5
Camada de Serviços

A camada de serviços tem como objetivo estender a camada publish-subscribe com funcionalidades que não fazem parte o paradigma básico de comunicação publish-subscribe, mas que podem ser necessários para a aplicação publish-subscribe sendo desenvolvida.

A camada de serviços é composta por diferentes módulos que disponibilizam cada um dos serviços. O módulo *luapsh* disponibiliza funcionalidades para o acesso a notificações antigas. O módulo *luapsg* disponibiliza funcionalidades de um gateway para clientes móveis, utilizando conexões persistentes. O módulo *luapsb* disponibiliza funcionalidades para tolerância a falhas. Neste capítulo iremos destacar as decisões de implementação da camada de serviços e detalhar cada um dos módulos implementados.

Como a camada de serviços busca apenas adicionar funcionalidades à camada publish-subscribe já existente, as funcionalidades da camada inferior podem continuar acessíveis. O processo de integração da camada de serviços com as outras camadas, portanto, merece atenção especial. A primeira seção do capítulo irá discutir as diferentes formas de integração existentes, buscando caracterizar os módulos que devem escolher cada uma das formas de integração. Nesta seção iremos, ainda, demonstrar a utilização das funções de integração disponibilizadas na interface da camada publish-subscribe. Terminaremos o capítulo com a descrição dos módulos de serviço implementados para o projeto.

Embora as funcionalidades implementadas sejam bastante interessantes para um sistema publish-subscribe, a principal função deste capítulo é servir como modelo para a implementação de outros módulos de serviço, guiando o desenvolvedor nas tomadas de decisões necessárias. Um outro objetivo deste capítulo é analisar a arquitetura escolhida, avaliando a extensibilidade proposta. O desenvolvimento dos módulos, portanto, busca confirmar que a separação em camadas facilita o desenvolvimento de extensões para o sistema original. Uma discussão sobre esta avaliação pode ser encontrada no fim do capítulo.

5.1 Integração com as Outras Camadas

A camada de serviços busca adicionar funcionalidades à camada publish-subscribe já existente e, sendo assim, as funcionalidades da camada inferior podem continuar acessíveis. Esta camada, portanto, além de disponibilizar novas funcionalidades, pode funcionar como intermediária entre a camada de aplicação e a camada publish-subscribe.

O acesso à camada publish-subscribe a partir da camada de aplicação pode acontecer de diferentes formas. A primeira forma de integração considera que a camada de aplicação deve interagir somente com a camada de serviços. Desta forma, todas as funções acessíveis à camada de aplicação devem estar implementadas na camada de serviços, mesmo que a implementação das funções consista apenas em uma chamada a funções da camada publish-subscribe.

Iremos chamar esta forma de integração de integração clássica, pois estamos admitindo a existência de uma pilha clássica de camadas. Esta forma de integração é particularmente interessante quando a camada de serviços deseja restringir as funcionalidades disponibilizadas pela camada publish-subscribe. Quando o serviço disponibilizado implica em alterações em grande parte das funções da interface disponibilizada pela camada publish-subscribe, este tipo de integração também pode ser bastante adequado. A figura 5.1 ilustra a integração clássica, destacando o acesso apenas entre as camadas vizinhas.

Figura 5.1: Integração Clássica

Na segunda forma de integração, que iremos chamar de integração múltipla, a camada de aplicação importa o módulo publish-subscribe e o módulo de serviços. Caberá ao módulo de aplicação, então, solicitar a execução dos métodos ao módulo correto. Quando mais de um módulo de serviços são carregados por uma aplicação, a integração múltipla pode garantir maior clareza na invocação aos métodos.

Ao considerarmos a integração múltipla estamos admitindo a existência de duas pilhas de camadas. Na primeira pilha de camadas, a camada de aplicação se integra diretamente com a camada publish-subscribe, não existindo nenhuma camada de serviço. Esta pilha de camadas estará ativa quando a camada de aplicação solicitar um método diretamente ao módulo publish-subscribe. Na segunda pilha de camadas, a camada de aplicação se comunica com a camada de serviços e esta se comunica com a camada publish-subscribe. Esta pilha estará ativa quando a camada de aplicação solicitar a execução de método da camada de serviço e esta camada executar comandos da camada publish-subscribe.

Figura 5.2: Integração Múltipla

A figura 5.2 ilustra a integração múltipla, destacando as duas possíveis formas de acesso aos métodos disponibilizados pela camada publish-subscribe: o acesso direto e o acesso através da camada de serviços.

A última forma de integração que iremos destacar será chamada de integração híbrida. Esta forma de integração mistura os principais conceitos das duas primeiras formas de integração. Por um lado, a camada de aplicação tem acesso apenas à camada de serviços, seguindo os conceitos que definem a integração clássica. Por outro lado, entretanto, a camada de serviços não precisa se preocupar em implementar os métodos disponibilizados pela camada publish-subscribe. Para evitar esta implementação, a camada de serviços redireciona as invocações para a camada publish-subscribe. Desta forma, o acesso a métodos da camada publish-subscribe é diferenciado do acesso aos métodos da camada de serviços, assim como acontece na integração múltipla.

Genericamente, podemos admitir que a camada de serviços estende a camada publish-subscribe, herdando os métodos já implementados. Embora a linguagem Lua não seja classicamente orientada a objetos, são disponibilizados mecanismos que permitem este tipo de modelagem. Maiores informações sobre herança com a utilização de metatabelas podem ser encontradas em (Ierusalimschy et al., 1996).

A figura 5.3 ilustra a integração híbrida, destacando a seção tracejada na camada de serviços. Esta seção representa o direcionamento das chamadas

Figura 5.3: Integração Híbrida

realizado pela camada de serviços, de modo que a camada de aplicação acessa funções da camada publish-subscribe diretamente.

5.2 Acesso a Notificações Antigas - Histórias

Sistemas publish-subscribe tradicionais não disponibilizam acesso a notificações antigas. Isto é, uma notificação é garantidamente entregue a um cliente se e somente se o mesmo tiver se inscrito no respectivo tópico previamente. Para possibilitar o acesso a notificações antigas, implementamos um módulo baseado no conceito de histórias. Podemos encontrar uma discussão sobre o acesso a notificações antigas em (Cilia et al., 2003) ou analisando o Rebeca (Mühl et al., 2004), um sistema que utiliza o conceito de histórias.

