4
Camada Publish-Subscribe

A camada publish-subscribe, implementada no módulo *luaps*, disponibiliza as funções necessárias para um sistema publish-subscribe completo. Esta camada disponibiliza funções para entrada e saída da rede publish-subscribe, criação de tópicos, inscrição e remoção de inscrições em tópicos e publicação de notificações. Para garantir a execução correta das funcionalidades citadas, a camada publish-subscribe gerencia os tópicos e as inscrições, implementando estruturas particulares que não são acessadas por outras camadas.

Para oferecer maior flexibilidade, permitindo que o sistema publish-subscribe seja utilizado por diferentes aplicações, a função de inscrição de um cliente em um tópico recebe como parâmetro uma função genérica de publicação. Esta função de publicação especifica o processo de publicação de uma mensagem do tópico em questão para o dado cliente. Desta forma, o projeto LuaPS será adequado tanto para sistemas onde a publicação de uma mensagem é implementada como o simples envio do texto para os clientes ou por sistemas onde as mensagens passam por tratamentos mais elaborados, como filtros por conteúdo, por exemplo. Neste capítulo, explicamos a estrutura utilizada para armazenamento e execução da função de publicação, permitindo a definição de uma função genérica.

Um dos principais objetivos do sistema LuaPS é o desenvolvimento de um sistema publish-subscribe flexível. A utilização de uma função de publicação genérica contribui para este objetivo, permitindo que as camadas superiores do sistema definam a metodologia de publicação das notificações. Além da definição da função de publicação, um sistema mais completo pode utilizar a arquitetura básica disponibilizada pela camada publish-subscribe e estender o sistema com a definição de uma camada de serviços.

Quando acontece alguma mudança na gerência dos tópicos, todas as informações referentes a um tópico realocado também devem ser realocadas para o novo nó. A camada publish-subscribe também é responsável pela manutenção do estado consistente das informações em qualquer instante, mesmo considerando a entrada e saída de nós da rede a qualquer momento.

Para garantir a execução das funções de forma consistente, a camada publish-subscribe implementa uma função para a gerência da execução dos comandos, garantindo que os comandos aguardem pela estabilidade das informações antes de serem executados. Genericamente, a função de gerenciamento da execução de comandos define um processo que permite que comandos sejam executados imediatamente ou buferizados para posterior execução dependendo do estado da aplicação.

Embora a camada publish-subscribe gerencie internamente a execução dos comandos publish-subscribe, novas funcionalidades que venham a ser desenvolvidas na camada de serviços também teriam que considerar esta necessidade. Para facilitar a extensão do sistema, a camada publish-subscribe disponibiliza na interface a função que gerencia a execução dos comandos. Desta forma, a utilização desta função garante que a camada de serviços estará executando os comandos somente em momentos de estabilidade das informações.

Ainda considerando a extensibilidade com a construção de uma camada de serviços, a camada publish-subscribe disponibiliza funções que permitem associar novos campos à estrutura de gerência dos tópicos. Desta forma, a camada assume a responsabilidade de realocar as informações associadas aos tópicos quando ocorrem entradas ou saídas de nós da rede. Evitamos, portanto, que a nova camada precise considerar questões referentes à localização de informações armazenadas. Genericamente, as funções definidas permitem associar campos aos tópicos gerenciados pelos nós e obter os valores armazenados.

Segue uma lista com as funções disponibilizadas e uma breve descrição:

- startPS() - Inicia uma rede publish-subscribe.
- joinPS(no,callback) - Entra em uma rede publish-subscribe.
- leavePS(aplicacao) - Sai de uma rede publish-subscribe.
- createTopic(topico,callback) - Solicita a criação de um novo tópico.
- subscribe(topico,funcao) - Solicita inscrição com a função de publicação.
- unSubscribe(topico) - Solicita a exclusão da inscrição em um tópico.
- publish(topico,texto) - Solicita a publicação de uma notificação.
- handleCommand(cmd,topico) - Gerencia execução de um comando.
- setTopicField(topico,campo,tabela) - Associa um campo a um tópico.
- getTopicField(topico,campo) - Retorna campo associado a um tópico.

