Interação e Concorrência
Licenciatura em Ciências da Computação
3º ano
-

Trabalho Prático

Modelação de uma instância de um sistema baseado em serviços *Web*
Relatório

Grupo	
A81241	José Alberto Martins Boticas
A80584	Nelson José Dias Teixeira

11 de Junho de 2019

Resumo

De forma sucinta e simplificada, este trabalho prático tem como objetivo principal a construção de uma instância de um sistema baseado em serviços *Web*. O desenvolvimento deste sistema tem de obedecer alguns requisitos que irão ser expostos mais à frente neste documento. A especificação requirida sobre este problema é expressa na linguagem *mCRL2*. Esta é uma linguagem formal e é adequada para a modelação, verificação e validação de sistemas e protocolos concorrentes.

Conteúdo

Capítulo 1

Introdução

O desenvolvimento de serviços *Web* levantam muitos problemas a nível de engenharia de software. Alguns destes são encontrados de forma regular em paradigmas de programação passados. A implementação de serviços *Web* é propensa a erros devido à complexidade das interações e da troca de mensagens que têm que ser especificadas.

É comum este tipo de serviços ser disponibilizado na *internet*. Globalmente, estes são processos distribuídos e indepedentes que comunicam entre si através de troca de mensagens.

Ora, é aqui que reside a principal questão sobre os serviços *Web*: de que forma é necessário colocar vários processos a trabalhar juntos sobre a mesma tarefa? Para tal, vamos procurar dar resposta a esta pergunta através da implementação de uma instância de um sistema baseado neste tipo de serviços.

Este relatório está organizado em diversos capítulos por forma a garantir uma estruturação correta do mesmo. Apresentam-se de seguida os capítulos presentes e os respetivos conteúdos:

- Análise e especificação: exposição informal do problema em causa, apresentando-se os requisitos que este inclui.
- Concepção/desenho da resolução: apresentação da estratégia para a resolução do problema, evidenciando-se a solução proposta pelo grupo;
- Codificação e testes: apresentação dos resultados obtidos para cada teste efetuado;
- Conclusão: síntese das análises e da implementação do trabalho prático, fazendo-se algumas observações do mesmo;
- Código do projeto: exposição completa do código fonte que corresponde à solução adoptada pelo grupo.

Com isto dá-se início à análise e especificação do problema em questão.

Capítulo 2

Análise e especificação

2.1 Descrição informal do problema

Dando seguimento ao que já foi mencionado anteriormente, é de salientar que os sistemas baseados em serviços *Web* têm de ir ao encontro com as necessidades evidenciadas pelos seus utilizadores. Para tal, é necessário especificar estes requisitos com precisão e clareza. Desta forma, respeitando o objetivo deste projeto, é essencial a formulação de um número de requisitos operacionais para tais sistemas, bem como a modulação (em linguagem *mCRL2*) de um conjunto de processos que os implementem.

De forma abstrata é indispensável satisfazer os seguintes aspetos:

- introdução de recursos relativos a serviços *Web* como sendo processos concorrentes e independentes;
- fazer uso de pelo menos um tipo de dados definido pelo utilizador em *mCRL2*;
- testar o sistema final com um número variável de propriedades escritas na lógica extendida de *Hennessy-Miller*. Estas, por forma a serem verificadas posteriormente, têm de ser implementadas na linguagem *mCRL2*.

O caso que os elementos deste grupo têm em mãos é um exemplo relativo ao bem-estar público e extraído de uma análise de domínio maior sobre o governo local de um determinado município. Uma vez implementado, este sistema vai possibilitar o apoio a cidadãos idosos, fazendo com que estes recebam assistência sanitária da respetiva administração pública.
É apresentado na seguinte secção deste documento os requisitos intrínsecos a este exemplo.

2.2 Especificação de requisitos

O exemplo que foi exibido no fim da última secção envolve vários atores. Entre eles destacam-se:

1. **Cidadão**: Esta entidade faz pedidos, troca informação com a agência e aguarda por uma resposta. Caso seja aceite, recebe o serviço em causa e faz o respetivo pagamento;
2. **Agência sanitária**: Este interveniente trata de gerir os pedidos enviados por parte das pessoas idosas. Inicialmente pergunta pela informação relativa aos cidadãos que fizeram o pedido. Dependendo disso, este último tanto pode ser recusado e o cidadão em causa espera por um novo

pedido (como refletido por uma definição recursiva), como pode ser aceite. Neste último caso, a sincronização é feita com o controlador da cooperativa (responsável pelo controlo das duas cooperativas) para encomendar a entrega de determinados serviços (transporte ou refeições). Posteriormente, a agência paga ao banco algumas taxas públicas e espera que este último envie um sinal indicando que o pagamento foi concluído;

3. **Cooperativas de transporte e de refeição**: Tanto a cooperativa de transporte como a de refeição são entidades que prestam os seus serviços. Estas, num determinado momento do sistema, avisam o banco para iniciar o pagamento e aguardam a recepção das suas taxas. Posteriormente, um controlador recebe uma notificação da agência e disponibiliza um dos serviços possíveis.

