COS30041 Creating Secure and Scalable Software

Lecture 08 Architecture Patterns

Learning Objectives

- After studying the lecture material, you will be able to
  - Understand and describe simple architectural patterns used to implement business logics
  - Understand and describe the advantages and disadvantages of using a particular architectural pattern
  - Understand the issues involved in programming a particular architectural pattern

Pre-requisite

- Object Oriented Programming
- Some experiences on OO Design / Modular Design

Outline

- Architectural Patterns
- Façade Patterns
  - Session Façade
  - Web Services Façade
  - Message Façade
- Business Interface [Lectures on EJB]
- Data Transfer Objects [Lectures on JDBC and JPA]
- Data Access Objects [Lectures on JDBC and JPA]

Roadmap

- **Architectural Pattern**
- Façade Pattern
  - Session Façade
  - Web Services Façade
  - Message Facade

Architectual Pattern

- aka Design Pattern in the LARGE (distributed environment)
- Recurring solutions to common problems in software design
- Best practices in developing integrated software components to work together for frequently occurred problems

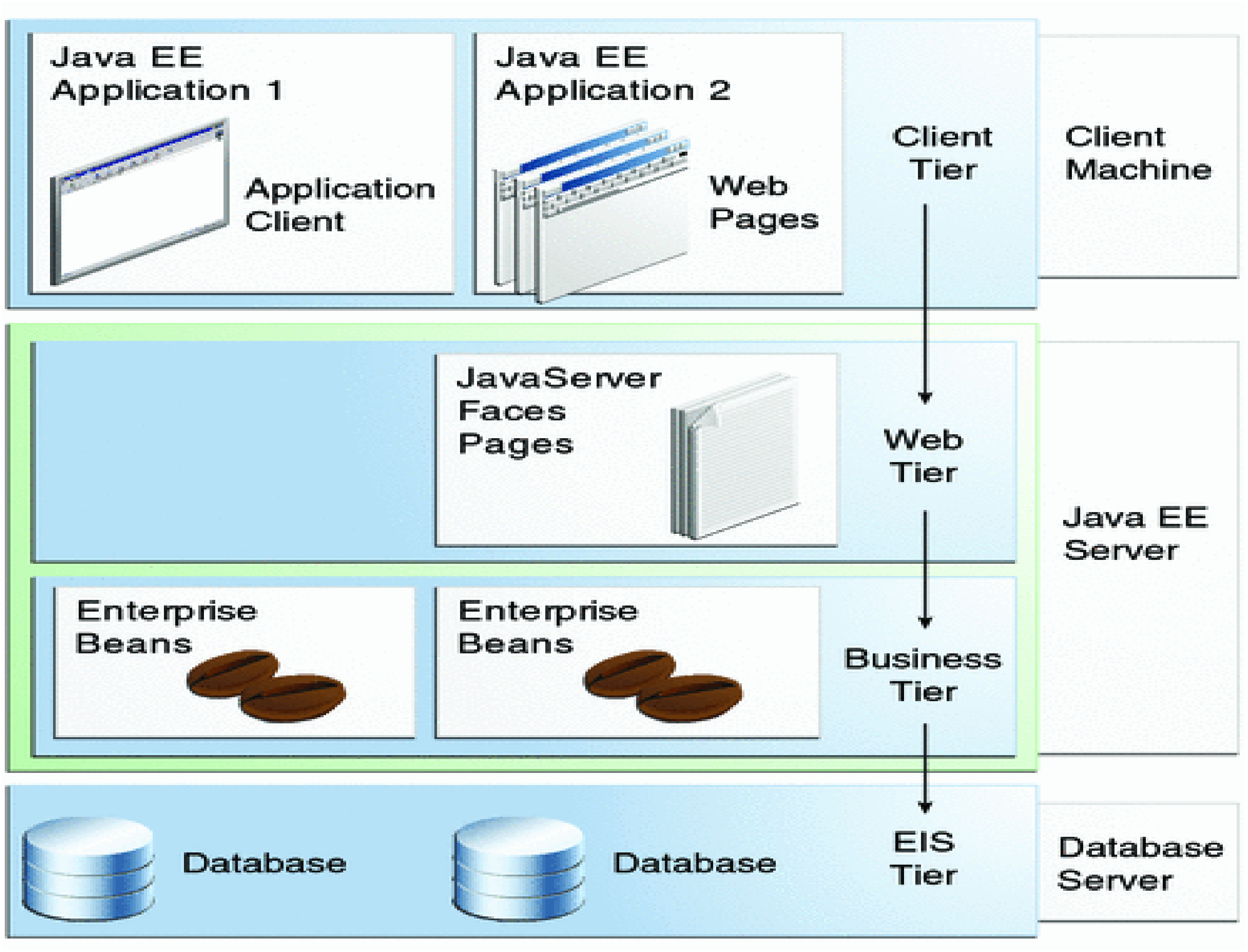
Famous Examples of Design Patterns

- Model-View-Controller (MVC)
  - Data; Presentation; Business / Presentation Displaying Logics
    - JavaBeans; JSP; Servlets
    - Managed Beans; JSFs; FacesServlet (page navigation rules)
- Boundary-Entity-Controller (BEC) [UML 1.0; design]
  - UI interfacing; Data; Controller

Architectural Pattern – Why?

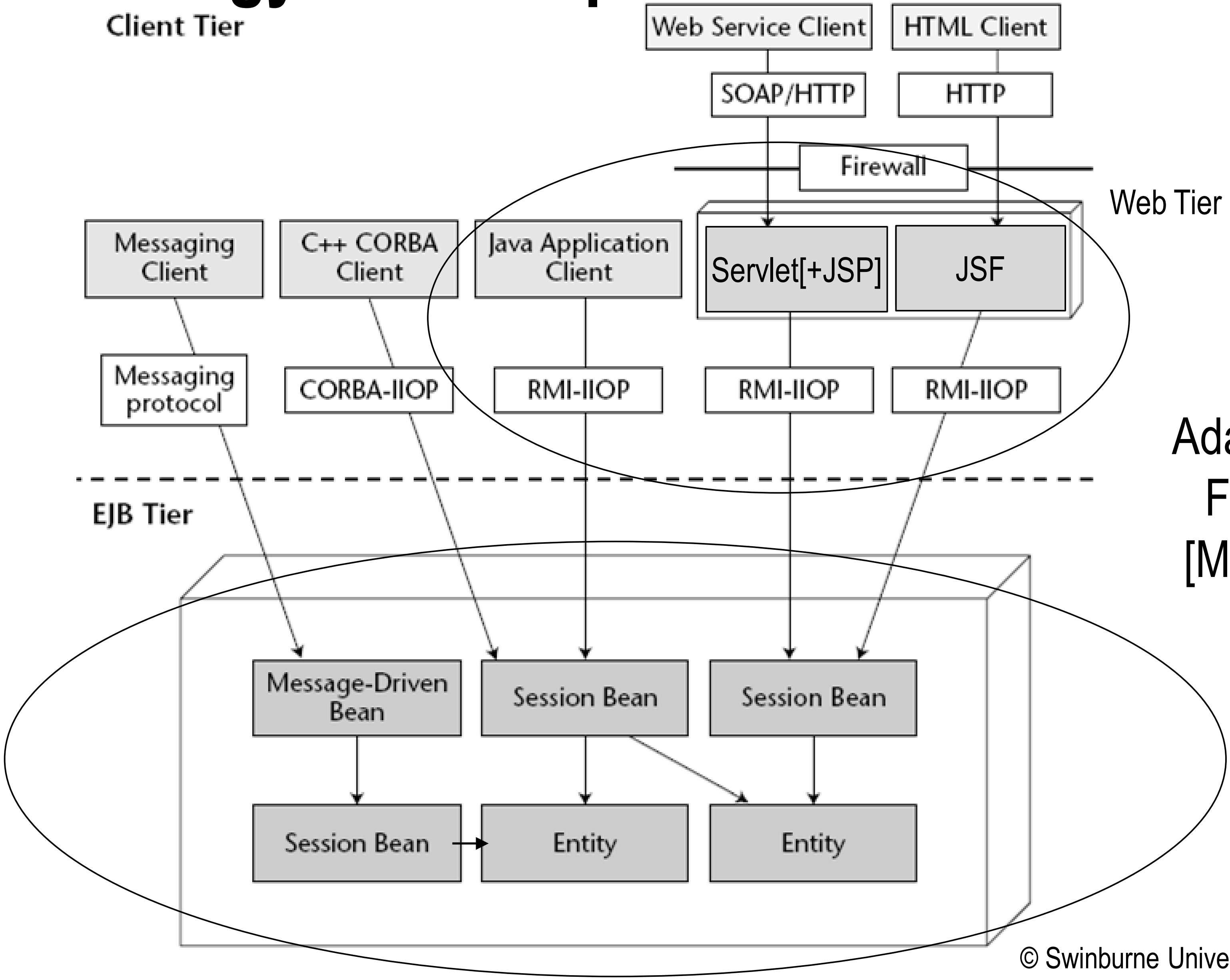
- Have "standard solutions" to solve frequently occurred problems
- Have been used by many developers
- Have been proven to be effective or less error-prone