Na primeira seção do capítulo iremos definir a estrutura utilizada para a gerência dos tópicos e inscrições. Na seção seguinte, iremos apresentar a função de publicação, destacando sua estrutura genérica. Em seguida, iremos discutir o protocolo definido para entrada e saída dos nós da rede, garantindo a consistência das informações. Na seção final, apresentaremos a implementação da função de gerência dos comandos.

4.1 Gerência sobre Tópicos e Inscrições

Ao definirmos a camada pastry, apresentamos o protocolo Pastry e o sistema LuaPastry, que disponibiliza uma tabela hash distribuída que pode ser utilizada para o desenvolvimento de sistemas distribuídos. No protocolo Pastry, o roteamento de uma mensagem na rede é feito utilizando os IDs dos nós e a chave que identifica o destino da mensagem. Segundo o protocolo, a mensagem será entregue para o nó cujo ID mais se aproximar da chave da mensagem.

A camada publish-subscribe foi desenvolvida utilizando a camada pastry e, desta forma, cada nó da rede tem um ID único. O sistema luaps é baseado em tópicos e cada tópico é unicamente identificado utilizando o hash de seu nome. A criação de um tópico na rede é feita com o envio da solicitação para o nó que será responsável pela sua gerência. Para associarmos os nós aos tópicos, utilizamos os IDs dos nós e o hash do nome do tópico. Desta forma, podemos utilizar as funções de roteamento de mensagens do Pastry para encaminhar qualquer solicitação para o nó correspondente utilizando o hash do nome do tópico como chave da mensagem.

Ao adotarmos o uso de uma tabela hash distribuída para implementar a infra-estrutura básica do LuaPS, estamos admitindo que a camada publish-subscribe não irá controlar a distribuição dos tópicos entre os nós da rede. Isto é, uma vez que o hash de um tópico será utilizado para determinar o nó que deve assumir a sua gerência, a camada pastry do sistema será responsável por esta distribuição. Desta forma, além de poder utilizar as funções disponibilizadas pelo sistema como clientes publish-subscribe, todos os nós do sistema devem estar aptos a gerenciar tópicos e inscrições.

A camada disponibiliza funções para criação de tópicos, inclusão ou exclusão de inscrições e publicação de notificações. Cada uma dessas funções, pela natureza distribuída, é composta por funções executadas no âmbito local e por funções executadas em nós remotos. A solicitação de execução de qualquer

função se inicia no cliente, com a execução de uma das funções disponibilizadas pelo sistema. Uma vez que, normalmente, os tópicos não estão armazenados no próprio nó, a solicitação local irá gerar uma mensagem que será enviada para o nó gerente do tópico. O nó gerente do tópico, então, irá executar a função propriamente dita.