4. **Banco**: Este interveninente do sistema recebe uma mensagem de acionamento da cooperativa envolvida na solicitação atual. Os diferentes pagamentos são então realizados: pagamento de taxas públicas pela agência e de taxas privadas pelo cidadão e pagamento da cooperativa pelo banco. Observe-se que a estratégia especificada para este serviço irá coletar o dinheiro primeiro e depois efetuar o pagamento do serviço.

Resumidamente, este sistema é composto pelas candidaturas de pessoas idosas à assistência sanitária, pelo atendimento de pedidos da respetiva agência, pelo serviço de transporte, pela entrega de refeições e pela gestão monetária do banco.

Feita a especificação dos requisitos associados a este problema dá-se início à implementação do mesmo no próximo capítulo.

Capítulo 3

Concepção/desenho da resolução

3.1 Implementação do problema

Quanto à codificação do sistema requerido, foram implementadas várias ações que transparecem a evolução momentânea do mesmo. Destas ações, salientam-se as que se sincronizam, que formam a maior parte do sistema e que garantem a ordem de execução correta. Assim, a sincronização de ações permite que o sistema funcione corretamente, garantindo, por exemplo, que o banco receba primeiro as taxas e só depois pague o serviço, ou que um cidadão usufrua de um serviçoo e que depois o pague.

Para além das ações referidas anteriormente, foram criados 6 processos, que fazem uso das mesmas, por forma a cumprir os requisitos exigidos:

- ***Bank***: processo que simula a ação do banco, esperando o pagamento das taxas por parte do cidadão e da agência e efetua, de seguida, o pagamento do serviço à cooperativa.
- ***Citizen***: processo que imita um cidadão que faz um pedido à agência, espera que o mesmo seja aprovado, usufrui do serviço pedido, caso chegue a aprovação e, por fim, efetua o pagamento das suas taxas, que lhe são comunicadas pelo banco.
- ***Agency***: processo que atende os pedidos dos cidadãos, aprova-os ou recusa-os, e trata de todas as diligências para que um serviço seja prestado e paga as taxas que lhe são impostas.
- ***Transport***: processo que diz respeito à prestação de serviços de transporte, notifica o banco para os valores a pagar por parte da agência e do cidadão e espera que o serviço lhe seja pago pelo banco.
- ***Meal***: processo que diz respeito à prestação de serviços de refeições, em tudo semelhante ao processo anterior.
- ***PassTime***: processo que simula a passagem do tempo.

De maneira a interligar todos os processos mencionados é necessário utilizar a noção de ***allow*** *e* ***comm*** em *mCRL2*. A primeira definição, *allow*, permite executar atomicamente as ações especificadas no mesmo, bloqueando as restantes. A segunda noção, *comm*, permite renomear conjuntos de ações para uma única ação. Essas diversas ações comunicam entre si (sincronizam-se) e dão origem à ação renomeada no processo.

Por fim, de forma a exibir o funcionamento do sistema, todos os processos em cima são executados em paralelo, sendo que são executados quatro processos *Citizen*, mas este número e os seus parâmetros podem variar. Isto é (em *mCRL2*):

```
    (Citizen(meal, 17) || Citizen(transport, 12) || Citizen(meal, 11) ||
Citizen(meal, 5) || Agency || Bank || Transport(70/3, 20/3) || Meal(70/3, 5)))
```

Capítulo 4

Codificação e testes

4.1 Verificação de propriedades (*model checking*)

Por forma a averiguar se o sistema construído possui um comportamento correto e esperado foram formuladas algumas propriedades, entre as quais destacam-se:

- **Propriedade 1**: Garantia de ausência de *deadlock* ao longo de todo o sistema.

```
nu X . ([true*]<true>true) && X
```

- **Propriedade 2**: Garantia de que sempre que um pedido é enviado num dado momento do sistema, é eventualmente verificado a aceitação ou recusa do mesmo.

```
nu X . ([true*][request_sent]
  (mu Y . <accepted || refused>true || [!(accepted || refused)]Y)) && X
```

- **Propriedade 3**: Garantia da impossibilidade de um cidadão pagar um determinado serviço antes de o poder usufruir.

```
nu X . (forall value:Real . [citizen_pay(value).use_service]false) && X
```

- **Propriedade 4**: Garantia de que o banco primeiro coleta o dinheiro e só depois efetua o pagamento do serviço, ou seja, após o pagamento de um valor à cooperativa, é impossível que haja de seguida um pagamento quer do cidadão quer da agência.

```
nu X . (forall v1, v2, v3:Real .
  [coop_pay(v1)][(agency_paid(v2) || citizen_paid(v3))]false) && X
```

4.2 Testes realizados e resultados

Após testar as propriedades mencionadas anteriormente, foi observado que todas estas eram verificadas, isto é, é retornado *true* por parte do *model checking* relativo ao *mCRL2*.

