Logical Architecture Refinement:

The term logical architecture refinement defines changes to be made in logical architecture.

objectives:

* Explore further problems in logical design and layer pattern, including inter layer collaboration.
* Include the logical architecture for case study iteration.
* In relation to the architectural layers, apply the Facade, Observer and controller patterns.

Modifications made are:

→ Inter layer and inter package Coupling.
→ Inter layer and inter package interaction - scenarios.
→ Collaborations with layer patterns.
→ Session facades and the application layer
→ System operation and layers.
→ Upward Collaborations with observer
→ Relaxed layered Coupling.

Eg: NextGen Sys (Refer book for all Points)

* Inter layer and Inter package Coupling:

To understand the logical architecture of NextGen System, it also informative to include a diagram in the logical view that illustrates coupling b/w layers and packages.

Applying UML:

1) The dependency lines can be used to communicate coupling b/w packages

2) plain dependency lines are excellent when the communicator just wants to highlight general dependencies.

3). Another use of package diagram is to hide the specific types and focus on illustrating the package - package coupling.

Eg refer text book (495)

* Inter layer and Inter-package Interaction Scenarios:

1) package diagrams show static

information.

2) To understand the dynamics in the NextGen logical architecture, it is use to include diagram of how objects across layers connect and communicate. Thus interaction diagram is useful.

Applying UML:

1) The package of a type can be shown by qualifying the type with UML path name expression <packageName>::<Type Name>.

eg: Domain::Sales::Register.

2) This can be used to highlight the inter-package and inter-layer connections in the interaction diagram.

3). A Subsystem is a discrete entity that has behavior and interfaces

4) A Subsystem can be modeled as a special kind of package.

* Collaborations with the layers pattern.

Two design decisions at an architectural level are;

1) what are the big parts?
2. How are they connected?

→ The Architectural layer pattern guides defining the big parts.

→ Micro architectural design patterns such as Facade, Controller and Observer are used for design of connections b/w layers and package.

Simple package Vs Subsystem (Refer b/c)

Facade;

* For packages that represent subsystems, the most Common pattern of access is Facade, a GoF design pattern

GoF → Gang of Four

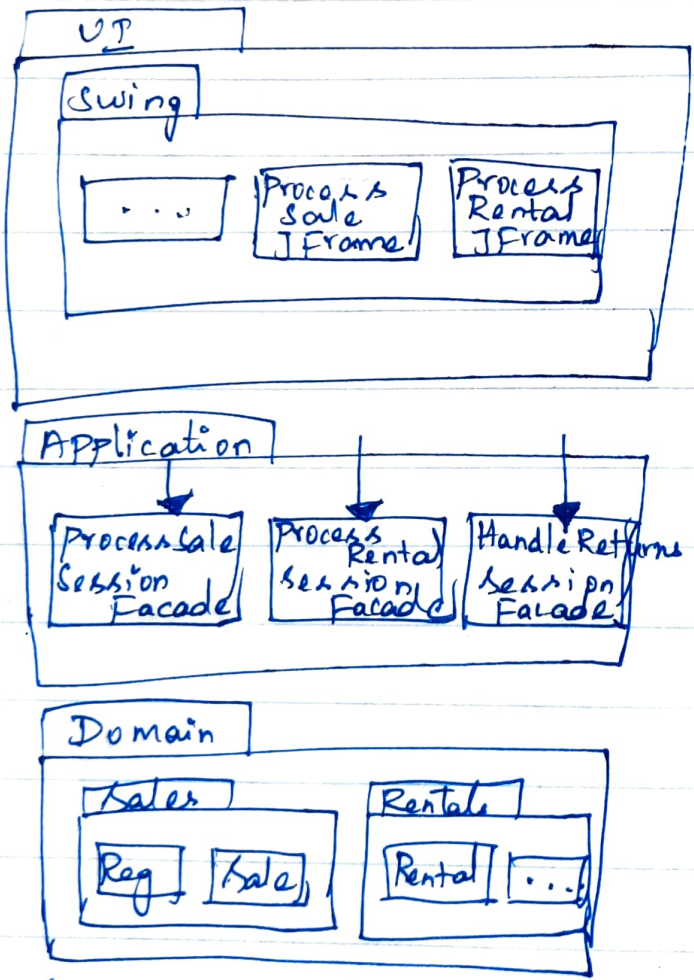
* Public facade object defines the services for the subsystem and clients collaborate with the facade, not internal subsystem components.
* Eg: For access to the rules engine and persistence we use posRulesEngine Facade and Persistence Facade.
* The facade should not normally expose a many low-level operations, it is desirable for the facade to expose a small number of high level operations.
* Facade is a mediator to the underlying subsystem objects, which do the work.

* Session Facades and the Application Layer:

→ when an application has many System operations and supports many usecases, it is common to have more than one object mediating b/w the UI and domain layers.

→ However, as the system grows to handle many usecases and system operations, it is common to introduce an application layer of objects that maintain session state for the operations of a usecase, where each session instance represents a session with one client. These are called "SESSION FACADES".

Eg:



Session Facades and Application layer

Controller

The GRASP Controller pattern describes common choices in client-side handlers for system operation requests emitting from the UI layer.

* **System Operations and layers.**

→ The SSDs illustrate the system operations, hiding UI objects from the diagram.

→ The system operations being invoked on the system are requests being generated by an actor via UI layer, onto the application layer.

* **Upward Collaboration with observer:-**

→ The facade pattern is commonly used for "downward" collaboration from a higher to lower layer.

→ When the lower layer needs to communicate upward with UI layer it is usually by "observer pattern".

→ That is UI objects in the higher UI layer implements an interface such as

* property Listener (or)
Alarm Listener.

These are the subscribers (or) listeners to events coming from objects in lower layers.

→ The lower layer objects are directly sending messages to UI objects but the coupling is only to the objects interface such as property listner.

*Relaxed Layered Coupling :-

In the protocol model, there is strict restriction that elements of layer N only access the services of immediate lower layer N-1.

The standard one is "relaxed layered" or "transparent layered" architecture in which elements of a layer collaborate with several other layers.