Para garantir o acesso a notificações antigas, é necessário que as notificações publicadas sejam armazenadas. Para isto, alteramos a implementação da publicação acrescentando o comando que nos permita realizar o armazenamento das informações. Uma vez que as mensagens são armazenadas, definimos funções que permitam aos clientes solicitar as notificações antigas.

Segue uma lista com as funções do módulo de histórias e uma breve descrição:

- getTime() - Retorna data e hora no formato yyyymmddhhmmss.
- addHistory(topico,texto) - Adiciona uma notificação ao histórico.
- history(topico,dataInicio,dataFim) - Solicita notificações antigas.

Optamos pela integração híbrida para este módulo por duas razões. Por um lado, devemos notar que as novas funcionalidades têm influência muito pequena na interface disponibilizada pela camada publish-subscribe.

Desta forma, seria interessante que a camada de aplicação acessasse a camada publish-subscribe diretamente. Por outro lado, queremos garantir que toda publicação seja precedida pelo armazenamento. Desta forma, preferimos evitar que a camada de aplicação importe os dois módulos e seja responsável pela escolha do módulo na invocação dos métodos. A integração híbrida garante que a camada de aplicação acesse apenas a camada de serviço disponibilizada e evita a redefinição dos métodos não alterados, redirecionando as invocações.

Uma vez definida a forma de integração da camada de serviços, precisamos determinar como o armazenamento das publicações foi feito. Para tomarmos esta decisão, levamos em conta a arquitetura da rede publish-subscribe já apresentada, a transitoriedade dos nós, o processo de roteamento e as funcionalidades disponibilizadas pela camada publish-subscribe.

As histórias devem estar armazenadas nos nós responsáveis por gerenciar os respectivos tópicos. Precisamos garantir, portanto, que no caso de alteração do nó responsável por um tópico, as histórias sejam também remanejadas. Embora possamos implementar este procedimento na camada de serviços, a camada publish-subscribe já o fez e, na sua interface, define métodos que visam permitir que a camada de serviço possa utilizá-los sem violar a integridade das camadas. Embora as funções já tenham sido apresentadas, esta seção ilustra melhor o momento onde as mesmas devem ser utilizadas.

A função *setTopicField* recebe o nome de um tópico, o nome de um campo e uma tabela com valores e associa a tabela ao tópico, utilizando o nome do campo como índice. No módulo de serviços que estamos implementando, iremos definir um campo chamado *historia*. Desta forma, a camada publish-subscribe assumirá que o campo *historia* é uma das informações associadas ao tópico e será responsável por garantir que, no caso de entradas ou saídas de nós da rede, os valores continuem consistentes e disponíveis.

A função *getTopicField* nos permite acessar e obter os valores de um dos campos que tenhamos associado a um tópico. Esta função recebe como parâmetro o nome do tópico e o nome do campo e retorna os valores antes armazenados.

Uma vez que, no caso de entradas e saídas de nós da rede, o nó responsável pelo armazenamento das histórias pode mudar, precisamos considerar os momentos transitórios de inconsistência das informações ao invocarmos uma função. A camada publish-subscribe enfrenta este mesmo problema ao invocar funções que atuam sobre os tópicos armazenados. Para resolver este problema, a camada publish-subscribe utiliza a função *handleCommand* para gerenciar a execução dos comandos. Desta forma, a camada publish-subscribe pode alternar entre a execução imediata de um comando ou a sua buferização

para posterior execução. Uma vez que estamos utilizando a camada publish-subscribe para gerenciar o armazenamento das histórias, é natural que utilizemos também a função de gerência de comandos utilizada pela camada publish-subscribe. A interface da camada publish-subscribe considera esta necessidade e disponibiliza a função de gerência.

Não precisaremos, portanto, nos preocupar em definir estruturas para armazenar as histórias na camada de serviços ou com a garantia de consistência das informações em todos os momentos. O processo de armazenamento e controle da execução dos métodos, entretanto, poderia ser totalmente realizado na camada de serviço. Esta camada poderia definir novamente as funções da camada publish-subscribe ou utilizar alguma forma de controle totalmente nova. As funções disponibilizadas na interface da camada publish-subscribe buscam apenas facilitar o trabalho de implementação de módulos de serviços adicionais.

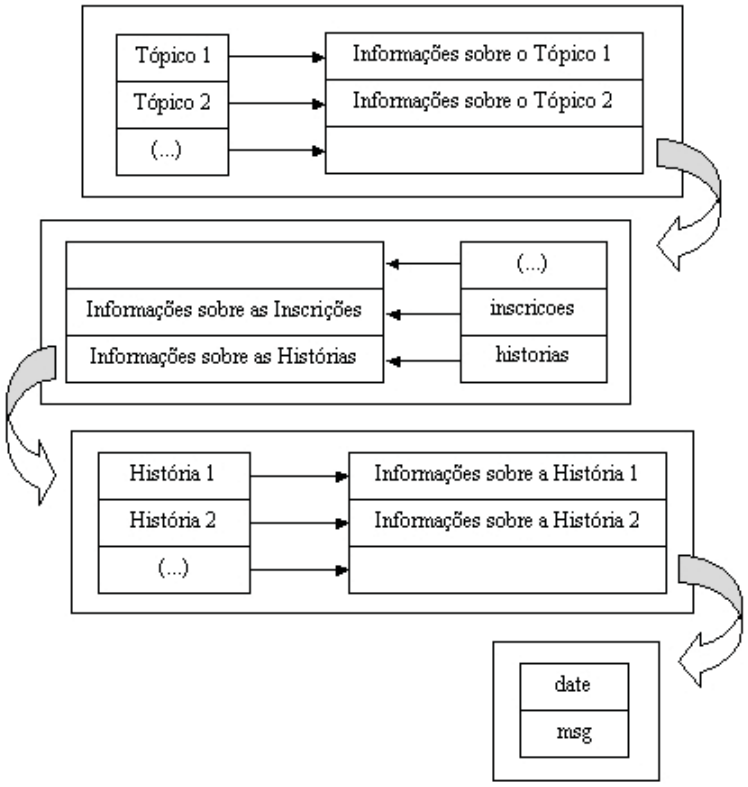

Figura 5.4: Estrutura de Armazenamento das Histórias

Embora tenhamos definido a forma de armazenamento das histórias, ainda não descrevemos a estrutura que utilizamos para representar uma história. Considerando as necessidades de acesso a notificações antigas, a estrutura da história deve conter campo um campo que especifique o horário

de sua publicação. Além do horário, a estrutura precisa armazenar o texto publicado. A figura 5.4 ilustra as estruturas e a associação destas estruturas com os tópicos, gerenciados pela camada publish-subscribe.

