Chapter 3
Façade Design Pattern

Concepts and Techniques

H.S. Sarjoughian

CSE 460: Software Analysis and Design

School of Computing, Informatics and Decision Systems Engineering
Fulton Schools of Engineering

Arizona State University, Tempe, AZ, USA

Design Patterns

A Design Pattern offers a generic solution to a recurring problem from which for a specific problem, a specialized solution can be derived.

> "A Design Pattern provides a scheme for **refining** the subsystems or components of a software system, or the relationships between them. It describes a **commonly-recurring structure** of communicating components that solves a **general design problem** within **a particular context**" [GoF, 1995]

A design pattern *implicitly promises* that (1) it can satisfy customer's needs and (2) the solution is feasible.

A Conceptual Roadmap to Software Architecture

Software Architecture Model

Design Pattern Models

typing | concurrency | persistence

abstraction | **hierarchy**

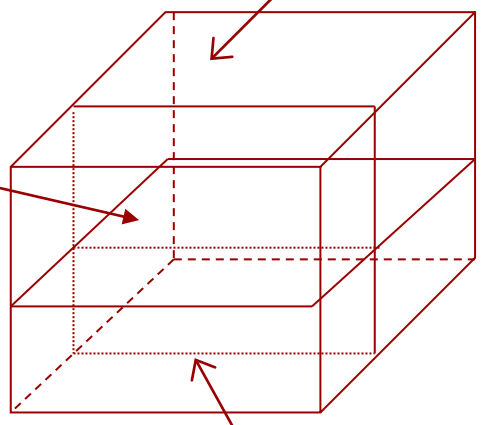
encapsulation | **modularity**

Object Model

UML Model

Logical/Dynamic:
interaction, … diagrams

Logical/Static:
Class, … diagrams



Physical/Static:
Component, … diagrams

Design Pattern Space

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	▪ Factory Method	▪ Adapter (class)	▪ Interpreter ▪ Template Method
	Object	▪ Abstract Factory ▪ Builder ▪ Prototype ▪ ***Singleton***	▪ Adapter (object) ▪ Bridge ▪ Composite ▪ Decorator ▪ ***Façade*** ▪ Flyweight ▪ Proxy	▪ Chain of responsibility ▪ Command ▪ Iterator ▪ Mediator ▪ Memento ▪ ***Observer*** ▪ State ▪ Strategy ▪ Visitor

source: GoF, 1994

Describing a Pattern: Façade

- **Intent**
  - provide a unified interface to a set of interfaces in a subsystem. It defines higher-level interfaces where it is useful to expose only a select set of the subsystem's interfaces
- **Motivation**
  - minimize (direct) dependencies among subsystems
  - expose only a subset of what may be seen and used by others
- **Applicability**
  - want to provide a simplified interface to a complex subsystem
  - want to layer the subsystem
  - want to support reuse

Describing a Pattern: Façade (Cont.)

Participants

- Façade
  - Knows which subsystem classes are responsible for requests
  - Delegates client requests to appropriate subsystem objects
- Subsystem classes
  - provide implementation of the subsystem functionality
  - handle work assigned by the Façade object
  - do not have knowledge of the Façade – classes do not have references to the Façade

