Exame de MAC5853 – Desenvolvimento de Sistemas de Computação

aluno: Marcelo da Silva Reis

banca: Paulo J.S. Silva, Flávio S.C. Silva e Alfredo Goldman

Instituto de Matemática e Estatística
Universidade de São Paulo

Primeiro semestre de 2009

Rotas (L) – Um serviço simples de controle de rotas

Fase 1 – Projeto

Cenários de comportamento dos componentes do sistema

As descrições apresentadas neste documento seguem as arquiteturas apresentadas nos diagramas *ModeloObjetosRotas*, *ModeloObjetosCliente*, *ModeloObjetosCET* (modelos de objetos em notação UML), assim como os modelos ER e físico do banco de dados dos componentes *Rota* e *CET* (diagramas *MER-Rotas*, *MER-CET*, *ModeloFisicoRotas* e *ModeloFisicoCET*).

Além disso, supõe-se que todos os componentes foram inicializados, seguindo as descrições do documento *Inicializacao.pdf*.

1 Cenários do componente *Cliente*

1.1 Motorista solicita uma rota

1. A interfaceCliente exibe a localização atual (rua, número e bairro) e pergunta qual é o destino do motorista
2. Motorista digita um nome de rua:
   (a) caso seja um nome inválido, a interface passa essa informação e o sistema volta para o ítem (1).
   (b) do contrário, interface solicita o nome do bairro.
3. Motorista digita um nome de bairro:
   (a) caso seja um nome de bairro inválido, a interface passa essa informação e volta novamente para o ítem (1).
   (b) caso o bairro exista, porém a rua não faz parte dele, a interface passa essa informação e volta novamente para o ítem (1).
   (c) caso contrário, o sistema solicita o número da rua.
4. Motorista digita um número de rua:
   (a) caso seja um número inválido para a rua-bairro, a interface passa essa informação e volta novamente para o ítem (1).
   (b) caso contrário, o sistema informa o endereço origem e o de destino.
5. Sistema pede para motorista confirmar a rota, escolher outra ou sair do sistema.

(a) Caso o motorista peça para escolher outra rota, sistema volta para o ítem (1);

(b) Se a escolha for sair do sistema, vai para o ítem (12).

6. Motorista confirma a rota e um objeto de *ProcessadorRotaCliente* é criado; esse objeto por sua vez inicializa um objeto de *BuscadorRota*.

7. buscadorRota tenta conexão com em Rotas.

   (a) Caso obtenha sucesso, uma mensagem XML análoga à abaixo é enviada à *Rotas*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mensagem do Modulo "Cliente" ao "Rotas". -->
<mensagem_cliente>
        <dados_cliente>
                <cpf>11100011100</cpf>
        </dados_cliente>
        <rota>
                <no_origem>1234</no_origem>
                <no_destino>5678</no_destino>
        </rota>
</mensagem_cliente>
```

   (b) caso não tenha obtido sucesso (timeout), uma mensagem de erro é registrada no log e uma nova tentativa é feita (ítem 7).

8. *Rotas* devolve uma mensagem, que pode ser:

   (a) Resposta esperada, como o XML abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mensagem de resposta do Modulo "Rotas" ao "Cliente". -->
<mensagem_resposta_cliente>
        <dados_cliente>
                <cpf>11100011100</cpf>
        </dados_cliente>
        <via>
                <no_origem>5678</no_origem>
                <no_destino>11456</no_destino>
```

```
                    <!-- Taxa de Ocupacao enviada para fins de simulacao -->
                    <taxa_ocupacao>0.27</taxa_ocupacao>
            </via>
    </mensagem_resposta_cliente>
```