Analogy 1 – Architecture Framework



Note: Adapted from Fig.1-1 from [JEE6T]

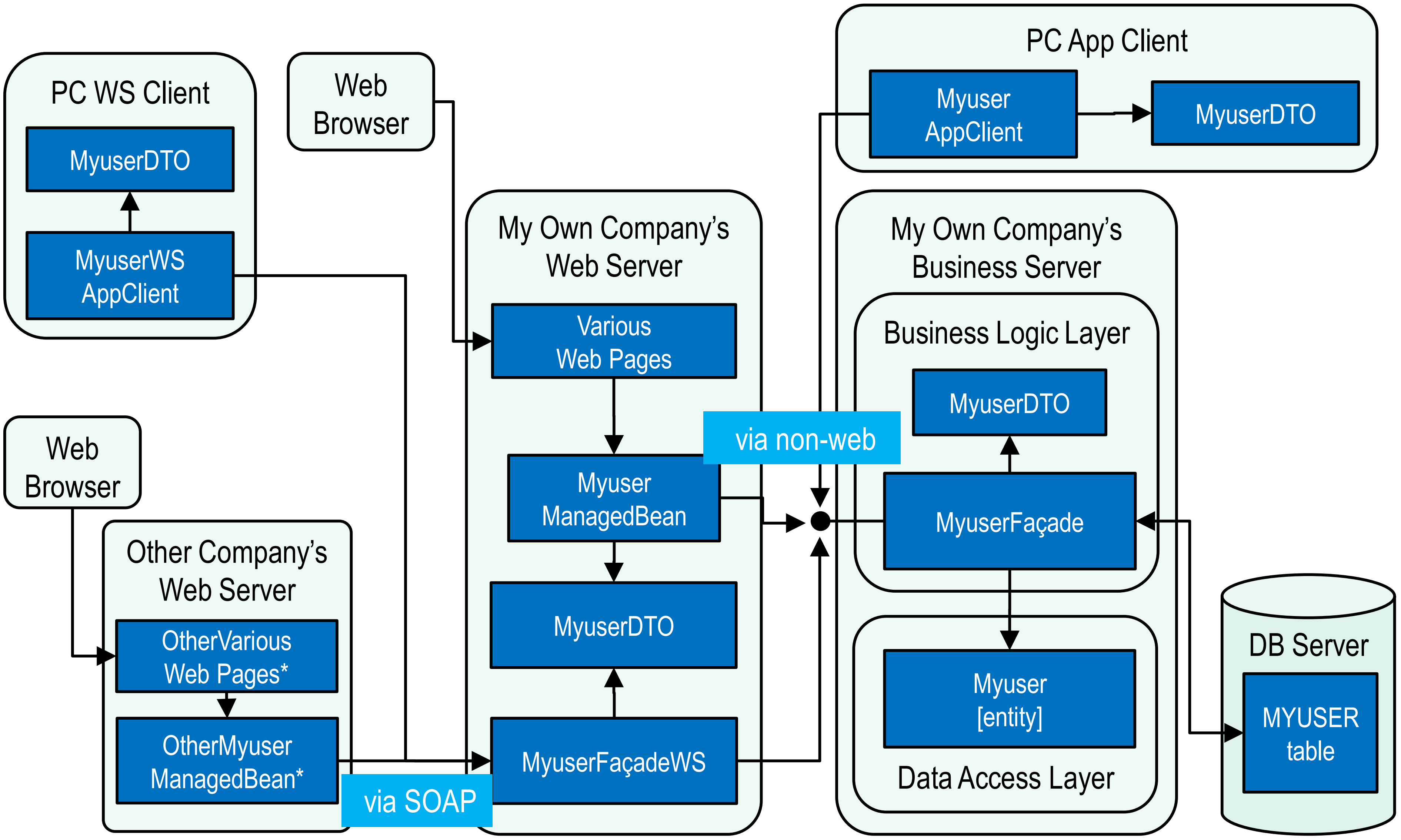
Analogy 2 – Enterprise Architecture



Adapted from Fig. 3.2 of [MEJB,p.63]

*Lab 09 (to come)

ED-JEE-DTO[+WS] Project [Labs 4,5 and 9*]