No momento da publicação de uma mensagem, portanto, o texto da mensagem e a data e hora local do sistema são utilizados para compor um registro que será armazenado na tabela de histórias. Desta forma, quando desejado, um cliente pode solicitar notificações antigas para um dado tópico que estejam dentro de um intervalo de tempo estabelecido.

5.3 Gateway Publish-Subscribe

Um gateway publish-subscribe pode ter diversas utilidades para as aplicações. Essencialmente, um gateway publish-subscribe é um intermediário entre os clientes e a rede publish-subscribe, executando algum tipo de tratamento nas mensagens trocadas.

Quando uma aplicação admite a utilização do módulo desenvolvido, o gateway faz papel de cliente na rede publish-subscribe e a comunicação com os clientes internos pode acontecer apenas na camada de aplicação. Como a principal função do gateway é servir como interface para diversos clientes na rede publish-subscribe, a aplicação sendo executada no gateway tem que permitir a conexão de diferentes clientes e transmitir suas requisições para a rede publish-subscribe. Para garantir que o retorno das solicitações ou a publicação de notificações seja direcionado para o cliente correto, o gateway deve relacionar as solicitações com os clientes. Desta forma, a rede publish-subscribe entende que existe apenas um nó realizando solicitações e cada um dos clientes entende que está se conectando diretamente a rede publish-subscribe.

Um exemplo simples de utilização de um gateway se baseia no conceito de gateway das redes de comunicação. Poderíamos considerar que as máquinas de uma rede interna de uma empresa não possuem acesso à rede publish-subscribe. Somente o gateway teria acesso à rede e, sendo assim, teria a responsabilidade de encaminhar as mensagens entre os clientes e a rede publish-subscribe. Neste caso, o único tratamento sendo executado pelo gateway seria a tradução dos endereços de rede.

Um gateway um pouco mais elaborado poderia ter a responsabilidade de filtrar o conteúdo das mensagens ou limitar o acesso à rede publish-subscribe

a certos momentos do dia. Neste caso, os clientes internos só poderiam receber ou enviar mensagens de acordo com as normas estabelecidas.

Uma questão bastante abordada em sistemas publish-subscribe é a utilização de clientes móveis. Como clientes móveis podem possuir recursos limitados e conectividade instável, a utilização de um gateway pode ser bastante útil. A utilização de um gateway poderia permitir, por exemplo, que o código executado nos clientes fosse mais simples. A instabilidade da conexão também poderia ser remediada, com o gateway garantindo a entrega das mensagens.

Na próxima seção do capítulo iremos descrever o módulo *luapsg*, que implementa um gateway para clientes móveis com conexões persistentes. Iremos descrever a arquitetura e explicar como a função de publicação poderá ser utilizada para facilitar a implementação da solução. Na seção seguinte iremos discutir a utilização de gateways para realizar correlação entre as notificações publicadas.

5.3.1 Gateway para Clientes Móveis - Conexões Persistentes

O módulo *luapsg* implementa um gateway publish-subscribe para clientes móveis com conexões persistentes. A primeira vantagem que o módulo oferece aos clientes móveis é a possibilidade de execução de um código simplificado, desejável para clientes com limitação de recursos. Para possibilitar a utilização de código simplificado, o módulo executado no gateway disponibiliza todas as funcionalidades necessárias aos clientes móveis. A existência de conexões persistentes, a segunda contribuição que o módulo disponibiliza para os clientes móveis, garante que as mensagens enviadas para os clientes não sejam perdidas, mesmo em momentos de desconexão. Desta forma, o módulo garante a consistência das informações nos dispositivos com instabilidade na conexão.

Dentre as funções disponibilizadas pelo módulo de gateway, podemos perceber todas as funções que a camada publish-subscribe fornece. Estas funções, entretanto, irão considerar a existência do gateway. Podemos perceber, também, que o módulo disponibiliza uma função para o envio de mensagens, redefinindo a função definida na camada ALua. A necessidade de redefinição desta função está associada ao controle no envio de mensagens e será mais bem explicada no capítulo referente à camada de serviços.

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

- createTopic(topico) - Solicita a criação de um novo tópico.
- subscribe(topico,cliente,funcao) - Solicita a inscrição em um tópico.
- unSubscribe(topico,cliente) - Solicita a exclusão da inscrição.
- publish(topico,texto) - Solicita a publicação de uma notificação.
- send(id,msg) - Envia uma mensagem para um processo.

O gateway intermedia a comunicação entre os clientes móveis e a rede publish-subscribe. Uma vez que a única função do gateway será esta, o gateway não precisa executar a camada de aplicação. Como o gateway será um cliente na rede publish-subscribe, o gateway precisará executar todas as camadas inferiores. Os clientes móveis não serão clientes da rede publish-subscribe, comunicando-se apenas com o gateway. Desta forma, a única camada que deve ser executada nos clientes é a camada de aplicação.

A camada de aplicação dos clientes móveis deve disponibilizar funções que encaminhem as solicitações para o gateway. O gateway disponibiliza todas as funções publish-subscribe desenvolvidas, sendo o acesso às funcionalidades feito pela camada de serviços. A cada solicitação recebida dos clientes móveis, o gateway deve verificar se já possui as informações necessárias para realizar o solicitado ou se deve encaminhar a requisição para a rede publish-subscribe. Mesmo quando o encaminhamento para a rede é necessário, entretanto, as funções da camada publish-subscribe são executadas pelo módulo de serviço. Considerando estas características, a forma de integração mais adequada para este módulo será a integração clássica.