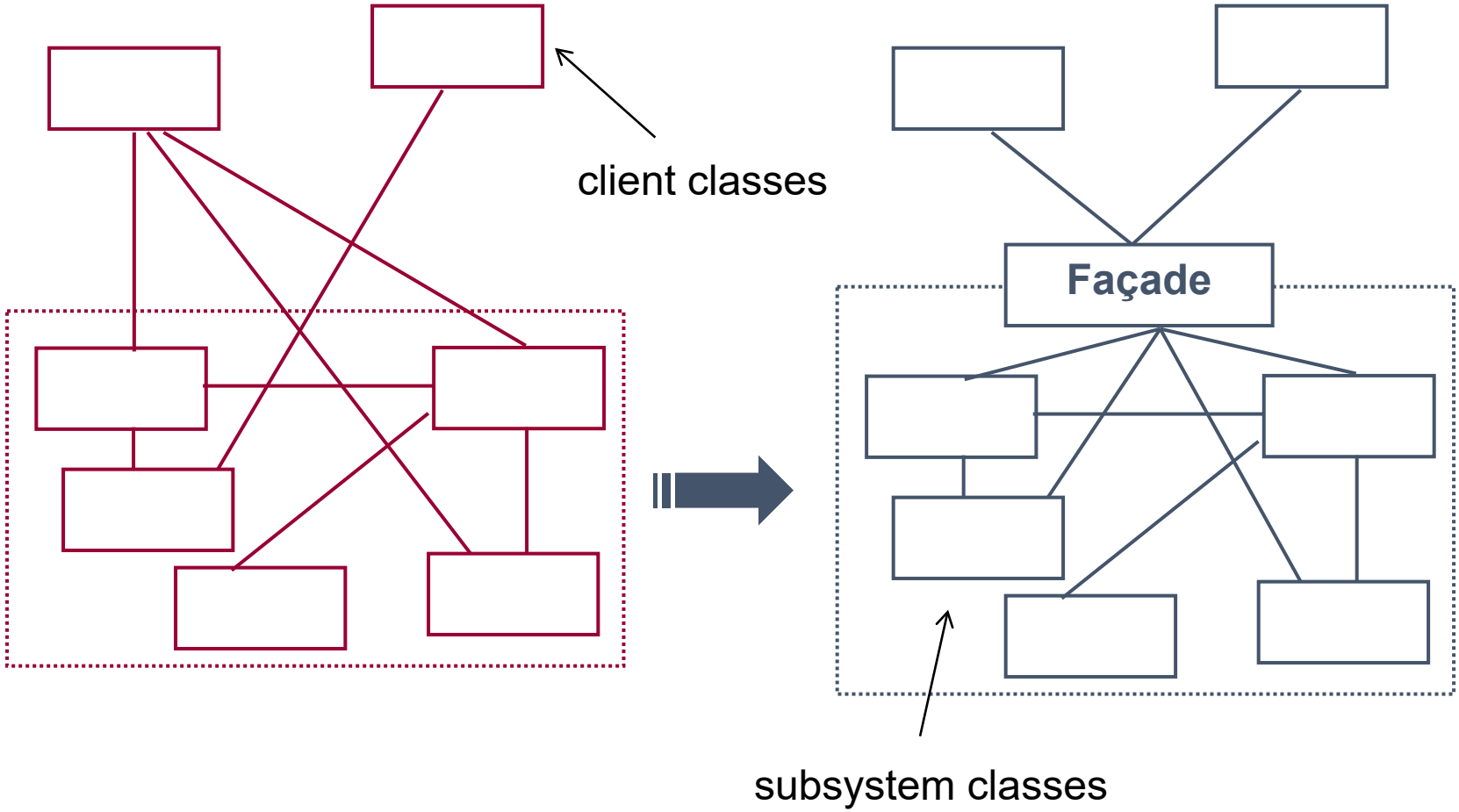
Describing a Pattern: Façade (Cont.)