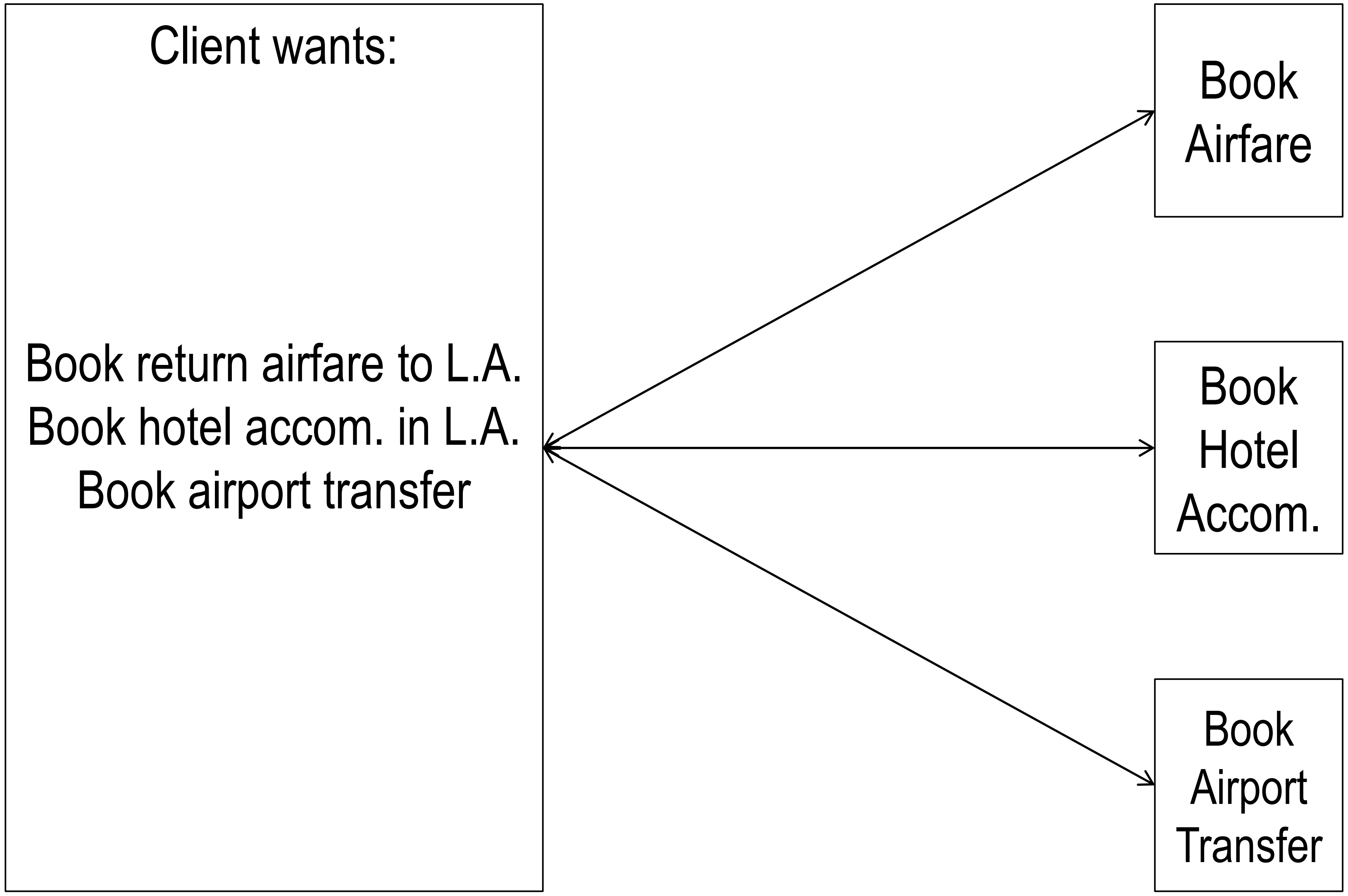
Roadmap

- Architectural Pattern
- **Façade Pattern**
  - Session Facade
  - Web Services Façade
  - Message Facade

