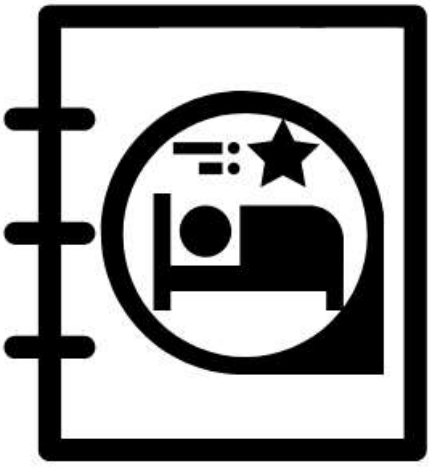
California State University Fullerton
CPSC 462

Object Oriented Software Design
SW Architecture Document (SAD)
for the



Hotel Reservation System

Josh Ibad
Chief Software Architect
joshcibad@csu.fullerton.edu

Revision History:

Version	Date	Summary of Changes	Author
1.0	2021-11-15	• Initial Release	Josh Ibad

Table of Contents

1 Architectural Representation....2

2 Architectural Decisions....3

2.1 Controller GRASP Decision....3

2.1.1 Decision to be made....3

2.1.2 Options Considered....3

2.1.3 Selection and Rationale....4

3 Logical View....5

3.1 Package Diagrams....5

3.1.1 Presentation (UI) Layer Components....5

3.1.2 Domain (Application) Layer Components....5

3.1.2.1 *Hotel*....5

3.1.2.2 *Reservation*....5

3.1.2.3 *Session*....6

3.1.3 Technical Services Layer Components....6

3.1.3.1 *Persistence*....6

3.1.3.2 *Logging*....6

3.2 Interface Diagrams....6

3.2.1 Presentation (UI) Layer Interface Diagrams....6

3.2.2 Domain Layer Interface Diagrams....6

3.2.3 Technical Services Interface Diagrams....7

1 Architectural Representation

After careful deliberation, I have determined that it is best to follow a Session Controller GRASP pattern when it comes to the Architecture Decision of the Controller GRASP Decision. The Session Controller GRASP pattern seems more appropriate for the Hotel Reservation system considering the granularity of the system's design and all the components in the Domain layer (to be discussed shortly). The alternative, the Facade Controller pattern, would have provided a singular system-level interface at the Domain level, that would have had high dependencies on all components of the Domain level. This means that the Facade option would have introduced high coupling in the system which would have made it harder to maintain and evolve. Namely, it scales horribly in the face of a growning number of functionality and classes within the components.

In comparison, the option of choice, the Session Controller pattern, is a use-case level interface at the Domain level, with one interface for every one or many use cases. Approximately one session controller interface would exist for each component, allowing for high cohesion amongst the classes or subcomponents within the component or subpackage. This also minimizes the coupling amongst components, which allows for highly modularized and highly scalable designs, that scales well when more functions and classes are added into the system in subsequent iterations. Thus, the Session Controller option is the option of choice when it comes to the Controller GRASP decision.

With that in mind, the overall packaging and hierarchy of the system is to be highly organized in a multi-tier / layer architecture so as to separate concerns or responsibilites amongst layers. Namely, there should be a UI layer responsible for receiving user input and formatting output, as well as relaying events into the lower layers of the system. A Domain layer in turn, would be responsible for business logic. The Domain layer will highly reflect the concepts of the Business domain and the relationships and functionalities between them. These two higher layers then relay inforamtion to the TechnicalServices layer, which is responsible with low-level responsibilities such as Secondary Storage management / persistence as well as the logging of events.

In further granularity, the Domain layer should have three components: Hotel, Reservation, and Session, each having a single interface that serves as a handler (as discussed above with the Session Controller decision). The Hotel component will be responsible for managing the Hotel as a container of Rooms, along with information relating to Rooms. The Reseration component in turn, will be responsible for managing Reservations, and more specifically, the reservation times and payment information for the reservation. The Reservation's association to Rooms will be minimized to produce low coupling between the two components. Lastly, the Domain layer will also have a Session component, which is responsible for managing user roles and permission. The Session does not lead directly to the two components, but rather, will direct the UI as to when it should use the other component's functionalities, and only exposes the functionalities a certain user's Session is allowed to have.

The TechnicalServices layer will have the typical components of Persistence and Logging. The Persistence component will, down the line, manage the persistence of data through CRUD operations, freeing all other components of the responsbility of writing and reading to files or databases. The Logging component in turn will manage the logging of system events in a well formatted manner, freeing all other components of the responsibility of excessive formatting.