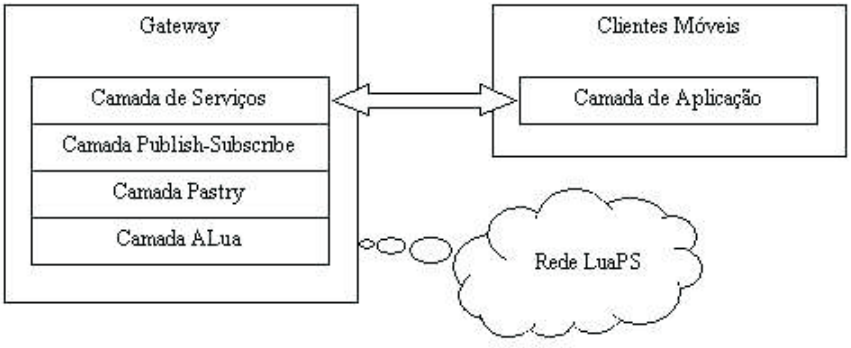

Figura 5.5: Arquitetura de Comunicação com Utilização do Gateway

A figura 5.5 ilustra a arquitetura de comunicação de uma sistema utilizando o gateway. O gateway executa as camadas inferiores do sistema, mas não executa a camada de aplicação. Os clientes móveis, por outro lado, executam apenas a camada de aplicação. Os clientes móveis somente se comunicam com o gateway e o gateway atua como um cliente da rede publish-subscribe.

Uma vez definida a arquitetura da camada de serviços, passaremos a analisar com mais detalhes a implementação do módulo de gateway. O módulo de gateway define uma estrutura para o armazenamento das informações sobre os tópicos nos quais os clientes móveis tenham feito inscrições. Quando uma nova inscrição é solicitada por um dos clientes, esta informação é adicionada à tabela mantida pelo gateway e é feita uma verificação por outros clientes inscritos neste tópico. Caso a inscrição seja a primeira inscrição no tópico, o gateway deve solicitar à rede publish-subscribe a sua inscrição no tópico. No momento da inscrição do gateway em um tópico, o gateway define como função de publicação uma função interna do gateway que irá encaminhar a notificação para todos os clientes móveis inscritos no tópico.

A opção pela utilização das conexões persistentes estará a cargo da aplicação sendo executada nos clientes. Caso a função de publicação utilizada na aplicação contenha algum comando que necessite do envio das informações para o cliente móvel, a utilização das conexões persistentes é recomendada. Neste caso, a função de publicação deve utilizar a função de envio de mensagens disponibilizada pela camada de serviços. Esta função de envio possibilita o reenvio de mensagens que não cheguem ao destino. O número de tentativas de envio das mensagens é definido pelo parâmetro *retryTries* e cada tentativa acontece após um intervalo de tempo definido pelo parâmetro *retryTime*.

Para possibilitar o reenvio das mensagens, o módulo de gateway define uma estrutura que armazena as mensagens que não atingiram o destino e controla o número de tentativas de envio. A estrutura, chamada de *tbMensagens*, é indexada pelo temporizador utilizado para o reenvio das mensagens. Ou seja, quando a função de envio determina que uma mensagem não atingiu o destino, a função de reenvio é programada para ser executada após o intervalo de tempo especificado. O temporizador responsável por esta programação indexa a tabela de mensagens.

A tabela de mensagens tem três campos. O primeiro campo identifica o endereço do cliente que deve receber a mensagem. O segundo campo identifica o número de tentativas de reenvio da mensagem, iniciado com zero. O terceiro campo consiste na própria mensagem. A figura 5.6 ilustra a estrutura especificada.

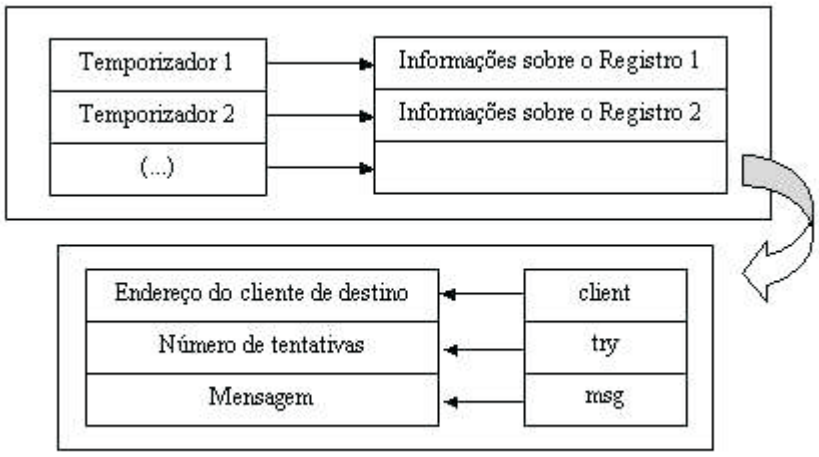

Figura 5.6: Tabela de Mensagens Armazenadas para Reenvio

A cada tentativa de reenvio, o temporizador é utilizado para indexar a tabela de mensagens e possibilitar a consulta dos campos que identificam o cliente e a mensagem a ser entregue. Caso a mensagem seja entregue com sucesso, o temporizador é excluído e o registro é retirado da tabela de mensagens. Caso a mensagem ainda não possa ser entregue, o número de tentativas é consultado. Caso o limite de tentativas tenha sido atingido, o sistema desiste da entrega da mensagem, indica o erro na entrega da mensagem e exclui o temporizador e o registro. Caso contrário, o número e tentativas é incrementado e a tentativa de reenvio será executada novamente após o intervalo de tempo definido.

Para finalizarmos a descrição do módulo, iremos considerar uma questão referente a tolerância a falhas. Ao descrevermos a camada publish-subscribe, explicamos que esta camada disponibiliza funções que permitem associar campos à estrutura de gerência dos tópicos e, desta forma, repassar à camada publish-subscribe a tarefa de garantir a consistência das informações das camadas superiores em caso de falha. No módulo de serviço desenvolvido, as informações foram armazenadas no próprio gateway e, desta forma, no caso de falha do gateway, as informações serão perdidas. Esta decisão não era obrigatória, mas garantia a minimização do tráfego na rede e a simplificação do desenvolvimento do módulo. Como podemos considerar que um gateway para clientes móveis precisa estar ativo para garantir a comunicação, assumimos que o gateway será executado em uma máquina que não estará suscetível a falhas freqüentes.

5.3.2 Gateway para Correlação

Em aplicações publish-subscribe complexas, muitos eventos acontecem, sendo que os eventos individualmente podem não ter grande significado para a aplicação. A correlação dos eventos entre si ou dos eventos com o tempo, entretanto, pode ter grande influência na aplicação.