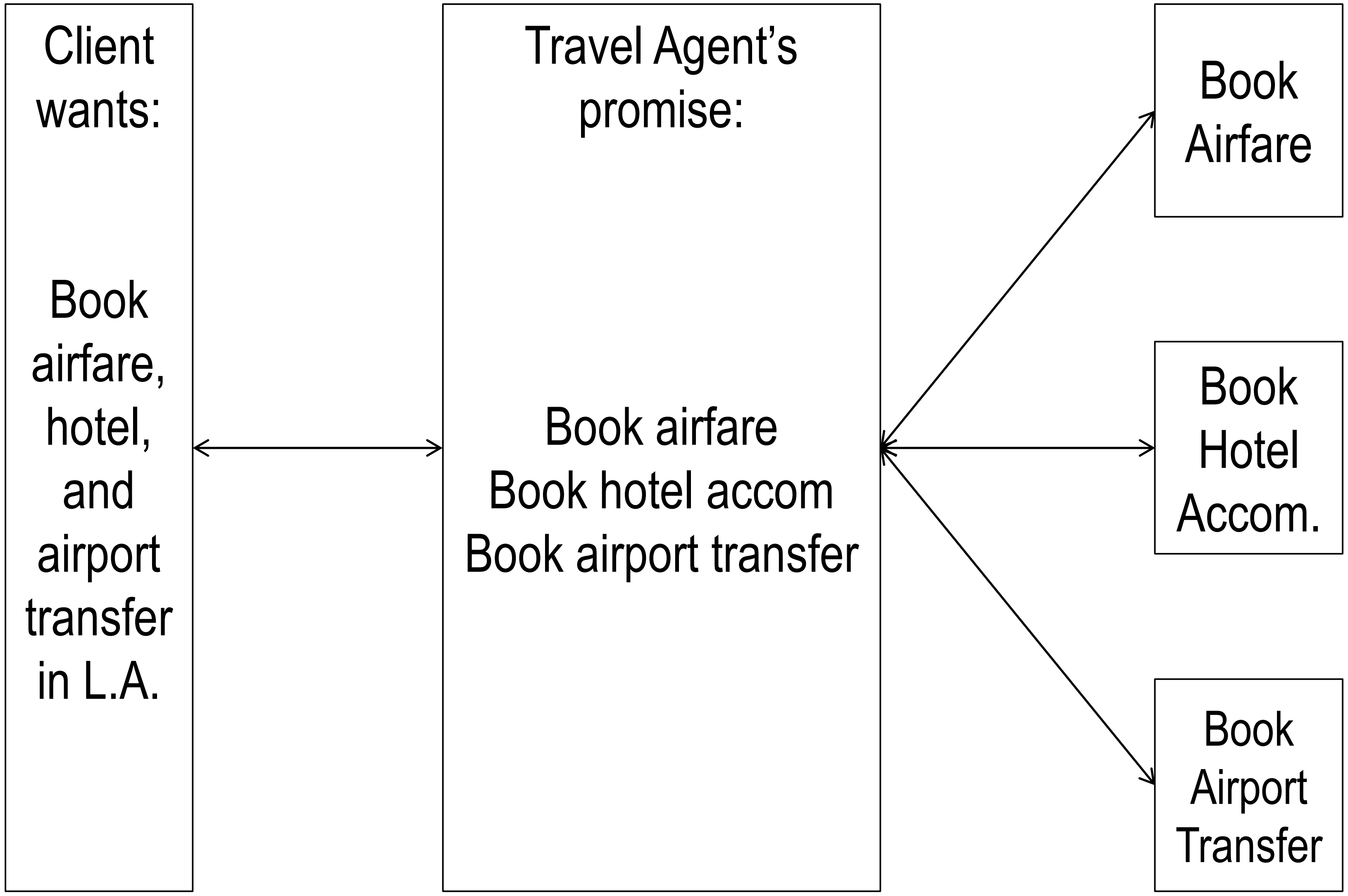
Façade

- Provide a unified interface to a set of interfaces of a subsystem
  - Usually for backend processing
- Usually: provide a higher-level interface that makes the subsystem easier to use
  - … backend processing hidden from other developers

