5 Lab: Modelação de comportamento (interações)

Enquadramento

Os diagramas de comportamento da UML permitem modelar situações em que se desenrola uma colaboração relevante (e.g.: entre objetos do software). Para além do tempo/fluxo, os diagramas mostram também as partes intervenientes. O diagrama de comportamento largamente mais usado é o diagrama de sequência que se adequa especialmente bem ao raciocínio "por objetos".

Objetivos de aprendizagem

- Explicar a colaboração entre objetos necessária para implementar uma interação de alto nível ou uma funcionalidade de código, recorrendo a diagramas de sequência.
- Representar a evolução de estados de um objeto como uma máquina de estados.

Preparação

- informação tutorial: "sequence diagrams" e "state machine diagrams".

5.1

Explique a interação modelada no Diagrama 1.

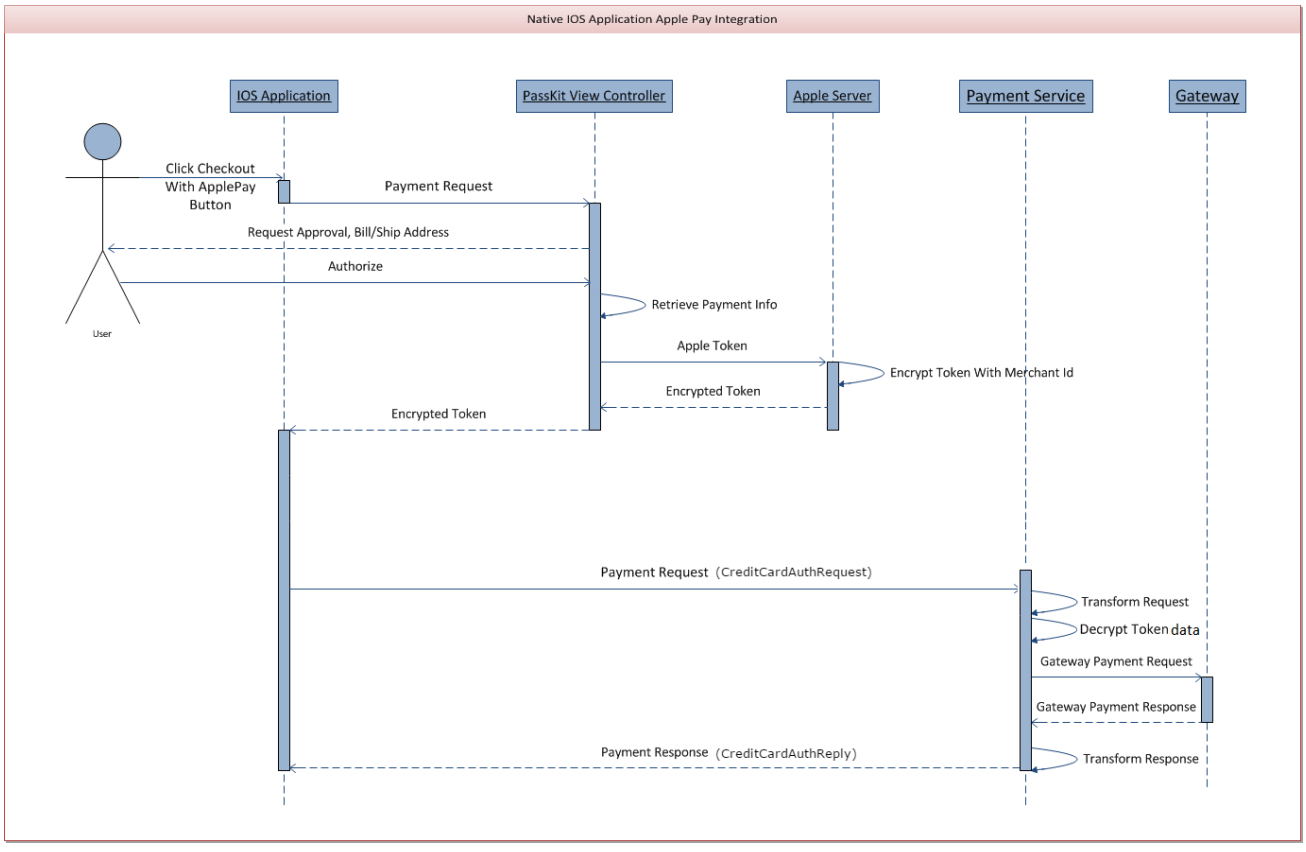

Diagrama 1: Integração de aplicações nativas (iOS) com o serviço Apple Pay. [In: https://docs.radial.com/ptf/Content/Topics/payments/apple-integration.htm]

5.2

Analisando o código disponível para o exercício 4.2 (lab 4), procure ilustrar (num diagrama de

sequência) as interações entre objetos que ocorrem quando a seguinte operação é solicitada:

- Pedido → calcularCalorias();

Nota: para criar cada *lifeline*, pode arrastar a classe correspondente (Pedido,...) da árvore do modelo para o diagrama, caso já as tenha criado.

