March 15, 2020

Modeling, Simulation, and Analysis (MSA) Individual Final

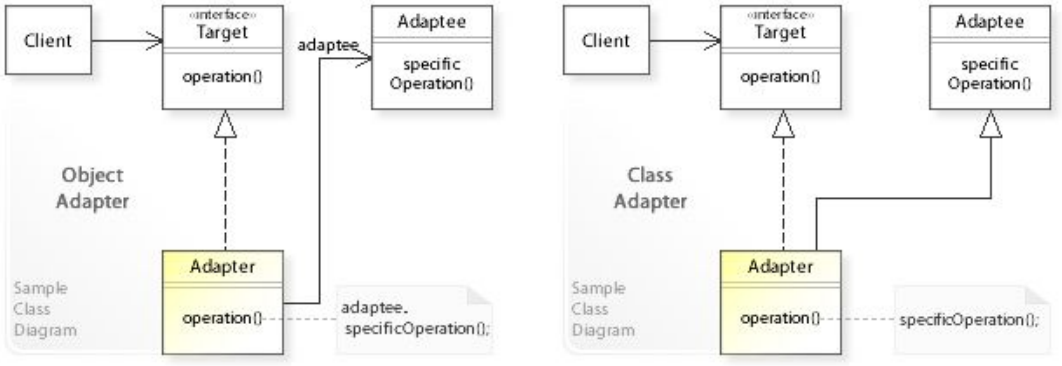
Nick McLoota

Link to Github Repository

1.Explain the difference between the Adapter and Decorator patterns.

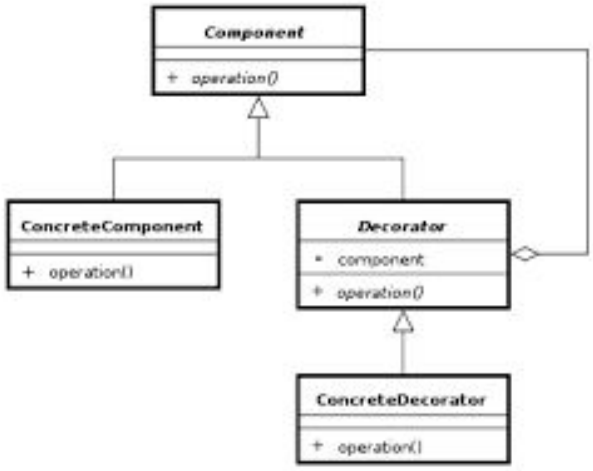
An adapter pattern is also known as a wrapper in software design pattern principles. The adapter pattern's main intent is to convert one interface to another in order to match what the client is expecting. Another appropriate time to use the adapter pattern is when the wrapper must respect a particular interface and must support a particular behavior. Allowing two previously incompatible interfaces to work together is a critical and real-world problem for developers and engineers. At a basic level, an adapter converts the interface of one class into an interface that another class is expecting.

Below is a nice sample image of an adapter pattern being utilized in a UML class diagram:



The primary intent of a decorator pattern is to dynamically add responsibility to an interface through wrapping the original code to work within a system. This makes it possible for a decorator to add or alter the behavior of an interface at or during run-time. The decorator pattern is useful for allowing behavior to be added to an individual object without affecting the behavior of other objects of the same class. These changes to an individual object's behavior are added dynamically.