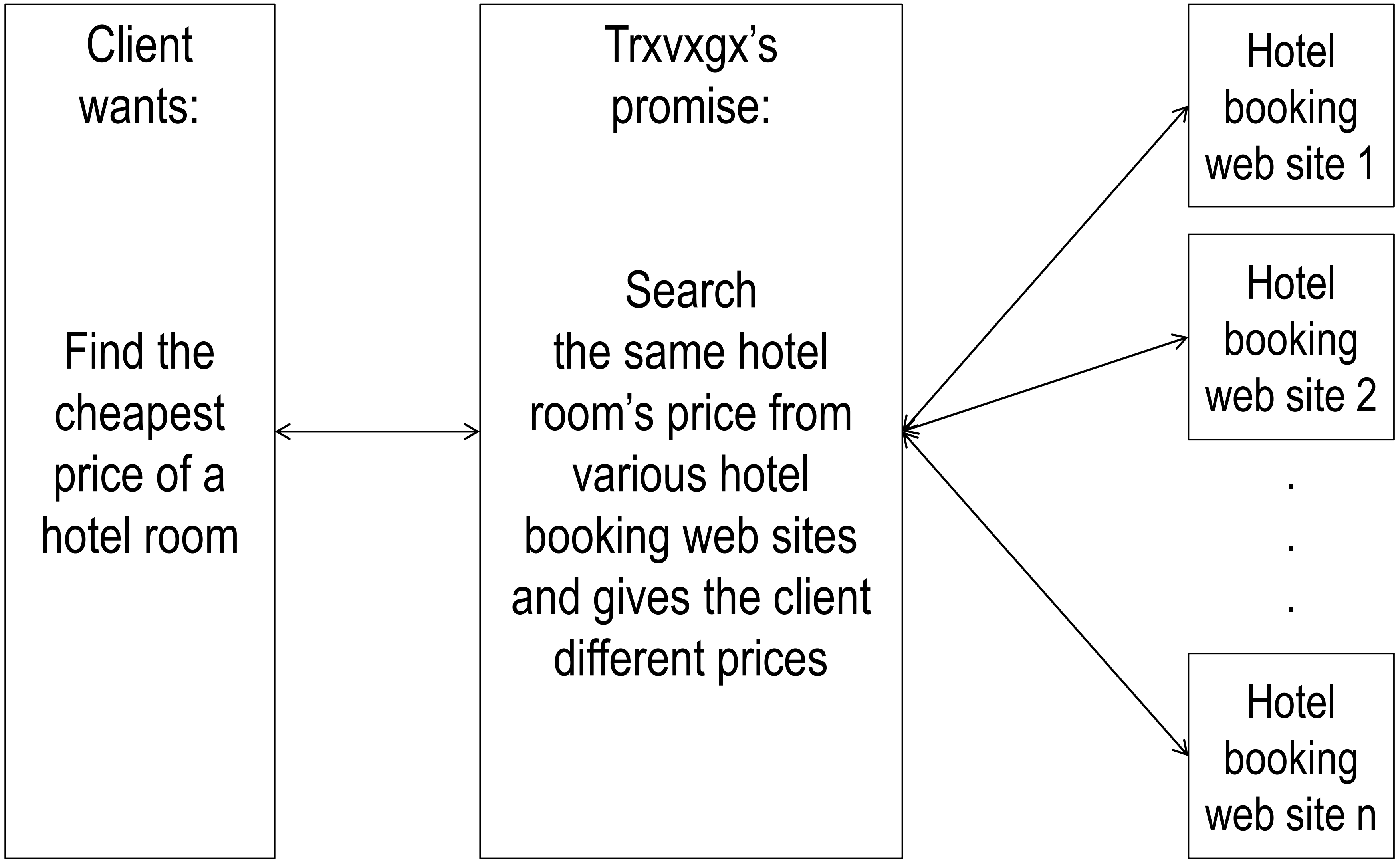
Façade – Analogy



Façade – Analogy (cont'd)



"Hotel – Trxvxgx"


Client wants:
Find the cheapest price of a hotel room
Trxvxgx's promise:
Search the same hotel room's price from various hotel booking web sites and gives the client different prices
Hotel booking web site 1
Hotel booking web site 2
.
.
.
Hotel booking web site n

Session Façade

Problem

- How can an enterprise client execute a use case’s business logic in one transaction and one bulk network call?

[**Related question**]

- Where should we put the business logic?

Session Façade (cont’d)

Answer: Client?

- High network overhead – load data from remote DB server
- Poor concurrency – client is too far from server / network slow
- High coupling – entity class tied to client
- Poor reusability – business logic in client
- Poor maintainability – transaction API interlaced with application logic
- Poor separation of development roles – presentation and business logic integrated together

Session Façade (cont'd)

- Answer: Server? DAL? DAO?
  - The DAO carries more business logics than necessary
  - Difficult to maintain

Session Façade (cont'd)

Solution

- Wrap the DAOs in DAL using business objects in BLL?
  - Java EE – Use Session EJB to access Entity Class
- Note: Client should have access only to the business objects in the BLL
  - [Java EE] Put the business logic in session beans (stateless, stateful or even singleton)
    - The session bean acts as an intermediary and buffers calls to the entity objects

Session Façade – Benefits

- Low network overhead