(b) Ou uma mensagem de erro, como a abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mensagem de resposta de "Rotas" ao pedido de "Cliente". -->
<erro_leitura_mensagem />
```

9. buscadorRota processa o arquivo obtido.

   (a) se o arquivo foi recebido corretamente e o parsing mostra tratar-se de uma mensagem de erro, retorna-se ao ítem (7).

   (b) se o arquivo foi recebido corretamente, é feito parsing do mesmo, atualizando o atributo *rotaÓtima* em processadorRotasCliente e exibindo a Via recebida para o motorista. buscadorRota desconecta-se de *Rotas*.

   (c) caso tenha ocorrido um erro na transmissão do XML, retorna-se ao ítem (7).

10. processadorRotasCliente então aguarda que leitorGPS comunique a chegada à um entroncamento.

11. leitorGPS comunica a chegada a um entroncamento; o valor de *noDestino* é comparado com o de *noAtual* (atualizado por leitorGPS).

    (a) se forem diferentes, leitorGPS repete o ítem (7), com o *noAtual* no papel de *noOrigem*.

    (b) caso sejam iguais, processadorRotasCliente imprime o percurso completo percorrido e comunica o final da rota; o objeto é então destruído.

12. interfaceCliente pergunta se o motorista deseja fazer uma nova rota; em caso positivo, vai para o ítem (1), do contrário, todos os objetos são destruídos e o sistema é encerrado.

2 Cenários do componente *Rotas*

2.1 Uma rota é solicitada por um *Cliente*

1. servidoraRotas aceita uma solicitação de conexão de um *Cliente*.
2. *Cliente* envia um XML, análoga ao do ítem (7.a) da seção anterior.
3. servidoraRotas processa o XML:
   (a) se o XML não foi recebido corretamente, é enviada uma mensagem XML de erro análoga à do ítem (8.b) da seção anterior.
   (b) se o XML foi recebido corretamente, servidoraRotas verifica se já existe um objeto da classe *ProcessadorRota* instanciado para o valor de *cpf*; se não existir, cria-o.
4. processadorRota recebe os valores de *noOrigem* e *noDestino*, utilizando-os para determinar a rota ótima:
   (a) caso *noOrigem* seja o nó inicial de *últimaVia*, significa que houve um erro na recepção da XML da resposta por parte de *Cliente*; *últimaVia* então é devolvida.
   (b) caso *rotaÓtima* seja igual à NIL ou *horárioÚltimaAtualização* de mapaViárioRotas seja mais novo que *horárioCálculoRota*, o caminho de custo mínimo de *noOrigem* à *noDestino* é determinado, utilizando o algoritmo de Dijkstra. O *horárioÚltimaAtualização* é atualizado, é retirado o último entroncamento da lista obtida (*rotaÓtima*), devolvendo a via composta por (*noOrigem*, *últimoNó*).
   (c) caso *rotaÓtima* seja diferente de NIL, e *horárioÚltimaAtualização* de mapaViárioRotas é mais antigo que *horárioCálculoRota*, é retirado o último entroncamento de *rotaÓtima*, devolvendo a via composta por (*noOrigem*, *últimoNó*).
5. processadorRota cria então um objeto de *ArquivoRotas* para armazenar a via percorrida:
   (a) Caso seja a primeira via de uma nova rota, comandos SQL análogos aos exemplos abaixo são executados:

```
INSERT INTO ConsultaRota
       (ID_Rota, horaConsulta, CPF, noOrigem, noDestino)
       VALUES
       ('id_rota','horaAtual', '00011122200', '123', '456')
INSERT INTO UtilizaVias
       (ID_Rota, horaConsultaVia, 'noInicial', 'noFinal')
       VALUES
       ('id_rota', 'horaAtual', '123', '456')
```

Onde ID_Rota pode ser calculado utilizando-se o "auto-increment" do MySQL. ID_Rota é então devolvido para processadorRota.

(b) Caso já exista uma rota dessa consulta, os comandos seriam parecidos com os seguintes:

```
INSERT INTO UtilizaVias
       (ID_Rota, horaConsultaVia, 'noInicial', 'noFinal')
       VALUES
       ('id_rota', 'horaAtual', '123', '456')
```

6. processadorRotas devolve uma via à servidoraRotas; caso esta seja a última via do trajeto, processadorRotas é destruído.

7. servidoraRotas envia ao componente *Cliente* a resposta, em formato XML análogo ao do ítem (8.a) da seção anterior e encerra a conexão.

2.2 É iniciada uma atualização do mapa viário

1. a cada 30 minutos, escalonadorMapaViário inicializa um objeto de *AtualizadorMapaViário*; nesses casos, a flag *ocupado* é setada para "verdadeiro".

2. atualizadorMapaViário tenta se conectar com *CET*:

   (a) Caso obtenha sucesso, uma mensagem análoga à abaixo é enviada ao componente *CET*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mensagem de atualizacao do Modulo "Rotas" a "CET". -->
<mensagem_atualizacao />
```

(b) Caso não tenha sido obtido sucesso (*timeout*), uma mensagem de erro é registrada num log e uma nova tentativa é feita (ítem 2).

