanismo denominado manipulador (*handler*). Esse mecanismo utiliza os recursos de extensão de Lua aliados às interfaces de introspecção do modelo CCM para mascarar as interfaces de conexão dos componentes, de forma que as portas possam ser manipuladas como atributos de um objeto Lua. Por exemplo, para estabelecer a conexão entre o receptáculo comm_bus do componente de envio de mensagens messager e a faceta socket do componente de transmissão de dados network através das interfaces CCM, seria necessário escrever a seguinte linha de código:

```
messager:connect_comm_bus(network:provide_socket())
```

Através do uso dos manipuladores do LOAF, essa mesma tarefa poder ser feita da seguinte forma:

```
messager.comm_bus = network.socket
```

Essa forma se mostra muito mais intuitiva, uma vez que representa melhor a idéia de que o componente de mensagem utiliza como meio de comunicação o recurso de transmissão por sockets do componente network. Da mesma forma, é possível obter a interface conectada ao receptáculo através da leitura do campo comm_bus, assim como desfazer a conexão atribuindo ao campo o valor nil. Note portanto, que nesse caso, a ordem da atribuição é importante, pois caso contrário, representaria uma atribuição da interface conectada ao receptáculo comm_bus ao campo socket, que representa uma faceta.

Já, no caso de canais de eventos, essa ordem da atribuição pode não parecer natural. Suponha, por exemplo, que o componente de mensagens notifique o recebimento de eventos através de um emissor denominado out. Para conectar um componente de impressão de mensagens ao componente de mensagens através de um canal de eventos, o código utilizado poderia ser:

```
messager.out = printer.in
```

Nesse caso, a menos que seja levada em consideração a forma de implementação de canais de eventos CCM, a ordem da atribuição não parece lógica. Por essa razão, os manipuladores do LOAF permitem a inversão da atribuição no caso de conectores orientados a eventos. Isso é possível, pois o modelo CCM não define operações para obter o consumidor de eventos conectado a um emissor (fonte de evento com apenas um único consumidor). Portanto, a leitura do campo out não devolve um consumidor de eventos, mas sim um objeto Lua que representa a fonte de eventos.

Outra dificuldade na manipulação de conexões CCM é o uso de identificadores de conexão (*cookies*) gerados automaticamente pela infra-estrutura (*i.e.*, contêiner). Dessa forma, os identificadores não apresentam nenhum significado para a aplicação, o que obriga que as aplicações mantenham um mapeamento dos identificadores com a semântica das conexões. Por exemplo, suponha uma aplicação de publicação de notícias, em que um componente de envio de mensagens é criado para cada usuário e esses são conectados com um o componente produtor de notícias. Nessa situação, a aplicação deve fazer o mapeamento dos nomes dos usuários para cada identificador de conexão, de forma que quando o usuário não tiver mais interesse no serviço, a conexão adequada possa ser desfeita.

Para contornar essa dificuldade, os manipuladores do LOAF fazem o mapeamento automático dos identificadores de conexão CCM para valores Lua. Ou seja, quando uma conexão é estabelecida, um valor Lua é utilizado como índice da conexão, fazendo com que os conectores CCM funcionem como os vetores associativos de Lua. Por exemplo, ainda no caso de um componente produtor de mensagens denominado reporter, que envia notícias através de um publicador de eventos (fonte de evento com vários consumidores) para os componentes de recebimento de mensagens de cada usuário, os manipuladores do LOAF permitem estabelecer canais de eventos entre esses componentes da seguinte forma:

```
reporter.outbox["jose"]   = jose.mailbox
reporter.outbox["maria"]  = maria.phonebox
reporter.outbox["carlos"] = carlos.smsbox
```

Internamente o manipulador guarda o identificador da conexão gerado pelo componente e mantém o mapeamento para o valor Lua fornecido como índice da conexão. Entretanto, uma vantagem do uso de identificadores gerados automaticamente, é a capacidade de poder estabelecer conexões sem a necessidade de definir um valor de índice. Isso é útil quando se deseja estabelecer uma conexão sem a intenção de desfazê-la posteriormente. Para esse caso, os manipuladores do LOAF permitem estabelecer conexões sem utilização de índices. Internamente um índice numérico é automaticamente gerado de acordo com o maior índice numérico utilizado por outra conexão. Por exemplo, se o componente reporter do exemplo anterior não possuir nenhuma conexão, então o código seguinte estabelece as conexões utilizando os índices numéricos 1, 2 e 3.

```
reporter.outbox = jose.mailbox
reporter.outbox = maria.phonebox
reporter.outbox = carlos.smsbox
```

Note que nesse caso, o código não parece natural pois sugere a idéia de substituição da conexão estabelecida com a fonte de eventos outbox, entretanto não é isso que ocorre. Portanto, em alguns casos a inversão da atribuição é mais natural, como quando um receptor de eventos recebe um publicador. Para tanto, o mesmo recurso de conexão sem definição de índice é utilizado para permitir a inversão da atribuição. Com a possibilidade de inversão, então o código acima poderia ser reescrito da seguinte forma:

```
jose.mailbox   = reporter.outbox
maria.phonebox = reporter.outbox
carlos.smsbox  = reporter.outbox
```

Uma das grandes limitações do modelo de componentes CCM é o suporte incompleto à rastreabilidade de dependências entre componentes [37]. Como as informações sobre as conexões são armazenadas apenas nos receptáculos e fontes de eventos, não é possível rastrear as dependências a partir de componentes que fornecem um serviço através de facetas ou receptores de eventos. Dessa forma, quando um componente é substituído ou o serviço através de uma faceta é descontinuado, não há mecanismos que permitam identificar quais componentes dependem do componente ou faceta e realizar as adaptações adequadas, como enviar notificações do encerramento do serviço ou substituição de referências.

Através da configuração de sistemas com uso de manipuladores LOAF, essas informações sobre as dependências entre os componentes podem ser capturadas. Entretanto, não há formas de associar essa informação aos componentes CCM (*e.g.*, quais componentes dependem de uma faceta). Apesar dos

recursos de adaptação dos componentes LuaCCM permitirem estender o componente para que essa informação possa ser armazenada, esse recurso não estaria presente em todos os componentes CCM. Além do mais, não há garantia de que em um dado sistema, todas as conexões entre componentes seriam feitas através de manipuladores LOAF. Portanto, como não há forma de fornecer esses recursos uniformemente entre todos os componentes CCM, esses recursos ainda não são oferecidos pelos manipuladores LOAF.