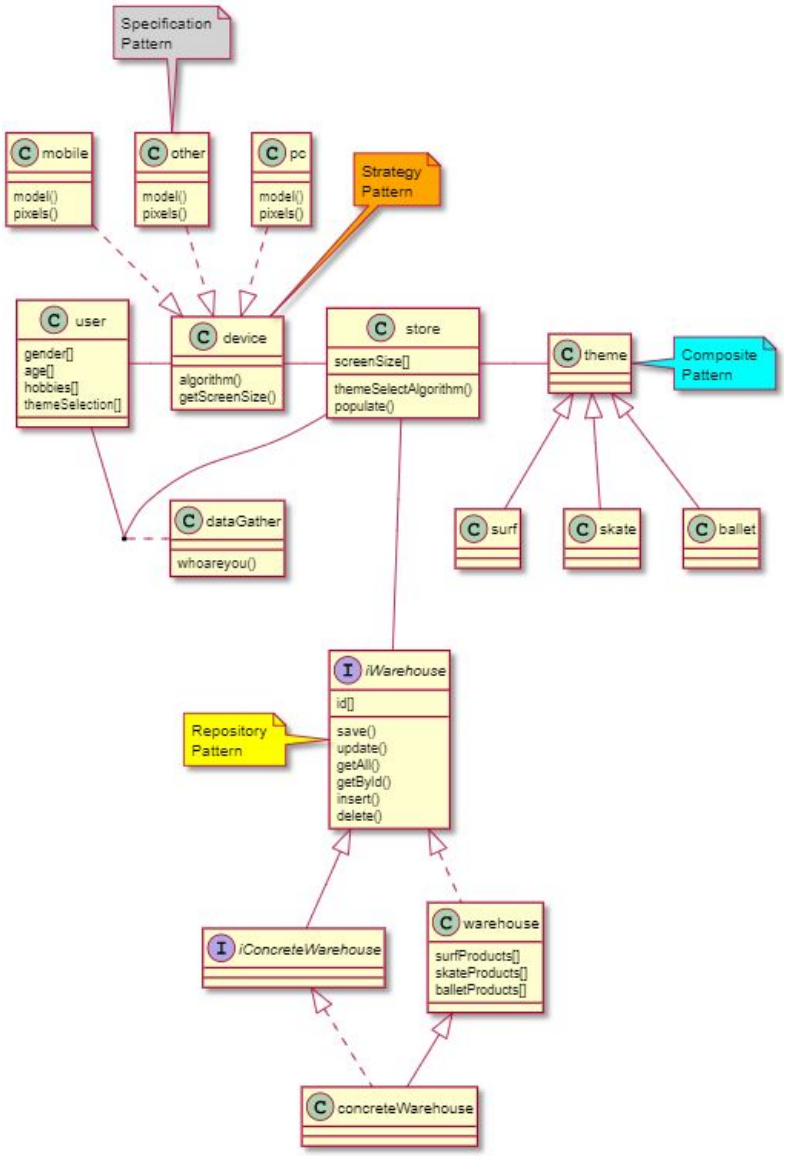
Below is a nice sample image of a decorator pattern being utilized in a UML class diagram:



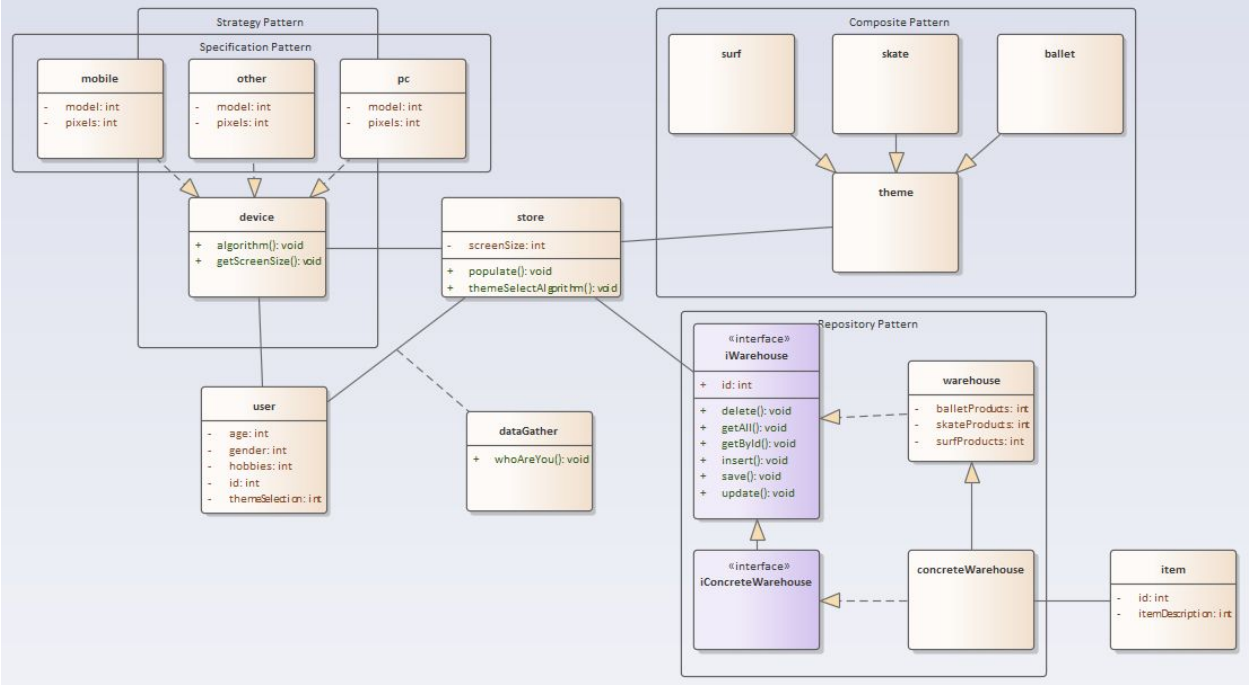
2.You are asked to model an online store that can have sub-stores based on a collection of themes (such as “surfer”, “music”, “books”, etc.) Give a domain model that allows populating the Store based on the selected theme collection. Use the following entities as a basis to build on: Theme, Store, Item Description, Warehouse. Utilize all of the following patterns: Composite(done), Repository(done), Specification(done), Strategy(done). Add the relevant entities and relationships/associations in a UML class diagram.