3. *CET* devolve um XML e encerra a conexão; a mensagem pode ser:

   (a) Resposta esperada, como o XML abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mensagem de resposta de "CET" ao "Rotas". -->
<mensagem_resposta_atualizacao>
        <via>
                <no_origem>5678</no_origem>
                <no_destino>11456</no_destino>
                <taxa_ocupacao>0.27</taxa_ocupacao>
        </via>
</mensagem_resposta_atualizacao>
```

   (b) Ou uma mensagem de erro, como a abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mensagem de resposta de "CET" ao pedido de "Rotas". -->
<erro_leitura_mensagem />
```

4. atualizadorMapaViário processa o arquivo obtido.

   (a) se o arquivo foi recebido corretamente e o parsing mostra tratar-se de uma mensagem de erro, retorna-se ao ítem (2).

   (b) se o arquivo foi recebido corretamente, é feito parsing do mesmo, atualizando o atributo *taxaOcupação* nas respectivas vias dos elementos *via*, assim como o *horárioÚltimaAtualização* em mapaViárioRotas.

   (c) caso tenha ocorrido um erro na transmissão do XML, retorna-se ao ítem (2).

5. a flag *ocupado* de escalonadorMapaViário é setada para "falso".

2.3 É iniciada uma limpeza de rotas no Banco de Dados

1. Uma vez por dia, num horário arbitrário, escalonadorLimpaRotas inicializa um objeto de *ArquivoRotas*; nesses casos, a flag *ocupado* é setada para "verdadeiro".

2. É chamado então o método *apagaRotasAntigas*, que executa o seguinte comando SQL:

```
DELETE FROM ConsultaRota
      WHERE SELECT DATEDIFF(horaConsulta, 'dataHoraAtual') > 30
```

   Onde DATEDIFF() é uma função em MySQL que devolve a diferença (em dias) entre duas datas.

3. Caso a operação tenha sido bem sucedida, arquivoRotas é destruído. Do contrário, além disso uma mensagem de erro é gravada no log do sistema. Por fim, a flag *ocupado* é setada para "falso".

2.4 É iniciada uma impressão de mala-direta de contas

1. Uma vez por dia, num horário arbitrário, escalonadorContas inicializa um objeto de *ImprimeContas*; nesses casos, a flag *ocupado* é setada para "verdadeiro".

2. O método *imprimeNaImpressora* então cria um objeto de *ArquivoRotas* e faz uma consulta ao banco, com as seguintes queries SQL:

```
SELECT *
  FROM Cliente
```

   Para cada cliente obtido com a query acima, faz-se uma query para buscar os dados de rotas do mesmo, do dia anterior. Para tal é feito:

```
SELECT ConsultaRota.ID_Rota, nomeVia, numeroInicial,
       numeroFinal, bairro, horaConsultaVia
  FROM ConsultaRota, RIGHT OUTER JOIN
       SELECT nomeVia, numeroInicial, numeroFinal, bairro, horaConsultaVia
         FROM UtilizaVias JOIN Via
           ON UtilizaVias.noInicial = Via.noInicial
          AND UtilizaVias.noFinal = Via.noFinal
    ON ConsultaRota.ID_Rota = UtilizaVias.ID_Rota
 WHERE SELECT DATEDIFF('dataDehoje', horaConsulta) = 1
 ORDER BY horaConsultaVia ASC
```

Na consulta acima, serão exibidos todos os registros do dia anterior com relação ao dia de chamada da query. imprimeNaImpressora então realiza o cálculo do valor da conta (proporcional ao número de vias utilizadas) e prepara para enviar os dados para a impressora.

3. imprimeNaImpressora envia as informações para a impressora. Caso tudo corra bem, o objeto é destruído após devolver "verdadeiro". Caso contrário, devolve "falso" e escreve uma mensagem de erro no log do sistema.

4. A flag *ocupado* é setada para "falso".

2.5 Encerramento do módulo *Rotas*

Para evitar-se inconsistências no Banco de Dados, assim como interromper rotas já inicializadas por clientes, o seguinte procedimento é adotado para o encerramento do *kernel* de Rotas:

1. O sistema verifica se a flag *ocupado* de servidoraCliente está setada (estado "verdadeiro"); se estiver, aguarda até que ela passe para o estado "falso". Após isso, destrói servidoraCliente

2. O sistema verifica se a flag *ocupado* de escalonadorImprimeContas está setada (estado "verdadeiro"); se estiver, aguarda até que ela passe para o estado "falso". Após isso, destrói escalonadorImprimeContas.

3. O sistema verifica se a flag *ocupado* de escalonadorMapaViário está setada (estado "verdadeiro"); se estiver, aguarda até que ela passe para o estado "falso". Após isso, destrói escalonadorMapaViário.