Para exemplificarmos a utilização de um gateway para correlação das notificações, vamos considerar uma aplicação publish-subscribe que notifica alterações nos preços das ações da bolsa de valores. Podemos considerar, por exemplo, uma ação cujo preço esteja oscilando com grande freqüência. Desta forma, quedas e aumentos do preço são publicados várias vezes em um pequeno intervalo de tempo. Ao fim do intervalo, entretanto, o preço da ação se encontra igual ao preço inicial. Para algumas aplicações, toda variação do preço pode precisar ser publicada. Para outras aplicações, entretanto, apenas uma alteração do preço que se mantenha por um intervalo de tempo poderá ser significativa. Poderíamos considerar, ainda, que a variação do preço desta ação somente seria significativa quando associada à variação do preço de um grupo de outras ações.

Sem a utilização do gateway para correlação, cada um dos clientes deveria se inscrever nos tópicos referentes a todas as ações e receber todas as notificações. Somente desta forma a camada de aplicação teria informações suficientes para notificar o cliente sobre as informações desejadas. Com a utilização do gateway, por outro lado, apenas o gateway precisaria se inscrever nos tópicos e receber as notificações. Caberia ao gateway, então, correlacionar as notificações e enviar as informações necessárias para os clientes. A utilização do gateway centraliza a correlação das notificações e as inscrições nos tópicos, diminuindo da troca de mensagens na rede.

A utilização do gateway e a diminuição da troca de mensagens é particularmente importante se considerarmos clientes móveis. O gateway permite, ainda, que o armazenamento e o processamento da correlação seja feito em uma máquina com maior disponibilidade de recursos, cabendo aos dispositivos móveis tarefas simplificadas.

Para que o gateway possa realizar a correlação, o mesmo deve estar preparado para receber as solicitações dos clientes, analisar a solicitação e identificar as inscrições necessárias e armazenas as informações que possibilitem a execução da correlação. Será preciso, portanto, a definição de uma linguagem que possa ser utilizada pelos clientes e pelo gateway para identificar os eventos correlacionados.

A implementação de um gateway para realizar correlação se distancia um pouco dos objetivos do projeto, mas a análise do problema permite a percepção da flexibilidade do sistema LuaPS. Podemos encontrar uma discussão mais detalhada sobre correlação, diminuição na troca de mensagens e definição de uma linguagem para correlação em (Li & Jacobsen, 2005).

5.4 Tolerância a falhas

Ao considerarmos a arquitetura utilizada na rede publish-subscribe, podemos perceber a importância de existir tolerância a falhas. Uma vez que os clientes da rede são responsáveis por armazenar os tópicos criados e as inscrições realizadas, a falha de um dos clientes pode resultar em perda de informações. O módulo *luapsb* desenvolvido busca possibilitar a re-estruturação da rede publish-subscribe em caso de falhas, realizando o backup das informações nos nós com identificadores vizinhos.

O módulo desenvolvido trata a questão com certa simplicidade, de modo que a tolerância a falhas não é garantida em todas as circunstâncias. Apesar disto, os objetivos da camada de serviços são alcançados e o módulo ilustra uma das formas de estender o paradigma publish-subscribe e incluir tolerância a falhas em uma aplicação. Para uma aplicação onde tolerância a falhas seja um ponto fundamental, o módulo poderia ser incrementado. Poderíamos, por outro lado, desenvolver um módulo de tolerância a falhas totalmente diferente. O gateway publish-subscribe, por exemplo, poderia ser utilizado para realizar o backup das informações e verificar as falhas dos nós.

Em um processo de backup genérico, podemos considerar que a replicação das informações pode acontecer de forma incremental ou completa. No backup completo, os dados são totalmente replicados quando necessário, enquanto que, no backup incremental, somente os dados alterados são adicionados ao backup. A primeira forma simplifica o processo, uma vez que não é necessário comparar os dados armazenados com os novos dados. Por outro lado, esta forma implica no maior trânsito de dados. No módulo que desenvolvemos, utilizamos a forma incremental. Para determinar os dados que devem ser adicionados ao backup, iremos alterar as funções publish-subscribe para que, a cada alteração nas informações, o backup também seja atualizado.

Assim como nos outros módulos de serviço desenvolvidos, as novas funcionalidades implementadas estendem as funcionalidades existentes na camada publish-subscribe. Desta forma, as funções definidas na camada publish-

subscribe continuam sendo executadas, embora possam ter sofrido alterações. Segue uma lista com as funções alteradas e uma breve descrição:

- joinPS(no,callback) - Entra em uma rede publish-subscribe.
- leavePS(aplicacao) - Sai de uma rede publish-subscribe.
- createTopic(topico) - Solicita a criação de um novo tópico.
- subscribe(topico,cliente,funcao) - Solicita a inscrição em um tópico.
- unSubscribe(topico,cliente) - Solicita exclusão de inscrição em um tópico.
- publish(topico,texto) - Solicita a publicação de uma notificação.

Este módulo, entretanto, estende as funções publish-subscribe de uma forma diferente dos outros módulos descritos. Para entendermos as diferenças de implementação, precisamos nos remeter ao capítulo que define a camada publish-subscribe e relembrar a natureza distribuída das funções, sendo compostas por funções executadas no âmbito local e por funções executadas em nós remotos. Nos módulos anteriores, alterarmos apenas as funções locais, isto é, as funções que fazem parte da interface da camada. No módulo de backup, entretanto, optamos por alterar as funções remotas. Embora esta forma de extensão não fosse obrigatória, simplifica o processo e exemplifica uma outra forma de estender o sistema. A utilização desta forma de extensão, entretanto, associa o módulo de serviço desenvolvido a este módulo publish-subscribe particular, uma vez que as funções alteradas não são funções da interface.

As funções locais, por serem funções da interface, são chamadas pelas camadas superiores do sistema. Quando uma aplicação carrega um módulo de serviço, as funções locais redefinidas passam a ser chamadas. As funções remotas, entretanto, são chamadas pelas funções locais já definidas e, desta forma, não são afetadas pelo carregamento do módulo de serviço. Para substituirmos as chamadas às funções remotas poderíamos, por exemplo, redefinir também as funções locais e alterar as requisições. A linguagem Lua, entretanto, nos permite simplificar este processo. Uma vez que as funções Lua são variáveis de primeira classe, iremos alterar a definição das funções remotas, evitando precisar alterar as chamadas e as funções locais.