I utilized plantuml to develop a rough draft class diagram after drawing the diagram with pen and paper.

Shown below:



Next, I moved the model to Enterprise Architect due to the benefits of utilizing such an application for modeling.
Shown below:



All files for this final report can be found in my [Github Repository](#).

3. Explain the purpose of the following elements of the Rich Services SOA blueprint: Application Service, Business Service, Enterprise Service, Domain Component Service, Messenger, Router/Interceptor (or Orchestrator), Infrastructure Service, Data/Service Connector.

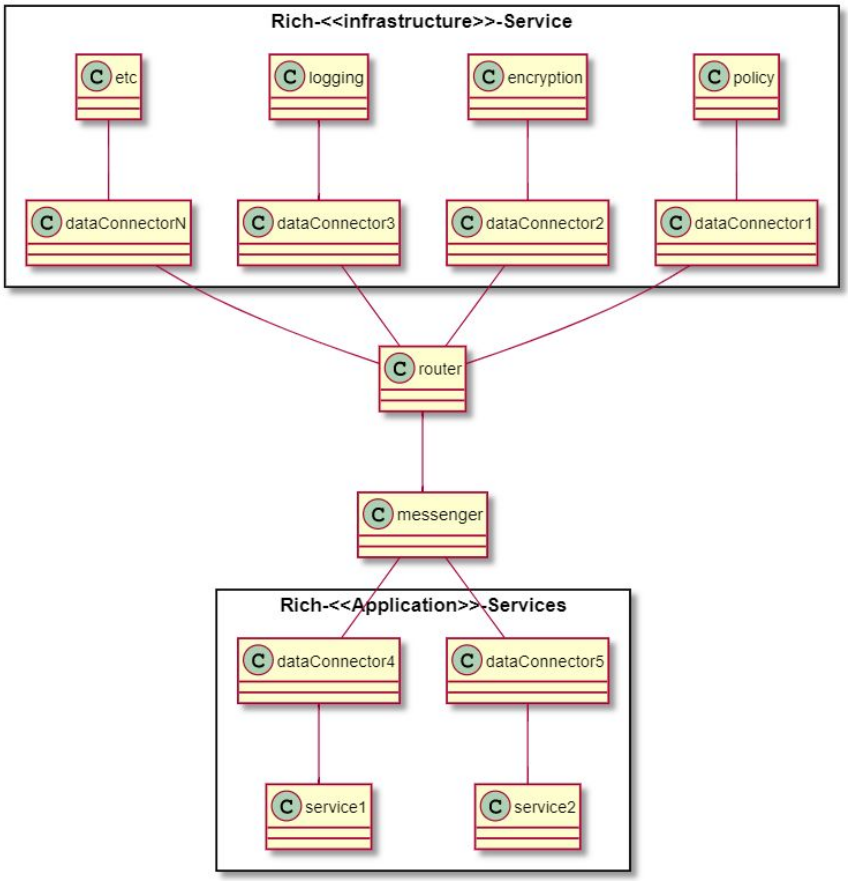
“The Rich Services Architecture is a type of service-oriented architecture (SOA) that allows hierarchical decomposition of a system architecture into separate concerns” (ieeeexplorer.com). The Rich Services SOA Blueprint allows for the capturing of different system aspects and the interactions between the loosely coupled system of systems. Crosscutting concerns can be sensibly accounted for with the Rich Services SOA Blueprint. Increased traceability is also a major benefit to using the Rich Services Architecture, and is allowed through the ability to directly translate from a logical to a deployed service-oriented architecture. Generating a model that can be easily leveraged into a deployment is an important aspect of the Rich Services SOA blueprint.