4. O sistema verifica se a flag *ocupado* de escalonadorLimpaRotas está setada (estado "verdadeiro"); se estiver, aguarda até que ela passe para o estado "falso". Após isso, destrói escalonadorLimpaRotas.

5. Finalmente, é exibida uma mensagem de término de programa, são destruídos os objetos restantes e o programa é encerrado.

3 Cenários do componente *CET*

3.1 Uma atualização de Mapa Viário é solicitada por *Rotas*

1. servidoraCET recebe uma solicitação de conexão de *Rotas*; ao se conectar, a flag *ocupado* é setada para "verdadeiro" e uma mensagem análoga à abaixo é recebida pelo componente *CET*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mensagem de atualizacao do Modulo "Rotas" a "CET". -->
<mensagem_atualizacao />
```

2. servidoraCET processa o XML:

   (a) se o XML não foi recebido corretamente, é enviada uma mensagem XML de erro análoga à do ítem (8.b) da seção (1).

   (b) se o XML foi recebido corretamente, servidoraCET verifica se foi solicitada uma inicialização ou uma atualização.

3. Caso tenha sido solicitada uma inicialização, é criada, através do método *geraListaInicialização* de mapaViárioCET, uma lista com todos as vias cujo atributo *taxaOcupação* seja maior que zero.

4. Caso tenha sido solicitada uma atualização, é criada, através do método *geraListaAtualização* de mapaViárioCET, uma lista com todos as vias cujo atributo *horaÚltimaatualização* tenha sido atualizado nos últimos 30 minutos.

5. Em ambos os casos, *CET* devolve um XML e encerra a conexão; o XML tem o seguinte formato:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mensagem de resposta de "CET" ao "Rotas". -->
<mensagem_resposta_inicializacao>
        <via>
                <no_origem>5678</no_origem>
                <no_destino>11456</no_destino>
```

```
                <taxa_ocupacao>0.27</taxa_ocupacao>
        </via>
</mensagem_resposta_inicializacao>
```

Onde quando trata-se de uma atualização muda-se apenas as tags externas.

6. Então, a flag *ocupado* de servidoraRotas é setada para "falso".

3.2 É iniciada uma atualização dos Perfis de Trânsito

1. A cada 30 minutos, escalonadorMapaViário chama o método *atualizaPerfilTrânsito* de mapaViárioCET; nesses casos, a flag *ocupado* é setada para "verdadeiro".

2. O método então cria uma lista com todos as vias cujo atributo *horárioÚltimaAtualização* foi alterado na última meia hora.

3. Com a lista do ítem anterior, é feita a seguinte query SQL no banco, para cada via da lista:

```
UPDATE FluxoVia
   SET taxaOcupacao = 'taxaAtual'
 WHERE noInicial = 'noInicial'
   AND noFinal = 'noFinal'
   AND horario >= SELECT DATE_SUB('hora_atual', INTERVAL 15 MINUTE)
   AND horario <  SELECT DATE_ADD('hora_atual', INTERVAL 15 MINUTE)
```

4. Caso tudo corra bem, o método simplesmente é encerrado. Caso contrário, escreve uma mensagem de erro no log do sistema.

5. A flag *ocupado* é setada para "falso".

3.3 Encerramento do módulo *CET*

Para evitar-se inconsistências no Banco de Dados, assim como interromper atualizações de Mapas, o seguinte procedimento é adotado para o encerramento do *kernel* de CET:

1. O sistema verifica se a flag *ocupado* de servidoraCET está setada (estado "verdadeiro"); se estiver, aguarda até que ela passe para o estado "falso". Após isso, destrói servidoraCET

2. O sistema verifica se a flag *ocupado* de escalonadorMapaViário está setada (estado "verdadeiro"); se estiver, aguarda até que ela passe para o estado "falso". Após isso, destrói escalonadorMapaViário.

3. Finalmente, é gravada, no log, uma mensagem de encerramento do sistema, são destruídos os objetos restantes e o programa é encerrado.

Referências

[1] G.R. Andrews. Foundations of Multithreaded, Parallel, and Distributed Programming. Addison-Wesley, 2000.

[2] MySQL Homepage. http://www.mysql.com/. *Acesso em 10 de fevereiro de 2009.*

[3] C.M.F. Rubira. Análise Orientada a Objetos. IC-Unicamp, 2000.

[4] A. Silberschatz e H. F. Korth. Sistemas de Bancos de Dados. McGraw-Hill, 1989.

[5] W3C XML Homepage. http://www.w3.org/XML/. *Acesso em 10 de fevereiro de 2009.*