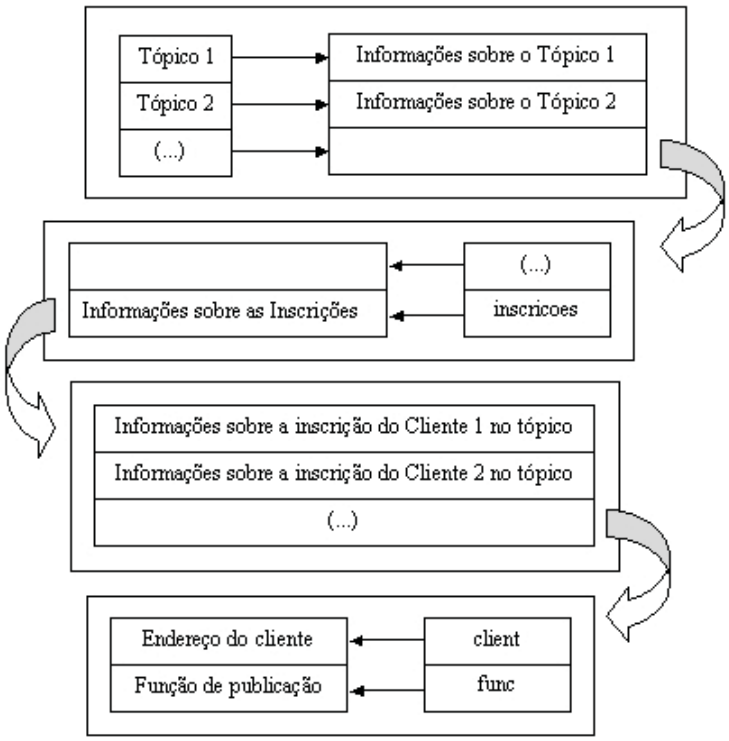

Figura 4.1: Estrutura utilizada na Gerência dos Tópicos

Para controlar os tópicos gerenciados por cada nó, definimos uma estrutura que armazena as informações referentes a um tópico, ilustrada na figura 4.1. Esta estrutura foi implementada como uma tabela que armazena referências. A implementação genérica da estrutura possibilita que campos com novas informações sejam associados aos tópicos por camadas superiores do sistema. Esta possibilidade será analisada no capítulo referente à camada de serviços.

O acesso a um tópico particular gerenciado pelo nó é feito utilizando o nome do tópico como índice da tabela. As inscrições em um tópico podem ser acessadas através do campo *inscricoes*. A estrutura que armazena as inscrições é composta por dois campos, utilizados na publicação de uma notificação. O primeiro campo identifica o cliente que fez a inscrição, armazenando seu endereço. O segundo campo armazena a função de publicação, responsável por definir e implementar o processo de publicação. Desta forma, uma nova

inscrição é feita com a adição de um registro à tabela de inscrições do tópico e a remoção de uma inscrição com a retirada do valor. Da mesma forma, a publicação de notificações é feita com o envio da notificação para todos os clientes existentes na tabela.

4.2 Função de Publicação

A função de publicação, passada como parâmetro no momento da inscrição de um cliente em um tópico, tem o objetivo de especificar o processo de notificação de uma mensagem ao cliente. A utilização de uma função genérica possibilita que o sistema LuaPS seja utilizado por diversas aplicações.

A utilização de funções genéricas para prover flexibilidade também é utilizada em outros sistemas, como o LuaTS (Leal et al., 2003). Neste sistema, um ambiente de espaço de tuplas reativo é implementado e funções genéricas permitem a execução da reação definida.

A utilização de uma função de publicação genérica, entretanto, implica em questões de segurança que devem ser destacadas. A flexibilidade proposta permite que qualquer código seja executado, tanto no cliente publish-subscribe quanto no nó gerente do tópico. Desta forma, não há controle sobre a execução de código que possa intervir na consistência das informações referentes aos tópicos. Para minimizar este problema, poderíamos considerar que a função de publicação seria sempre executada no cliente. Uma outra possibilidade seria a utilização de uma sandbox, isto é, um repositório com recursos limitados onde o código seria executado, como em (Boyer & Griswold, 2005), por exemplo. Desta forma, entretanto, diminuiríamos a flexibilidade do sistema. Optamos pelo maior grau de flexibilidade possível na camada publish-subscribe, possibilitando, entretanto, que as camadas superiores, ao definirem a função de publicação, considerem limitações visando o aumento de segurança necessário para a aplicação sendo desenvolvida.

A associação da função de publicação à inscrição de um cliente em um tópico traz flexibilidade ao sistema. Primeiramente, possibilita a coexistência de diferentes aplicações na mesma rede LuaPS. Podemos considerar, por exemplo, uma rede LuaPS com clientes normais executando uma certa aplicação e com clientes que funcionarão como gateways executando uma aplicação específica. Esta situação será explorada mais adiante. Além disto, facilita a criação de filtros por conteúdo, como discutido na próxima seção.