The Application and Infrastructure Services of the Rich Services SOA Blueprint help “address the vertical integration challenge” in enterprise architecting (Choppy & Sokolsky). Rich Application Services interface directly with the Messenger, but have no influence on how messages are routed. However, Rich Infrastructure Services interface directly with the Router/Interceptor and have the ability to change the routing of messages. The Data Connector encapsulates and hides the internal structure of the connected Rich Service. It also exports descriptions and interfaces in order to make the connected Rich Services externally visible. The Data Connector also has transformation capabilities to enable abstraction of the storage mechanisms implemented by a Rich Service. This makes a connection to a simple database versus a large collection or cluster of databases nearly indistinguishable - leading to cost savings when building the underlying physical infrastructure [Choppy 102].

Rich Application Services provide core application functionality, mandating the flow of business services. While Rich Infrastructure Services “do not initiate any communication by themselves, but reroute or filter messages defined by Rich Application Services” [researchgate]. Domain component services are “helper classes to coordinate activities between different domain model entities. This does not mean that domain services are equal to domain behavior. Entities have their own behavior, while domain services are simply intermediaries between said entities. The Enterprise Service Bus provides patterns responsible for tasks; these can range from routing to reachability; to interact between messages and protocol transformation and manages the SOA environment [windomjobs].

4.Rich Services SOA model

a) Use Enterprise Architect to create a domain model (UML class diagram) of the relationships between the elements of the Rich Services SOA blueprint. Be sure to express the notion of hierarchy adequately.



b) Explain briefly (2 paragraphs max) how you could model and implement a specific routing scheme for an instance of the blueprint.

Modeling and implementing a specific routing scheme for an instance of the blueprint could be done with Enterprise Architect (EA). EA “helps individuals, groups and large organizations model and manage complex information” (sparxsystems). As far as the actual implementation of the architecture, first, one would need a routing table to understand the topology of the network you are surrounded by. This routing table, or routing information base, is the data table stored in a router that will list routes to particular destinations over the network.

This could be implemented with the Rich Services SOA blueprint to enable a user to view network connections and routes hierarchically and from different perspectives. Addressing the challenge of verticality, the Rich Services SOA blueprint would be visually helpful when implementing a routing scheme. Complex routing could be supported with message routing based policies or call-outs to external services, while still having the ability to support point-to-point and one-to-many routing scenarios.

5.