5.3

O projeto "AirQuality" permite pesquisar a previsão de qualidade do ar (e da previsão meteorológica) para uma zona. Pode aceder ao código no GitHub. O *backend* do projeto está foi desenvolvido em Java e usa a "*build tool*" do Maven[1]. A solução disponibiliza uma API que permite interrogar a qualidade do ar, e.g.:

— http://127.0.0.1:8080/today/?address=Aveiro

O pedido acima levará à invocação do método:

— tqs.airquality.app.controller.AirQualityRestController#getAirQualityOfTodayFromCoordinates

a) apresente um diagrama de sequência para ilustrar a colaboração entre objetos que deve acontecer quando o utilizador invoca o método getAirQualityOfTodayFromCoordinates[2].

   Nota: o diagrama foca-se na troca de mensagens entre objetos próprios desta implementação (geralmente, omite-se o envolvimento de tipos de objetos da linguagem/JDK, como String, etc).

b) apresente também um diagrama de classes que represente a informação que se pode inferir do diagrama de sequência. Nota: não se pretende um diagrama (de classes) do projeto completo; apenas do que se "tira" da alínea anterior.

5.4

O projeto anterior utiliza o conceito de *cache* para otimizar as operações: guarda os resultados da previsão da qualidade do ar para uma localização, durante um tempo designado (*time to live* – TTL). Surgindo um pedido "idêntico", é respondido com os resultados anteriores guardados na *cache* (se ainda estiverem válidos), em vez de chamar a(s) API remota(s).

Independentemente dessa implementação, considere que:

— Um *put* [inserção] de um resultado [i.e., uma previsão para uma dada localidade] que ainda não existia, cria uma nova entrada [válida] na *cache*, e atualiza a marca temporal [*timestamp*]

— Um *put* de um resultado que já existia em *cache*, atualiza a validade daquela entrada.

— Um *get* de um resultado existente na *cache*, mas expirado, leva à sua remoção da cache.

— Um *get* de um resultado que não existe na *cache*, leva a um "*cache miss*" (mas não altera as entradas na *cache*).

— Passado o tempo TTL definido para uma entrada, ela passa a inválida [marcada como "*dirty*"], mas permanece na *cache*.

— Periodicamente, a cache é reavaliada e as entradas inválidas são retiradas.

Modele a máquina de estados associada a cada elemento [i.e., entrada] da *cache*.

---

[1] Num projeto Maven, o código encontra-se em "**src/main/java**/...". Para abrir o projeto localmente, deverá fazê-lo num IDE preparado para usar o Maven, e.g.: IntelliJ ou VS Code com as extensões do Java. Embora útil, **não é necessário** ter um ambiente de desenvolvido no seu computador, para abrir o projeto localmente. Pode "navegar" no código usando um *browser*, a partir do repositório Git.

[2] Para a resposta, precisa de analisar a implementação do método e identificar a "colaboração" de outros objetos [que são chamados a partir deste método].

5.5

Considere a seguinte informação recolhida pelo analista sobre a gestão da relação com os estafetas parceiros (de um serviço de entrega de comida baseado numa plataforma na Internet):

- A "frota" de estafetas é bastante dinâmica e é formada através da adesão de interessados à plataforma. Os interessados candidatam-se na Plataforma de forma espontânea; após a análise dos elementos solicitados, deverá haver uma aprovação da candidatura pela direção de logística, passando o proponente à condição de Estafeta; a parceria com os estafetas pode ser suspensa, ou até cancelada, em função de denúncias recebidas. O estafeta pode pedir para terminar a sua adesão a a qualquer altura.

A partir deste trecho (hipotético), explique se a informação aqui presente podia ser usada para criar os seguintes modelos bem como a relevância de o fazer. Para os casos considerados relevantes/adequados, apresente um "mini-diagrama".

a) Modelo de atividades
b) Modelo de casos de utilização
c) Modelo do domínio (com o diagrama de classes)
d) Modelo de interação (com o diagrama de sequência)
e) Modelo de (máquina de) estados (com o diagrama de estados).