Assim que o módulo de backup é carregado, declaramos variáveis que referenciam as funções remotas definidas na camada publish-subscribe do nó. Em seguida, implementamos as novas funções remotas utilizando, inclusive, as funções referenciadas. Após a implementação das novas funções, alteramos as referências que a camada publish-subscribe do nó tinha para as funções remotas, fazendo com que a camada referencie as novas implementações.

Desta forma, as chamadas não precisam ser alteradas para que as novas implementações sejam executadas. De uma forma geral, as novas funções irão executar as funções previamente definidas e, em seguida, realizar o backup das informações necessárias.

Para podermos entender o processo de backup das informações, devemos considerar o processo de entrada e saída da rede descrito no capítulo referente à camada publish-subscribe. Quando um nó deixa a rede publish-subscribe, os tópicos por ele gerenciados são distribuídos entre os processos do seu conjunto de folhas. De forma análoga, no processo de backup, os tópicos gerenciados por um nó são replicados entre as folhas. Sendo assim, uma vez que um nó falhe, os nós que devem passar a gerenciar os tópicos já terão as informações necessárias para manter a consistência da rede. Quando um nó deixa a rede publish-subscribe, portanto, o mesmo não mais precisa transferir as informações armazenadas. Assim, a falha de um nó ou a sua saída da rede podem ser tratadas da mesma forma.

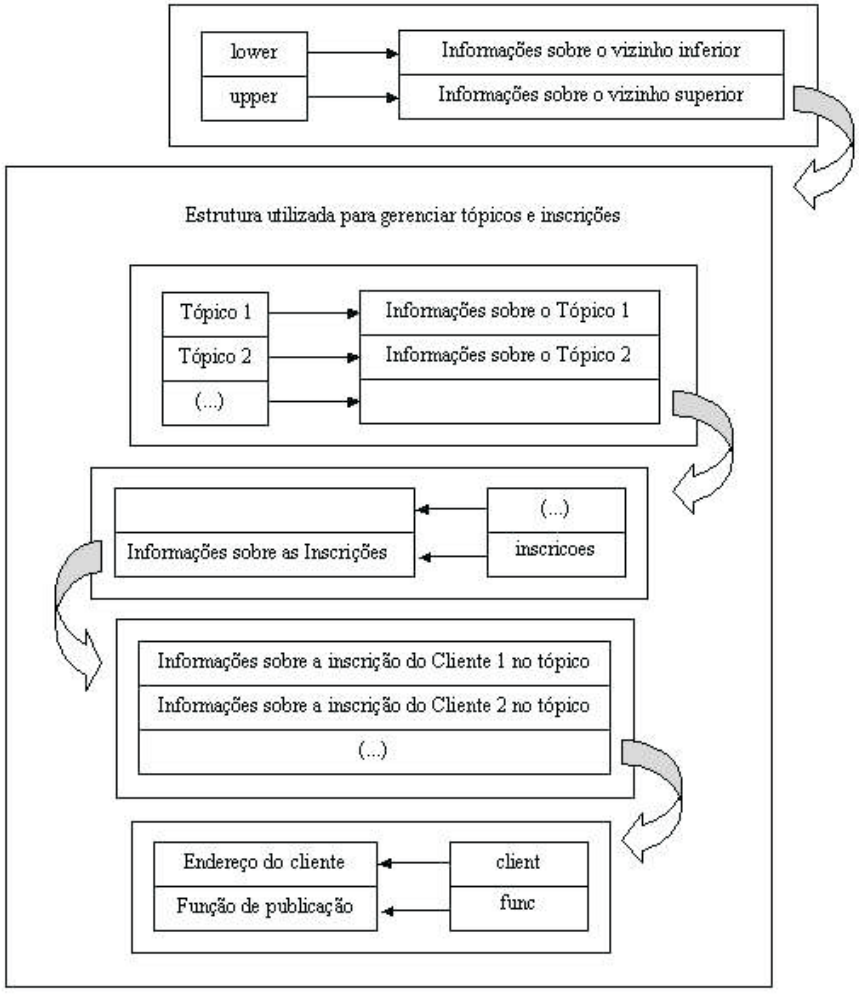

Figura 5.7: Estrutura de Backup

Para armazenar as informações replicadas, o módulo de serviço de tolerância a falhas define uma estrutura denominada *tbBackup*, conforme definido na figura 5.7. A estrutura de backup indexa duas outras tabelas: a tabela *lower* armazena as informações do nó com identificador inferior e a tabela *upper* armazena as informações do nó com identificador superior. Cada uma das tabelas indexadas referencia uma estrutura idêntica à estrutura que gerencia os tópicos e inscrições, conforme definido no capítulo que apresenta a camada publish-subscribe.

Uma vez definido o processo de replicação das informações, passemos a analisar o processo de restauração das informações armazenadas no momento em que a falha de um cliente é identificada. O módulo de backup implementado utiliza as características do roteamento para identificar a falha de um nó. Um cliente irá receber a solicitação de execução de um comando referente a um tópico presente em sua tabela de backup se, e somente se, o vizinho responsável pelo tópico tiver falhado.

Poderíamos utilizar diversas estratégias para determinarmos que aconteceu uma falha em um cliente. Poderíamos, por exemplo, configurar os clientes para, com uma dada freqüência, consultar os nós em seu conjunto de folhas e identificar possíveis falhas. A estratégia escolhida elimina a necessidade de mensagens de consulta, diminuindo o tráfego na rede. Por outro lado, a estratégia admite que possa ocorrer perda de informações em função das operações executadas entre a percepção da falha de dois nós com identificadores vizinhos.

No capítulo referente à camada publish-subscribe, definimos a função para gerência da execução de comandos, que recebe um comando e um tópico como parâmetro. O módulo de backup redefine esta função, acrescentando a verificação pela necessidade de restauração do backup. Para isto, a nova função verifica a existência do tópico nas tabelas indexadas pela tabela de backup. Uma vez que a falha seja identificada, inicia-se o processo de restauração. Para explicar a restauração do backup, iremos considerar a situação ilustrada na figura 5.8, que mostra os clientes e os respectivos conjuntos de folhas. No estágio 1, nenhum dos nós falhou e a rede ainda está consistente.