2 Architectural Decisions

2.1 Controller GRASP Decision

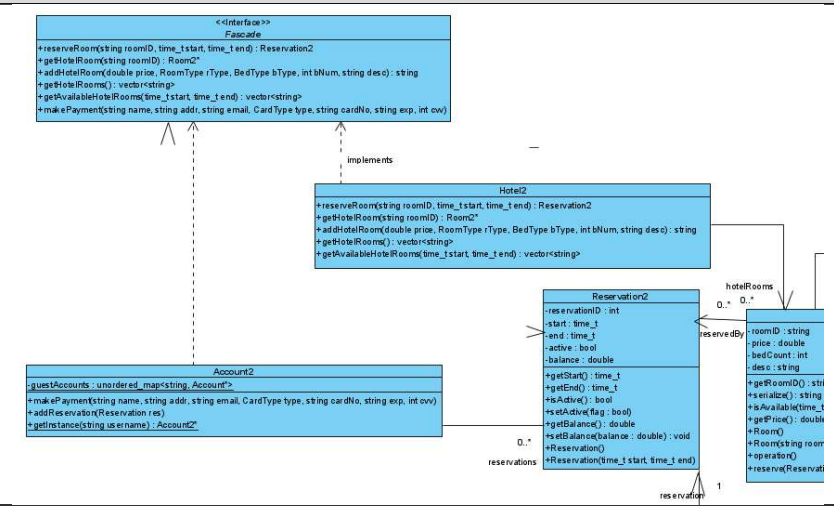
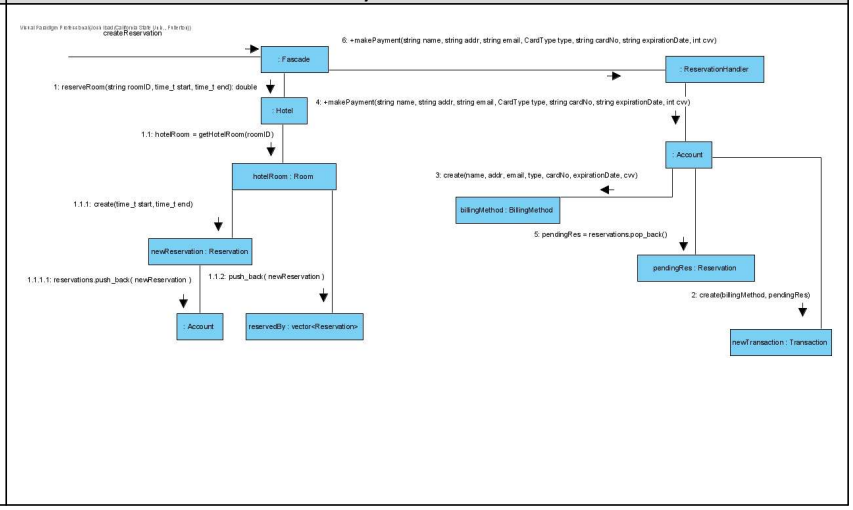
2.1.1 Decision to be made