• Suppose you have developed an architecture – both the functional and the physical architecture

• Suppose you have done all the necessary models, but you have not yet done the executable models and the analysis

• Suppose you are asked to present the architecture to management. At the completion of your presentation, the senior technical manager asks you:

- "How can you show that the proposed implementation of the architecture, as described by the physical architecture, will actually meet the requirements implicit in the functional architecture?"
    - Please state your answers to his question

First of all the physical architecture should consist of more than one logical model to describe the physical solution. The conceptual designs, drawings, schematics, and diagrams that define the system, its form, and the arrangement of its components and association interfaces should all help describe the physical architecture in a uniform manner. Iterative development and proper creation of a physical architecture should assure that all of the necessary components and interfaces necessary reside within the physical architecture. Being sure to document work, and steps taken, is a sure-fire way to assure stakeholders that you have taken the necessary steps and precautions when developing your architecture(s). The recursive loops taken while developing the physical architecture should be taken in order to evolve the physical architecture alongside the requirements and functional architecture. Design drivers can and should be identified as soon as possible, and therefore, it is imperative to identify key, driving requirements from the functional architecture and the combined processes of Stakeholder Requirements Definition, Requirements Analysis, and Architecture Design are all providing key insights to risk early on in the development cycle. Agile development processes will keep stakeholders happy, and if you are in close contact with the stakeholders on a regular basis, they should be in concordance that the requirements in the functional architecture will be met by the proposed implementation of the physical architecture.

6.

• Describe briefly what is meant by designing an architecture through a process of successive refinement and identify several managerial challenges in accomplishing that. For example, you can identify several current practices that impede the application of this approach.

Successive Refinement was conceived during development of IEEE 1801-2009 UPF (verificationacademy) and can be employed to define the power intent of a processor. It has to do with power consumption management and I am assuming the question is asking more about successive refinement in terms of agile development, not power consumption management; though the methodologies do have a lot in common when it comes to architecture design.

The primary idea of successive refinement that I will discuss is a methodology that enables designers to develop projects with more agility and to gain verification and simulation all in the same moment. To speed the design development process, the Successive Refinement Methodology aims to work at a variety of levels of abstraction in concurrence. Through the use of abstraction, designers have the ability to work with the complexity of a system while still hiding unnecessary details until later (Muller, Rosentiel and Ruf).

“Higher abstraction levels imply higher simulation speed” (Silva and Dias). With less harware implementation details to be described, simpler code can be generated; thus, leading to execution at a more rapid pace. Various methodologies and tools have been developed in order to allow for designers of many backgrounds to implement the Successive Refinement Methodology.

Works Cited

Admin. “Repository Pattern.” *DevIQ*, deviq.com/repository-pattern/.

Choppy, Christine & Sokolsky, Oleg. (2010). Foundations of Computer Software. Future Trends and Techniques for Development. 10.1007/978-3-642-12566-9

"Crafting Wicked Domain Models." Vimeo, uploaded by NDC Conferences, 8 years ago, https://vimeo.com/43598193.

Dias, Sandro, and Diogenes Silva. “The Successive Refinement Methodology in the Embedded Systems Design.” *Academia.edu - Share Research*, www.academia.edu/34191894/The_Successive_Refinement_Methodology_in_the_Embedded_Systems_Design.

“Enterprise Architect in 30 Minutes.” *Sparx Systems*, sparxsystems.com.au/enterprise-architect/.

“Find and Share Research.” *ResearchGate*, www.researchgate.net/.

“The GoF Design Patterns Memory - Learning Object-Oriented Design & Programming.” *w3sDesign*, 1 Oct. 2017, w3sdesign.com/?gr=s01&ugr=struct.

Ofner, Wolfgang, et al. “Repository and Unit of Work Pattern.” *Programming with Wolfgang*, 13 Jan. 2018, www.programmingwithwolfgang.com/repository-and-unit-of-work-pattern/.

“Physical Architecture.” *AcqNotes*, acqnotes.com/acqnote/careerfields/physical-architecture.

“SOA Blueprint in Service Oriented Architecture (SOA) Tutorial 16 March 2020 - Learn SOA Blueprint in Service Oriented Architecture (SOA) Tutorial (25044): Wisdom Jobs India.” *Wisdom Jobs*, www.wisdomjobs.com/e-university/service-oriented-architecture-soa-tutorial-1723/soa-blueprint-25044.html.

“Strategy Pattern.” *Wikipedia*, Wikimedia Foundation, 9 Jan. 2020, en.wikipedia.org/wiki/Strategy_pattern.