4.2.1 Definição e Execução da Função de Publicação

A função de publicação é passada como parâmetro no momento da inscrição de um cliente em um tópico. Sempre que uma publicação de notificação é solicitada, a camada publish-subscribe percorre a lista de clientes inscritos do dado tópico, acessa as informações armazenadas e realiza a publicação da notificação.

A função de publicação é uma string com um trecho de código Lua genérico, podendo conter diversos métodos e estruturas. Para que a camada publish-subscribe possa executar o código no momento da publicação, precisamos especificar um ponto de entrada inicial no código. Definiremos, então, que o processo de publicação se dará com a execução do método *run*, recebendo como parâmetros o endereço do cliente e a mensagem a ser publicada.

```
funcPrint = [[ function run( id, msg )
                  alua.send( id, [[ luaps.imprime( "]] .. msg .. [[\n" ) ]] )
               end ]]
```

Figura 4.2: Função de Publicação Simples

A figura 4.2 ilustra uma função de publicação simples. A função implementa somente o método *run* que, ao ser executado, envia a mensagem a ser publicada para ser impressa na tela do cliente. No trecho de código podemos perceber algumas notações Lua para strings: a concatenação de strings é feita com a utilização de dois pontos em seqüência e colchetes duplos, que iniciam e terminam uma string, permitem a utilização de strings internas a outras strings.

4.2.2 Filtro sobre as Notificações

Na introdução sobre publish-subscribe comentamos que os sistemas publish-subscribe baseados em conteúdo são os considerados mais expressivos. Ao discutirmos a gerência dos tópicos no início deste capítulo, entretanto, definimos o sistema LuaPS seria baseado em tópicos, por ser mais adequado à utilização de uma tabela hash distribuída. A utilização de uma função de publicação genérica, entretanto, nos permite aliar as vantagens da utilização

de uma infra-estrutura baseada em uma tabela hash distribuída com a expressividade do filtro por conteúdo.

Para ilustrar a afirmação, consideremos um exemplo. Suponhamos que um jornal resolveu disponibilizar um serviço publish-subscribe para seus assinantes. Através de um software instalado nas máquinas, os clientes podem entrar na rede publish-subscribe para receber informações publicadas pelo jornal ou publicar informações, por exemplo, de classificados.

A tela inicial do software permite que o usuário determine suas áreas de interesse dentre todas as áreas disponibilizadas, como esportes, economia, televisão e classificados, por exemplo. Em função do perfil, o usuário terá direitos particulares em cada uma das áreas. Um assinante, por exemplo, não poderá publicar notícias de economia, mas poderá publicar avisos nos classificados e receber mensagens de todas as áreas. Os jornalistas, por outro lado, podem publicar ou receber mensagens de todas as áreas, exceto classificados. Ao escolher suas áreas de interesse o usuário estará, na verdade, realizando inscrições nos tópicos desejados.

Após a seleção das áreas de interesse, os usuários podem especificar filtros que serão utilizados no momento do envio das mensagens. Um assinante pode, por exemplo, especificar inscrições nas áreas de esportes e classificados. Notemos, portanto, que notícias de economia não serão enviadas para o usuário. Na área de classificados, o usuário, que deseja comprar um carro, especifica que somente as mensagens que possuam a palavra *carro* devem ser enviadas. Desta forma, a relevância das mensagens recebidas passará a ser muito maior. Na área de esportes, o usuário especifica que deseja receber todas as mensagens sobre futebol referentes a seu time e que não contenham a palavra *derrota*, evitando notícias desagradáveis.

Uma vez que o usuário selecione as palavras que irão filtrar as mensagens, o software irá traduzir a lista de palavras em funções de publicação que serão associadas às inscrições nos tópicos. As funções irão determinar se as palavras listadas estão contidas no texto da mensagem e, no momento da publicação, somente as mensagens que satisfizerem os requisitos serão entregues.