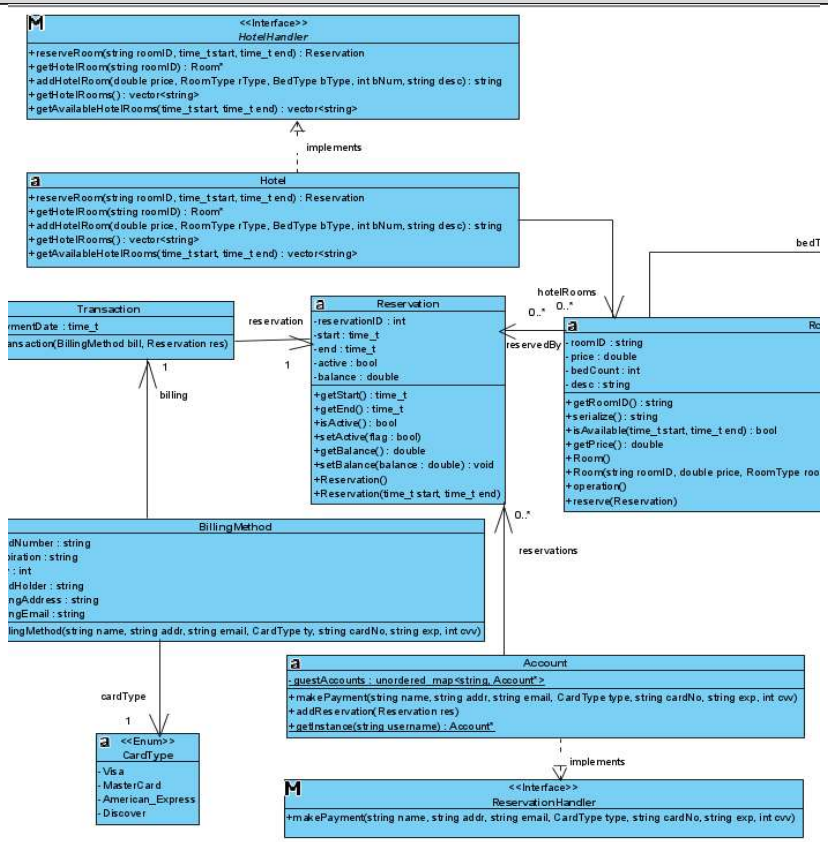
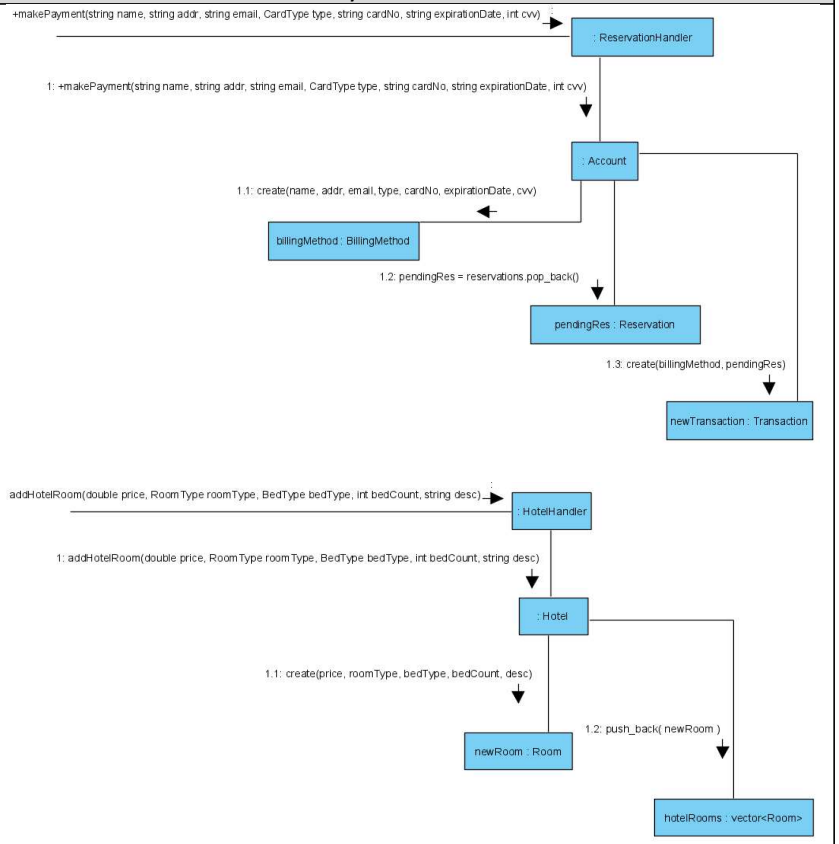
Who should be responsible for handling an input system event whenever a user attempts to manage hotel rooms?

A good decision should provide good abstraction and minimize coupling by making sure that system level events are only concerened with system level details and have implementation details hidden below.

A poor decision would have too many Classes or interfaces that must be interfaced with for a simple procedure, and would be unscalable as classes and interfaces are added when the system grows.

2.1.2 Options Considered

Controller	Static View	Dynamic View
Option 1: Facade (System) Controller (Rejected)		

Controller	Static View	Dynamic View
Option 2: Session (Use Case) Controller (Selected)		
Design Model Reference	In document 09 - Design Model, at section '1 Static View', under subsection '1.1 Hotel Reservation system - Room and Reservation Management'. (Page 2).	In document 09 - Design Model, at section '2 Dynamic View', under subsection '2.2 addHotelRooms Sequence of Execution' (Page 4), and subsection '2.6 makePayment Sequence of Exeuction' (Page 8).