Na situação descrita no exemplo, o cliente C falha e o cliente B recebe uma solicitação de execução que identifica a falha. Caso o cliente D tivesse percebido a falha, o processo seria análogo e não será descrito neste texto. Assim que o processo B identifica a falha do nó C, o mesmo restaura as informações armazenadas na tabela de backup do nó superior, transferindo estas informações para sua tabela de tópicos. Uma vez que sua tabela de tópicos foi alterada, o mesmo deve solicitar a atualização do backup nos nós de sua

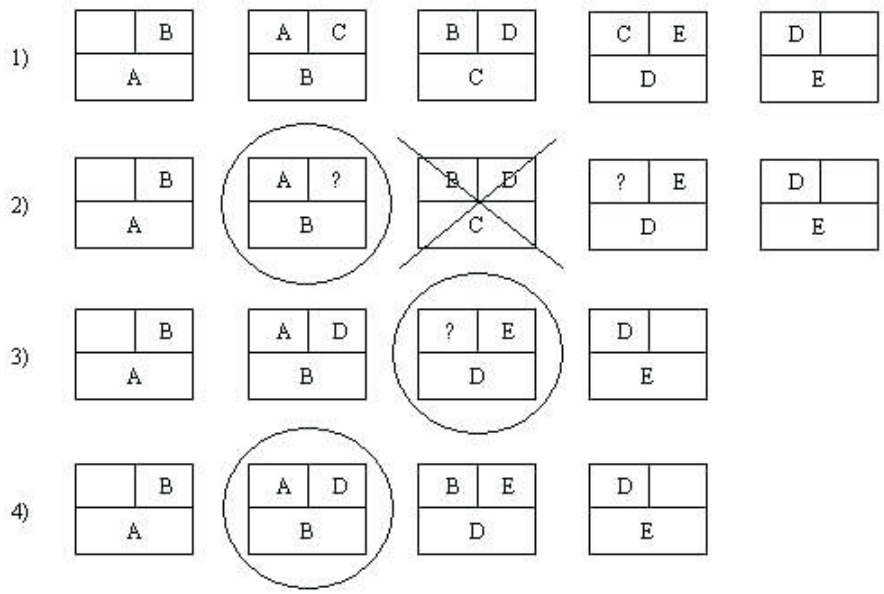

Figura 5.8: Processo de Restauração do Backup

tabela de folhas, utilizando a função *sendbackup* ao longo do processo descrito a seguir. Neste momento, o backup não será incremental. A função identifica quais tópicos devem ser enviados para cada vizinho e todas as informações do tópico são enviadas. O processo iniciado quando B detecta a falha de C deverá também atualizar as informações do nó D, o outro cliente afetado pela falha de C.

Para que o processo B identifique a falha em de seu nó superior, o mesmo tem que ter recebido uma solicitação roteada pela rede. Caso a solicitação tenha chegado a partir dos nós de sua esquerda, nós com identificadores menores, a entrada que referencia o cliente superior em sua tabela de folhas, indeterminada na figura, já estará ajustada. Isto acontece porque, ao rotear a mensagem, o sistema tenta enviar a solicitação para C utilizando esta folha. Como o cliente C falhou, a entrada é corrigida, passa a apontar para D e, em função dos identificadores dos nós, a mensagem é roteada para B. Caso a solicitação tenha chegado a partir dos nós de sua direita, nós com identificadores maiores, a entrada que referencia o cliente superior em sua tabela de folhas ainda estará referenciando o nó C. Por outro lado, a entrada que aponta para o nó inferior na tabela de folhas do nó D já estará ajustada, referenciando o nó B. Como não temos como determinar por onde a solicitação chegou, não podemos saber qual das entradas da tabela de folhas já está correta e ambas precisarão ser corrigidas. O estágio 2 representa esta situação, indicando o nó C que falhou, o nó que tem a atividade no processo com um círculo e as posições cujos valores são incertos com as interrogações.

O processo de restauração se inicia com a correção da entrada da tabela de folhas do nó B. Para isto, o nó roteia uma mensagem M1 com destino ao valor registrado na tabela. Caso a entrada já esteja correta, a mensagem chega ao nó D. Caso a entrada ainda esteja errada, o roteamento da mensagem corrige a entrada, que passa a referenciar o nó D. Neste caso, entretanto, não sabemos qual nó será o destino da mensagem. A mensagem M1 foi roteada para C e, em função dos valores B, C e D, poderá chegar a B ou a D. Para garantir o controle do processo, a mensagem M1 roteada será uma solicitação de envio da mensagem M2 para D. Desta forma, ou B irá enviar uma mensagem para D ou D mandará uma mensagem pra ele mesmo. Independente do roteamento seguido, o estágio 3 representa o estado da rede após a transmissão das mensagens M1 e M2: a tabela de folhas do nó B foi corrigida e o nó D tem a atividade no processo.

A mensagem M2, que obrigatoriamente será executada em D, transfere as informações da tabela de backup para a tabela de tópicos. Além disso, a mensagem precisa corrigir a entrada de sua tabela de folhas. Para isto, a mensagem M2 terá uma solicitação de roteamento da mensagem M3 com destino ao valor registrado na tabela. Assim como no caso anterior, a entrada será corrigida e M3 chegará a B ou a D. O Conteúdo de M3 será uma solicitação de envio da mensagem M4 para B. Desta forma, M4 será executado em B obrigatoriamente. Independente do roteamento seguido, o estágio 4 representa o estado da rede após a transmissão das mensagens M3 e M4: a tabela de folhas do nó D foi corrigida e o nó B tem a atividade no processo.

No momento da execução de M4, e somente neste momento, podemos garantir que as entradas da tabela de folhas de B e D foram corrigidas e que os dados da tabela de backup de B e D já foram restaurados. Neste momento, portanto, B pode executar a função *sendbackup* para realizar o backup de suas informações em A e D. Em seguida, B envia uma mensagem M5 para D, requisitando que D execute a função *sendbackup* e envie o backup de suas informações para B e E.

5.5 Combinando Módulos de Serviço

Uma vez que a camada de serviços é composta por diversos módulos independentes, um sistema publish-subscribe pode desejar utilizar mais de um módulo combinado na mesma aplicação. Para definirmos uma forma de combinar os diferentes serviços, vamos analisar a arquitetura dos módulos.