O sistema LuaPS, portanto, utiliza uma forma de executar filtro sobre as notificações que combina o nome do tópico para selecionar os assuntos desejados e a função de publicação para filtrar as notificações levando em consideração seu conteúdo.

A figura 4.3 ilustra uma função de publicação utilizada para filtrar as mensagens. A função descarta todas as mensagens que não possuem a palavra *carro*, conforme desejado pelo usuário do exemplo.

```
funcFiltro = [[ function run( id, msg )
                    if string.find( msg, "carro" ) ~= nil then
                       alua.send( id, [[ luaps.imprime( "]] .. msg .. [[\n" ) ]] )
                    end
                end ]]
```

Figura 4.3: Função de Publicação utilizada como Filtro

4.3 Entradas e Saídas de Nós na Rede Publish-Subscribe

Na rede Pastry, utilizada como substrato para a rede publish-subscribe, cada nó possui um identificador único e cada tópico possui uma chave, gerada a partir do hash do nome do tópico. A gerência de cada tópico é atribuída ao nó cujo identificador mais se aproximar de sua chave. Desta forma, a entrada de novos nós na rede podem significar que os gerentes por certos tópicos sejam alterados.

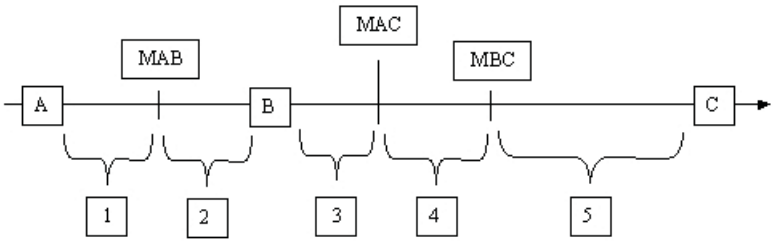

Figura 4.4: Mudança na Gerência dos Tópicos com a Entrada de Nós na Rede

Após a entrada de um nó da rede, os tópicos que podem mudar de gerência apresentam uma mesma estrutura, destacada na figura 4.4. No esquema ilustrado, os nós A e C existem previamente na rede publish-subscribe e o novo nó B deseja entrar na rede. O ID dos nós é de tal forma que o ID novo nó B é maior que o ID de A e menor que o ID de C. Desta forma, B está localizado entre A e C na reta, não necessariamente no ponto MAC, o meio entre A e C. Antes da entrada do nó B na rede, os tópicos cujo hash do nome se localiza antes de MAC, isto é, em 1, 2 ou 3 são gerenciados por A. Os tópicos cujo hash do nome se localiza após MAC, isto é, em 4 ou 5, são gerenciados por C. Com a entrada do nó B, alguns dos tópicos gerenciados por A e C mudarão de gerência. Considerando o nó A, todos os tópicos pertencentes ao conjunto 2 e 3 deverão passar a ser gerenciados por B. Quanto ao nó C, este deverá passar a gerência dos tópicos pertencentes ao conjunto 4 para o novo nó B. Resumidamente, quando um novo nó entra na rede é preciso que seus vizinhos

verifiquem se existem tópicos por eles gerenciados cujo hash se aproxime mais do ID do novo nó do que de seu próprio ID. Neste caso, a gerência deve ser passada para o novo nó.

Assim que o novo nó entra na rede, ele já está apto a executar comandos referentes aos tópicos que deve gerenciar. O processo de transferência dos tópicos, entretanto, pode ainda não ter terminado. Se alguma função for executada neste momento, a funcionalidade pode ser executada de forma errada ou incompleta, uma vez que o nó não terá todas as informações referentes ao tópico. Desta forma, precisamos garantir que o novo nó só passe a executar as funções sobre os tópicos quando o processo de transferência tiver terminado. Assim que o novo nó entra na rede, suas funções de publish-subscribe passam a ser buferizadas. Desta forma, em vez das funções serem executadas em um momento possivelmente inconsistente do nó, as mesmas são armazenadas e executadas quando os tópicos já tiverem sido reorganizados. Neste momento, as funções buferizadas são executadas e novas requisições não mais são buferizadas.