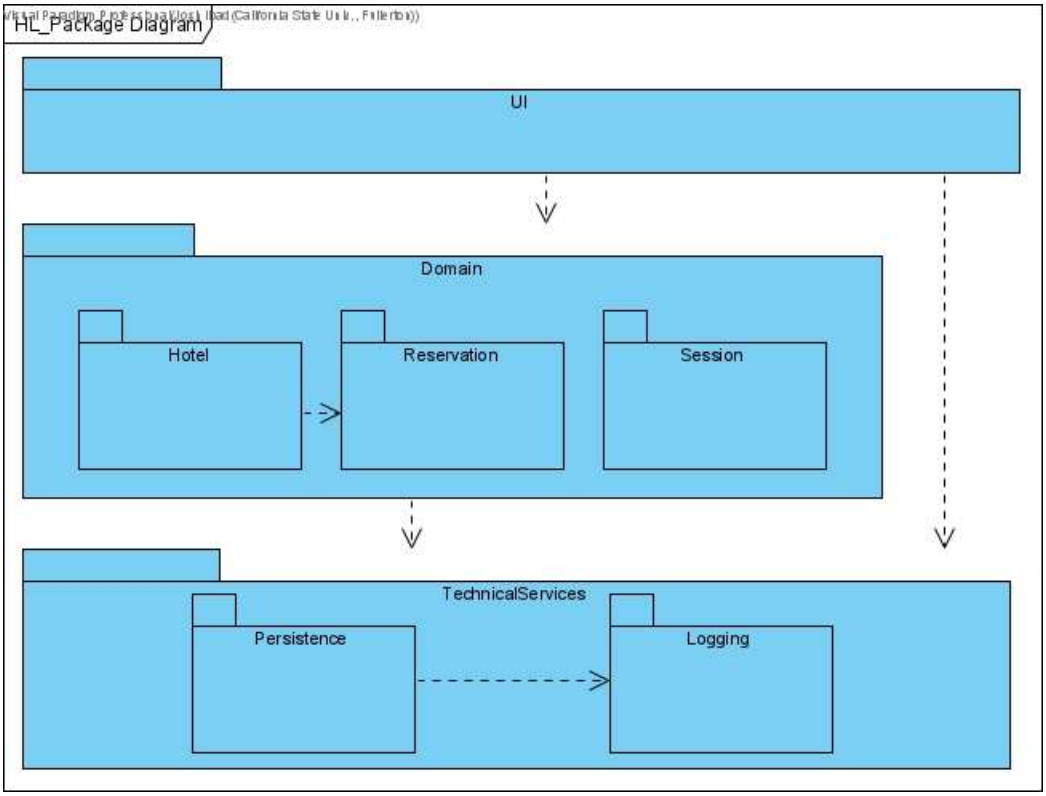
2.1.3 Selection and Rationale

Option 1 has been discarded because, although it centralizes all interaction into a single system-wide Facade controller, it also means that the singular controller will be highly coupled with all other Domain-layer components and classes. In the long-run this means higher coupling and it also means lack of scalability as more functions and classes are added in the Domain-layer components.

Option 2 has been selected because it elegantly permits the subdivision of the Domain-layer into a subset of low-coupling components or sub-packages that scales well when more functionalities and classes are added. By making each system level message a single interface method relayed to a single component, rather than use case messages that then send system-level messages scattered all over the domain layer, we have have a better design with the Session controllers (use-case level controllers with handlers for each component).

3 Logical View

3.1 Package Diagrams



3.1.1 Presentation (UI) Layer Components

N/A

3.1.2 Domain (Application) Layer Components

3.1.2.1 Hotel

The *Hotel* domain component is the component responsible for the Hotel concept, as a container of Rooms. The primary corresponding use case that the *Hotel* component is responsible for is the *Manage Hotel Rooms* use case. The *Hotel* component holds in it, classes such as Hotel and Room as well as the interface HotelHandler. The *Hotel* component's responsibility is to keep track of a Hotel and it's Rooms, along with the description of said Rooms. These Rooms have a price, descriptions, bed count, along with a BedType and RoomType. Any functionality, class, enumeration, or interface to support the *Manage Hotel Rooms* use case are packaged into the *Hotel* component.

3.1.2.2 Reservation

The *Reservation* domain component is the component responsible for the concept of Reservations and payment related concepts such as BillingMethods, Transactions, and Accounts. The primary corresponding use case that the *Reservation* component is responsible for is the *Manage Reservations* use case. The *Reservation* holds in it, classes such as Reservation, BillingMethod, Transaction, Account as well as the interface ReservationHandler. The *Reservation* component's responsibility is to keep track of reservations and payments for these Reservations, along with the times of reservation, time of payment, billing information such as card number, expiration, cvv codes, card holder name, address and contact, card types, payment dates, etc. Any functionality, class, enumeration or interface to support the *Manage Reservation* use case that deals explicitly with the Reservation rather than the Room, is packaged into the *Reservation* component.

