

Architecture Mismatch Patterns

Task 1: Medi(t)ative Air

Design an application which enables you to book the cheapest flight to a destination of your choice out of a number of providers.

1a) Assume, every provider is known in advance, and implements an interface `IFlightProvider`, which provides operations for querying for a connection, and for booking a flight. Develop an architecture which enables clients to interface to these providers and book the cheapest flight on offer for the destination and date they are interested in. Flight providers should require (and receive) no knowledge on other flight providers known to the system. Also, clients should not need to know which flight providers are registered with the system.

Which design pattern could you use?

1b) Many airlines offer on-line booking services as web services. How can you incorporate such an airline as a flight provider?

Task 2: Photo-realistic Facade

Ray tracing is a rather complex technique. It consists of a number of steps from parsing a scene-graph description (often called a ‘script’), building a scene-graph instance in memory, optimising the scene graph, tracing rays through all pixels of the target image, possibly oversampling to provide anti-aliasing, to actually rendering the image; that is, transforming the ray colour values into the value range of image colour values. On the other hand, as a client all you want to do is provide a script and obtain an image.

2a)

Use the FACADE pattern to provide clients of a ray-tracing subsystem with easy access to ray-tracing functionality.

Task 3: Producer-Consumer with Mediator

Assume, you are realising a UNIX shell. All programs running on the UNIX shell are provided with three communication channels: `stdin`, `stdout`, and `stderr`.

3a) Design interfaces for `stdout`, `stderr`, and `stdin`. Sketch an implementation for these interfaces by the `'ls'` command, and how this would be used by your shell implementation.

3b) The `'argouml'` program is a visual editing tool for UML diagrams. Assume that the model being edited is encapsulated completely behind an instance of `ModelFacade`. What design pattern could you use to allow for `'argouml'` to be used for manipulating UML models from the command line? It should accept an XMI file on `stdin` and output a transformed XMI file on `stdout`.

3c) Which design pattern is used by your shell to realise a command like

```
cat mymodel.xmi | argouml - > mytransformedmodel.xmi
```

?

Task 4: Pattern Relations

In this task you will explore the relations between the various patterns that we have been looking at in the course so far.

4a)

Compare `TEMPLATE METHOD` and `TEMPLATE CLASS`. What do they have in common, what is the major difference? How do they achieve variability? What is their relation to the `TEMPLATE HOOK` and the `OBJECTIFIER` patterns?

4b)

Compare the extensibility patterns `DECORATOR`, `COMPOSITE`, `CHAIN OF RESPONSIBILITY`, and `OBSERVER`. What are the mechanisms through which they achieve extensibility? Why does `PROXY` not provide extensibility? What is the relation of these patterns to `TEMPLATE CLASS` and `OBJECT RECURSION`?

4c)

Now compare the architecture-glue patterns `ADAPTER`, `FACADE`, and `MEDIATOR`. How do they cope with architectural mismatch? How do they compare to the variability and extensibility patterns?

4d) *

Sketch a chart of the relations between the design patterns `TEMPLATE METHOD`, `TEMPLATE CLASS`, `OBJECTIFIER`, `BRIDGE`, `STRATEGY`, `STATE`, `VISITOR`, `PROXY`, `ADAPTER`, `FACADE`, `MEDIATOR`, `OBJECT RECURSION`, `DECORATOR`, `COMPOSITE`, `CHAIN OF RESPONSIBILITY`, and `OBSERVER`. Use arrows to indicate specialisation (based on class structure, behaviour, or intent) and introduce additional helper concepts if you need them to represent commonalities which have not yet been abstracted into an individual pattern.