Figura 4.5: Protocolo para Entrada de Nós na Rede

A figura 4.5 mostra o protocolo utilizado na entrada de novos nós na rede. Assim que o nó entra na rede, as funções de publish-subscribe passam a ser buferizadas. Em seguida, são enviadas solicitações de tópicos para seus vizinhos. Cada um dos vizinhos executa a solicitação e o resultado é passado para o novo nó, que adiciona os tópicos à tabela. Quando os vizinhos tiverem executado as requisições, as funções buferizadas podem ser executadas em um estado consistente e as funções de publish-subscribe podem ser executadas normalmente. Uma vez que um nó pode ter apenas um vizinho ou mesmo nenhum vizinho (no caso de ser o primeiro nó da rede), o nó precisa determinar o número de solicitações de tópicos enviadas, consultando o conjunto de folhas. Desta forma, quando todas as solicitações forem atendidas, o processo estará terminado.

O processo de saída de um nó da rede publish-subscribe, embora seja análogo ao processo de entrada de um novo nó, não pode ser tratado exatamente da mesma forma. Assim como na entrada de um novo nó, a saída de um nó da rede implica na reorganização dos tópicos já criados. Nesta reorganização, a figura 4.4 utilizada na seção referente a entradas de novos nós na rede continua válida. No caso da saída de um nó da rede, o nó B estará saindo da rede e os tópicos por ele gerenciados serão distribuídos entre os tópicos A e C.

Quando um novo nó entra na rede publish-subscribe, o primeiro passo do processo é a entrada do nó propriamente dita. Desta forma, o nó poderia passar a receber mensagens antes de estar apto a processá-las de forma correta. Os nós vizinhos, antes responsáveis por tópicos agora gerenciados pelo novo nó, não mais recebem mensagens referentes aos dados tópicos. Desta forma, a decisão por buferizar as mensagens até que os tópicos sejam reorganizados garante a consistência da rede em todos os momentos.

No caso da saída de um nó da rede, o processo de saída de rede propriamente dito precisa ser o último passo do processo. Desta forma, o nó que está saindo da rede ainda continua recebendo mensagens que se referem a tópicos que ele não mais deveria gerenciar. Se os comandos recebidos forem executados por este nó, as informações sobre os tópicos enviadas para os vizinhos podem ficar desatualizadas. Por outro lado, os nós vizinhos, que serão responsáveis pelos tópicos em questão, podem ainda não estar aptos para executar os comandos.

Sendo assim, no processo de saída da rede, um nó deve buferizar as mensagens recebidas até que as informações sobre os tópicos por ele gerenciados sejam enviadas para os vizinhos. Assim que este processo terminar, o nó deve encaminhar as mensagens recebidas para o vizinho responsável pelo dado tópico. Somente quando não existirem mais mensagens buferizadas o nó poderá deixar a rede definitivamente.

Figura 4.6: Protocolo para Saída de Nós da Rede

A figura 4.6 mostra o protocolo utilizado na saída de nós da rede. Assim que o nó inicia o processo de saída da rede, as funções de publish-subscribe passam a ser buferizadas. Em seguida, os tópicos por ele gerenciados são enviados para seus vizinhos. Cada um dos vizinhos adiciona os tópicos recebidos e notifica o nó a respeito do fim do processo. Quando os dois vizinhos tiverem adicionado os tópicos, as funções buferizadas podem ser enviadas para os novos gerentes dos tópicos e executadas em um estado consistente. Quando não existirem mais comandos buferizados o nó pode deixar a rede.

