Workshop 9

SWEN30006 SEM 2 2018

Extended Example: Architectural Design

This week we will be finalising our running example of the Point of Sale (POS) system by reviewing the system architecture challenges that we would face when implementing the POS system. We will be tackling this problem by looking at the architectural structure and control decomposition, and then modelling our solution using UML Component Diagram notation.

Requisite Knowledge and Tools

It is expected by this point that you are familiar with the following terms and concepts:

- Architectural Analysis
- Logical Architecture Refinement
- UML Architecture Diagram Notation
- Specifying Architectural Interfaces

It is suggested that you have attended the lectures on Architectural Design and Architectural Analysis and read, at a minimum, chapters 13, 33, 34, and 38 in Applying UML and Patterns by Larman (the prescribed text).

Background Knowledge

Here we provide some overviews of Architectural Design considerations and patterns that are commonly used when designing and describing software architectures.

Structural Decomposition Patterns

- Layered Architectural Pattern - Portable and maintainable with little or no interaction between the parts, and where the parts can be developed separately.
- Function Oriented Pipelining Pattern - Portable and maintainable where streams of data need to be transformed from one stream to another.
- Client Server Pattern - Manage access to shared data and resources in a situation where a large number of distributed clients wish to access the data and resources.
- Model-View-Controller - Very similar to the client server architecture. It can also be used in a situations where you need to manage a large number of distributed clients wishing to access shared data and resources.

Control Decomposition Patterns

- Centralised Control - One module has overall control for the sequencing of operations and for decision making.
- Decentralised Control - There is no one key module that has responsibility for control. Subsystems are autonomous and independent but co-ordinate their actions to achieve the system goals.
- Event Based Control - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.

It is possible - *and often necessary* - to combine different patterns to create a good architecture. It is also often necessary to adapt, modify or add to an architectural pattern to meet non-functional requirements. **The choice of final architecture often depends on the non-functional requirements!**

Specifying Architectures

Start by choosing a combination of decomposition patterns - either structural or control patterns - and then describe the responsibilities of each module, the relationships between modules and the collaborations between modules. Once you have worked out the decomposition, responsibilities and relationships then you have an opportunity to test the coupling and cohesion between modules.

Specifying Interfaces

Interfaces consist of the names of the publicly available operations together with their inputs, outputs and exceptions. The operations can be described by:

- UML Class diagrams, or UML interface specifications;
- Interface Description Languages that are available in many programming languages; or simply
- The names of the operations, inputs, outputs and exceptions, for example,
  - set time of day setTime(time : integer): void
  - pre-condition: time is a valid time
  - post-condition: the clock must be set to the provided input value

Exercise One - Core

Working in teams of 2 or 3, the aim is to develop a software architecture natural to the implementation of the POS system. Refer back to coverage of Architectural Analysis, including architectural factors and technical memos, as a guide.

Let's start by thinking about the problem!

- Can you group the requirements into cohesive subsets?
- What are the non-functional requirements in the problem?
- What requirements - *functional or non-functional* - will have the biggest influence on the architecture?

Now think about the possible architectures!

- What is a natural decomposition pattern for the POS system
- What is a natural control pattern for the POS system?

Using a simple block diagram, sketch your architecture and assign responsibilities to the modules in your architecture. Discuss the reasons for your choices with the class. Consider any frameworks or libraries which you could use to make implementing your architecture easier. Do you need to make any adjustments or changes to use these frameworks or libraries?

Exercise Two - Core

Specify your Software Architecture using Component Diagrams.

Now lets turn your simple block diagrams into UML component diagrams (you may wish to look here for help with the notation). Your task is to specify the POS system using a UML component diagram.

Here are some questions to help you understand your architecture better.

- You will need to think carefully about the interfaces between components, the internal structure of components - for example, are there subsystems existing inside your components?

- Look at your solution to the previous workshop. Is there any data that needs to be shared between components?
- Do you have operations on the interfaces between components that send or receive the shared data?
- Which of your components communicate using messages and which communicate using method calls?
- Can you specify the format of messages that must be sent between components?

Workshop Submission

You are required to demonstrate your solution to all core exercises, in your workshop, to your workshop tutor for review. If you do not complete all the exercises this week, you must complete the remaining exercises by next week's workshop and present these to your tutor in order to receive the mark for this workshop.