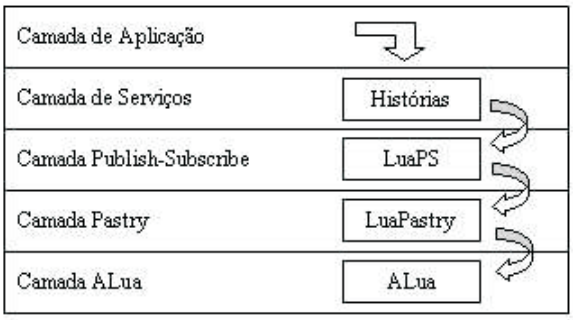

Figura 5.9: Aplicação com um Módulo de Serviço

Cada módulo importado por uma aplicação é definido em tempo de execução como uma tabela Lua cujos campos representam as variáveis e funções. Desta forma, as diferentes camadas do sistema LuaPS são implementadas como tabelas que possuem uma referência para a tabela que implementa a camada inferior. A figura 5.9 ilustra a estrutura de uma aplicação que tenha carregado apenas o módulo de histórias na camada de serviços. O módulo de serviços, que precisa das funcionalidades disponibilizadas pela camada publish-subscribe, possui uma referência para a tabela que define o módulo LuaPS.

Consideremos, agora, uma outra aplicação que precisa carregar o módulo de histórias e o módulo de tolerância a falhas na camada de serviços. Cada um dos módulos, que precisa das funcionalidades providas pela camada publish-subscribe para funcionar corretamente, estende o sistema de forma diferente. O módulo de tolerância a falhas altera a execução de todas as funções publish-subscribe, realizando o backup e verificando a necessidade de restauração das informações. O módulo de histórias altera apenas a função de publicação, armazenado as notificações antes de realizar a publicação propriamente dita.

Uma primeira forma de abordar esta situação seria o desenvolvimento de um outro módulo de serviços que utilizasse as funcionalidades dos módulos já definidos e disponibilizasse as funcionalidades combinadas para a camada de aplicação. O novo módulo deveria importar os dois módulos de serviços já existentes e tratar a incompatibilidade entre as funções de publicação. Neste caso, a nova função de publicação seria definida como uma chamada à função de publicação do módulo de histórias seguida por uma chamada à função de publicação do módulo de tolerância a falhas. Todas as outras funções seriam definidas como chamadas às funções do módulo de tolerância a falhas. A camada de aplicação, então, importaria apenas o novo módulo de serviços definido. Existem outros exemplos, entretanto, onde tratamentos mais elaborados poderiam ser necessários para eliminar a incompatibilidade entre funções de módulos combinados.

Para o exemplo discutido, entretanto, a arquitetura definida e as características da linguagem Lua nos permitem abordar a combinação dos módulos de serviço com um pouco mais de simplicidade, sem a necessidade do desenvolvimento de um módulo de serviço específico. Para isto, bastaria alterarmos a referência para o módulo LuaPS que existe no módulo de histórias, fazendo com que o módulo passasse a referenciar o módulo de tolerância a falhas.

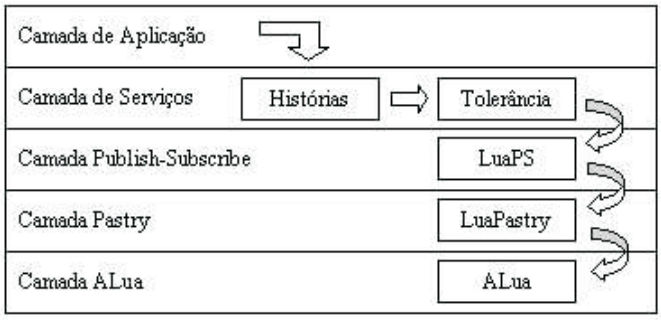

Figura 5.10: Aplicação com dois Módulos de Serviço Combinados

A figura 5.10 ilustra a estrutura da aplicação considerando a combinação dos serviços. O módulo de histórias utiliza o módulo de tolerância a falhas para prover as funcionalidades publish-subscribe de que necessita e a aplicação utiliza o módulo de histórias para acessar as funcionalidades.

Após analisarmos as possibilidades para combinação dos serviços, podemos perceber que, independentemente da solução utilizada, o desenvolvedor pode precisar de algumas informações sobre a implementação dos módulos sendo combinados. Tais informações podem ser obtidas diretamente no código fonte ou através de documentação disponibilizada junto com os módulos.

5.6 Arquitetutra e Extensibilidade

Ao terminarmos o desenvolvimento dos módulos de serviço, que disponibilizam novas funcionalidades à camada publish-subscribe, podemos analisar mais criteriosamente a extensibilidade oferecida pela arquitetura utilizada. Buscamos implementar módulos com enfoques diferentes, tanto a nível de serviço com a nível de implementação. Desta forma, exploramos as diferentes formas de estender o sistema básico.

O módulo de histórias adiciona funcionalidades para requisitar as histórias e altera as funções da interface, possibilitando o armazenamento das

informações. A principal contribuição deste módulo é a análise das funções que permitem associar novos campos à estrutura de gerência de tópicos. Desta forma, pudemos associar informações aos tópicos sem ter o conhecimento dos procedimentos iniciados com a entrada ou saída dos nós da rede, facilitando o desenvolvimento.

O módulo de gateway nos permite analisar o grau de extensibilidade disponibilizado pela arquitetura. O módulo desenvolvido é bastante flexível e nos permite o desenvolvimento de gateways para diversas aplicações. Podemos, inclusive, alterar a arquitetura da rede publish-subscribe, passando a utilizar clientes que não se conectam diretamente à rede.

O módulo de tolerância a falhas nos permite analisar uma outra forma de estender o sistema, utilizando a flexibilidade da linguagem Lua para alterar as funções remotas dos nós. Embora esta forma de extensão possa ser bastante útil, devemos lembrar que, ao utilizarmos funções que não pertencem à interface, estamos associando o módulo de serviço ao módulo publish-subscribe.

Após o desenvolvimento dos módulos, pudemos perceber as facilidades de implementação introduzidas pela arquitetura utilizada. Este capítulo, portanto, além de definir a camada de serviços, nos permitiu validar os princípios de extensibilidade propostos.