O protocolo de saída de nós da rede é utilizado quando um nó deseja explicitamente sair da rede publish-subscribe. No caso de falhas de nós, comum ao considerarmos a mobilidade dos clientes, as informações serão perdidas. Este comportamento pode não ser adequado para algumas aplicações. Para contornar esta limitação iremos implementar um processo de tolerância a falhas na camada de serviços, permitindo que as informações armazenadas em nós que deixem a rede de forma não prevista não sejam perdidas. O desenvolvimento da tolerância a falhas permitirá que o processo de saída de um nó da rede seja simplificado, uma vez que seu vizinho já terá as informações necessárias para manutenção da consistência das informações.

A preocupação com a saída de nós da rede também existe em outros sistemas publish-subscribe, mesmo que estes tenham optado por arquiteturas diferentes da escolhida pelo sistema LuaPS. O Scribe (Rowstron et al., 2001), por exemplo, também utiliza o hash do nome do tópico para determinar o nó gerente de um tópico. Cabe ao gerente do tópico armazenar informações como a tabela de permissões de acesso. Ao contrário de nossa implementação, entretanto, as inscrições são armazenadas de forma distribuída ao longo de uma árvore de filhos iniciada no gerente do tópico. Todos os nós da rede devem enviar mensagens de tempos em tempos para seus filhos, permitindo que seus filhos determinem que os mesmos ainda estão ativos. Caso um filho pare de receber mensagens de seu pai, uma nova inscrição é solicitada, admitindo que o seu pai tenha falhado. Para garantir a tolerância a falhas do nó gerente do tópico, as informações armazenadas neste nó são replicadas em K vizinhos, que virão a assumir a gerência do tópico.

4.4 Execução e Buferização de Comandos

Quando acontece alguma mudança na gerência dos tópicos, todas as informações referentes a um tópico realocado também devem ser realocadas

para o novo nó. Para garantir a execução das funções de forma consistente, a camada publish-subscribe implementa uma função para a gerência da execução dos comandos, garantindo que os comandos aguardem pela estabilidade das informações antes de serem executados.

Seguindo o modelo de comunicação ALua tradicional, as mensagens trafegadas entre processos são strings com trechos de código Lua que são executados no momento do recebimento. Desta forma, o recebimento de um comando implica na sua execução imediata. Para possibilitar a buferização para execução posterior de um comando, iremos definir uma função chamada *handleCommand*, que será responsável por gerenciar a execução de comandos. Desta forma, chamadas a comandos cujo momento da execução seja dependente do estado da aplicação não devem ser feitas diretamente. Nestes casos, a função de gerência deve ser executada e o comando em questão passado como parâmetro.

As funções Lua são variáveis de primeira classe e, sendo assim, podem ser associadas a variáveis. Esta característica da linguagem será utilizada ao implementarmos a buferização de comandos. A função *handleCommand* não possui implementação, podendo referenciar a função de execução, *executeCommand*, ou a função de buferização, *bufferCommand*, dependendo do estado da aplicação. No caso de execução imediata, o comando passado como parâmetro para a função é carregado e executado. No caso de buferização, o comando é armazenado em uma estrutura particular para que seja executado quando a aplicação estiver em um estado consistente.

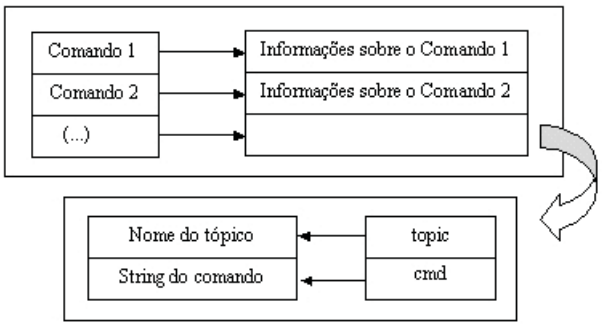

Figura 4.7: Estrutura utilizada na Gerência da Execução dos Comandos

A estrutura que representa um comando tem um campo que armazena a string do comando propriamente dita e um campo que identifica o tópico referenciado pelo comando. Cada nó, então, deve instanciar uma tabela que será povoada com estruturas deste tipo. A figura 4.7 ilustra a estrutura em questão.