3.1.2.3 Session

The *Session* domain component is the component responsible for serving as an actor's user Session in the system. The *Session* component facilitates the authentication functionality, which is not an explicit use case mentioned earlier, but is an essential supporting functionality that allows all other use cases to occur. The *Session* component takes care of user roles and permissions, making sure that they are both able to reach the functionalities that they are intended to have, and ONLY the functionalities and interfaces that they are intended to have. The Session draws greatly from the TechnicalServices layer's *Persistence* component to store user credentials and roles.

3.1.3 Technical Services Layer Components

3.1.3.1 Persistence

The *Persistence* technical services component is the component responsible for managing persistent data, that is, data that persists throughout system usage and system shutdowns. Ideally, a *Persistence* component would perform all CRUD operations to store and manage information, but temporarily, it solely performs Reads from regular text files in order to read the configuration of the UI and Logging.

3.1.3.2 Logging

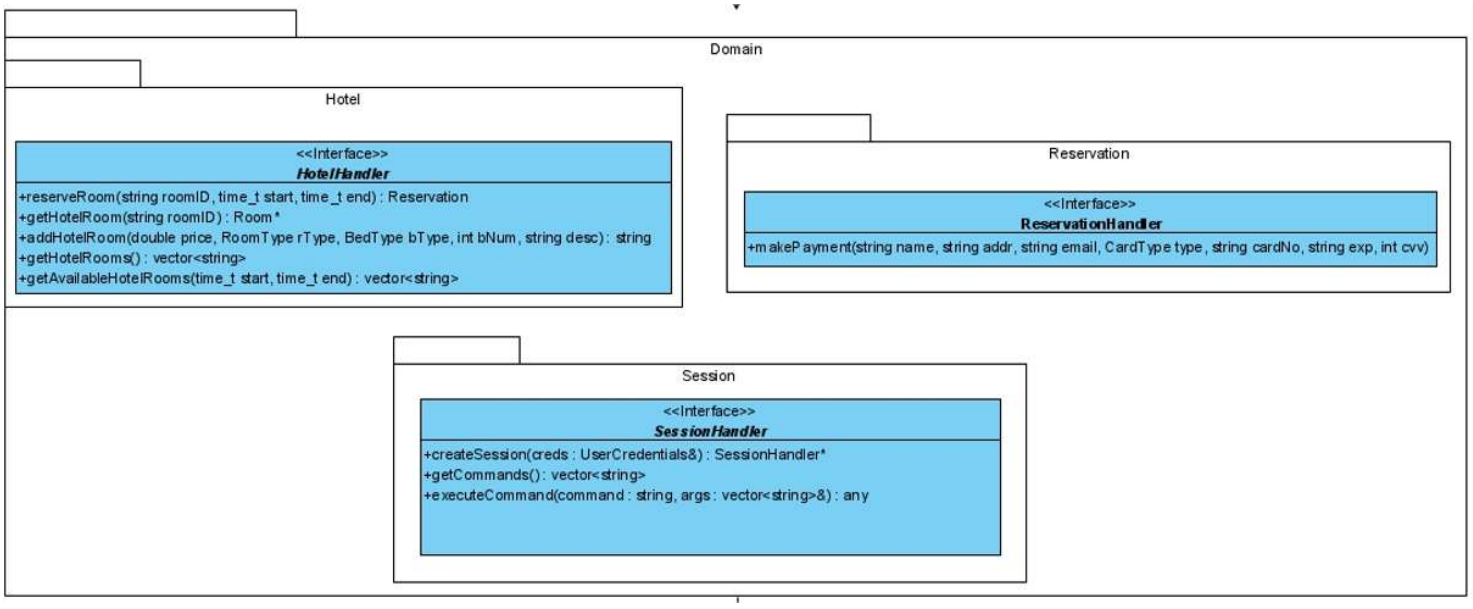
The *Logging* technical services component is the component responsible for logging system events and interactions. It takes care of printing messages into the command line in a well formatted manner, printing out the date and time of an event, followed by messages from the corresponding event. Logging provides insight to the functionality of technical services components such as the Logging and Persistence along with the UI layer. It also has the potential use of logging system events occuring the Domain layer if found necessary. With the first iterations of the Hotel Reservation system being a console application, the *Logging* component is essential for giving the user information on the system.

3.2 Interface Diagrams

3.2.1 Presentation (UI) Layer Interface Diagrams

N/A

3.2.2 Domain Layer Interface Diagrams



3.2.3 Technical Services Interface Diagrams