Para proceder a esta verificação, inicialmente converte-se o ficheiro *.mcrl2* para a extensão *.lps* (com a opção *regular2* ativada no menu *lin-method*). De seguida transforma-se este último ficheiro para a extensão *.pbes*, indicando a fórmula a ser testada. Por fim, de modo a saber a veracidade da fórmula, invoca-se sobre o ficheiro com extensão *.pbes* o método *pbes2bool* presente no menu *Analysis*.

Capítulo 5

Conclusão

Após a realização deste trabalho prático ficou mais claro para os elementos que compõem este grupo a utilidade da linguagem *mCRL2* sobre este tipo de problemas. Isto é, a modelação, verificação e validação de uma instância de um sistema composto por processos concorrentes. Foi possível ainda melhorar a compreensão dos sistemas concorrentes e dos seus problemas, limitações e soluções possíveis.

Apesar de este ser um sistema simples, foi possível perceber que é muito complicado modelar um sistema que permite o atendimento de múltiplos clientes e, ainda, que é difícil e exigente garantir que a ordem em que se executam as ações é a correta e permite sempre a progressão do sistema.

Apêndice A

Código do projeto

Lista-se a seguir o código que foi desenvolvido pelos 2 elementos que compõem este grupo:

```
sort Service = struct meal | transport;

act
    message_received: Real # Real; message_sent: Real # Real; message: Real # Real;
    request_received; request_sent; request;
    refused_received; refused_sent; refused;
    accepted_sent; accepted_received; accepted;
    coop_fees: Real; coop_paid: Real; coop_pay: Real;
    agency_paid: Real; agency_pay: Real; payment_agency: Real;
    citizen_paid: Real; citizen_pay: Real; payment_citizen: Real;
    question_received; question_sent; question;
    info_received: Service # Int; info_sent: Service # Int; info: Service # Int;
    completed_agency; completed_bank; completed;
    value_receivedA: Real; value_sentA: Real; priceA: Real;
    value_receivedC: Real; value_sentC: Real; priceC: Real;
    use_service; provide_service; service;
    tick; done;
    send_order_meal: Int; receive_order_meal: Int; send_meal: Int;
    send_order_transport: Int; receive_order_transport: Int; send_transport: Int;

proc Bank =
    sum valueA: Real . sum valueC: Real . message_received(valueA, valueC) .
    value_sentA(valueA) . value_sentC(valueC) .  agency_paid(valueA) .
    citizen_paid(valueC) . coop_pay(valueA + valueC) . completed_bank . Bank;

proc Citizen(s: Service, time: Int) =
    request_sent . question_received . info_sent(s, time) .
    ((accepted_received . use_service . sum value:Real . value_receivedC(value) .
    citizen_pay(value)) + refused_received). Citizen(s, time);
```

```
proc Agency =
    request_received . question_sent . sum service: Service . sum time: Int .
    info_received(service, time) .
    ((service == transport) ->
        (accepted_sent . send_order_transport(time) . (sum value:Real .
        value_receivedA(value) . agency_pay(value)) . completed_agency)
    <> ((service == meal) ->
        (accepted_sent . send_order_meal(time) . (sum value:Real . value_receivedA(value)
    <> refused_sent)) . Agency;

proc Transport(valueA: Real, valueC: Real) =
    sum time:Int . receive_order_transport(time) . provide_service . PassTime(time) .
    message_sent(valueA, valueC) . coop_paid(valueA + valueC) .
    Transport(valueA, valueC);

proc Meal(valueA: Real, valueC: Real) =
    sum time:Int . receive_order_meal(time) . provide_service . PassTime(time) .
    message_sent(valueA, valueC) . coop_paid(valueA + valueC) . Meal(valueA, valueC);

proc PassTime(time: Int) =
    (time > 0) -> tick . PassTime(time-1)
    <> done;

init
    allow({payment_agency, payment_citizen, coop_fees, message, request, refused,
           accepted, service, completed, question, info, priceC, priceA, send_meal,
           send_transport, done, tick},

        comm({agency_pay | agency_paid -> payment_agency,
              citizen_pay | citizen_paid -> payment_citizen,
              coop_pay | coop_paid -> coop_fees,
              message_received | message_sent -> message,
              request_received | request_sent -> request,
              refused_received | refused_sent -> refused,
              accepted_received | accepted_sent -> accepted,
              use_service | provide_service -> service,
              completed_agency | completed_bank -> completed,
              question_received | question_sent -> question,
              info_received | info_sent -> info,
              value_sentC | value_receivedC -> priceC,
              value_sentA | value_receivedA -> priceA,
              receive_order_meal | send_order_meal -> send_meal,
              receive_order_transport | send_order_transport -> send_transport},

             Citizen(meal, 17) || Citizen(transport, 12) || Citizen(meal, 11) ||
             Citizen(meal, 5) || Agency || Bank || Transport(70/3, 20/3) ||
             Meal(70/3, 5)));
```