- Transactional integrity
- Low coupling
- Good reusability
- Good maintainability
- Clean and strict separation of development roles
    - Business vs presentation layer
- A clean separation of business logic from domain logic
- Hide the data object from client

Session Façade – DOs and DON'Ts

DOs

- Group use cases with similar functions into one business object
- Rely on the entity class to update the corresponding data
- Create additional business objects for common business logic

DON'Ts

- Never create a business object GOD-class
- Never put domain logic in business logic
- Never duplicate business logic across the session façade

Other Similar Façade Patterns

Web Services Façade

- Interface with web services
- Use web services objects as front-end ("Big" or RESTful)
  - ☐ Extract the information in the web services call
    - ☐ "Big" using SOAP / HTTP
    - ☐ RESTful using XML / JSON
  - ☐ Call the required business objects to perform the business process

Message Façade

- Interface with message objects
- Use message objects as front-end (MDB in JavaEE)
  - ☐ Extract the message content
  - ☐ Call the required business objects to perform the business process
- Used in asynchronous communications

References

- [EJBDP] F. Marinescu (2002) *EJB Design Patterns – Advanced Patterns, Processes, and Idioms*, 2nd ed., John Wiley & Sons – Chapter 1
  - A free ebook, can be download from www.theServerSide.com
- [MEJB] R.P. Sriganesh, G. Brose, M. Silverman (2008) *Mastering Enterprise JavaBeans 3.0*, 4th ed., John Wiley & Sons – Chapter 3

References

- Java EE Tutorial