Collaborations

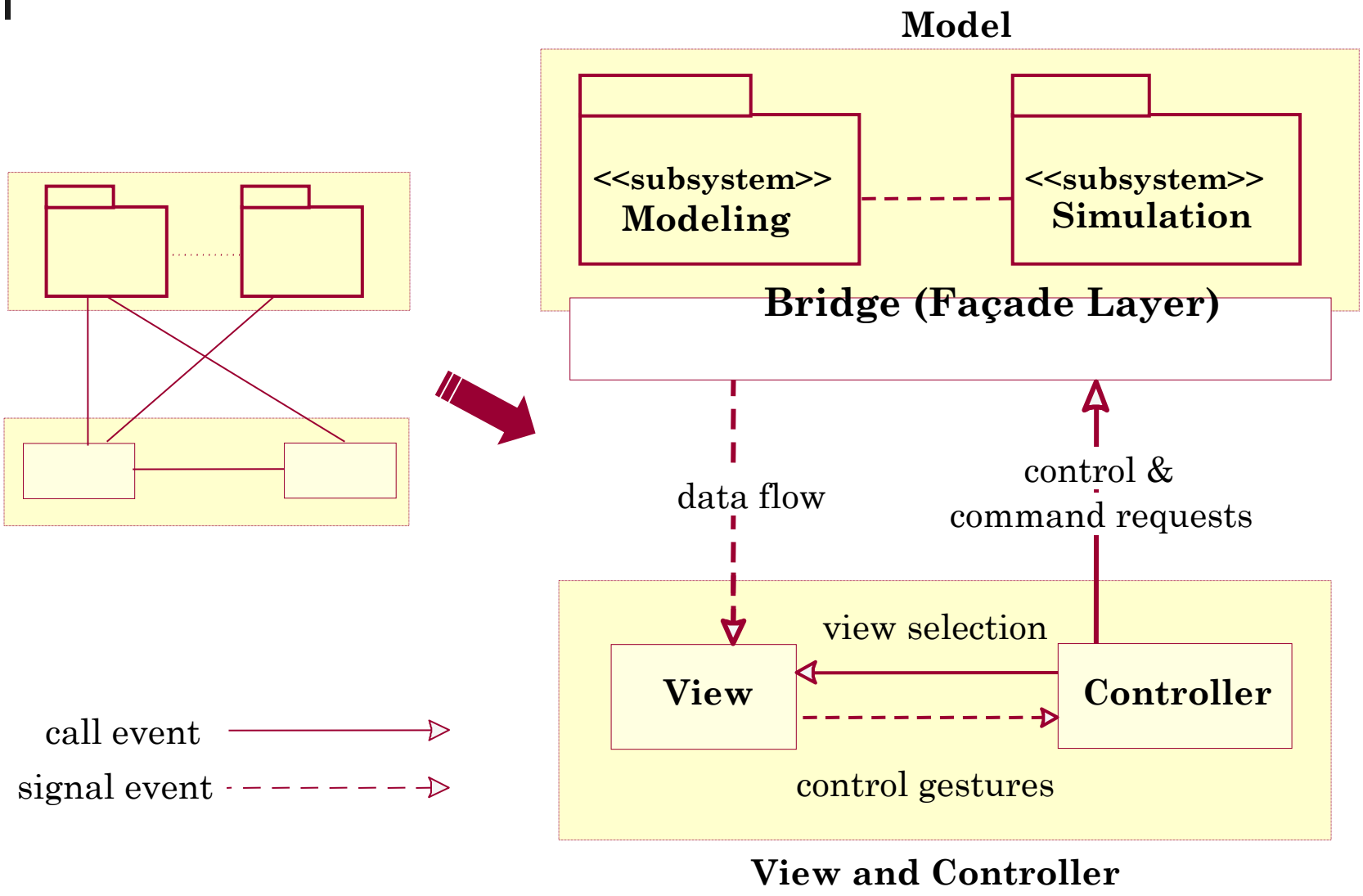
- Clients communicate with the subsystem by sending requests to façade which forwards them to the appropriate subsystem objects
- Façade may need to do some work on its own
- Clients that use the façade are not required to access its subsystem objects directly

Describing a Pattern: Façade (Cont.)

Structure



A Façade Design Pattern Example



Describing A Pattern: Façade (Cont.)

- **Consequences (Benefits)**
  - Shields its clients from subsystem components; it can help significantly the clients to only interact with fewer components of the subsystem – makes the subsystem easier to use
  - Supports weak couplings between the subsystem and its clients – this helps to hide strong couplings that may exist among the subsystem components.
    - helps to change what is “inside” a subsystem without affecting its clients
    - provides a thin layer for a complex subsystem thus supporting independent development (or refinement) of a subsystem
    - It does not prevent clients (or other subsystems) from using its classes directly if there is a need
- **Related patterns**
  - Abstract Factory
  - Mediator

Summary

- Design Patterns can provide quick help in solving many design problems – a design pattern support one or more software quality attributes (e.g., modifiability and performance)
- A design pattern offers suitable level of abstractions (e.g., choice of objects and their interactions)
- Design patterns complement software architecture design – some levels of details are not suitable for consideration in software architecture design
- There may not necessarily exist any single perfect design pattern
- Design patterns may be necessary in order to solve multiple problems often faced in large-scale designs (different design patterns solve different quality attributes)

References

- *Design Patterns: Elements of Reuseable Object-Oriented Software, E. Gamma, R. Helm, R. Johnson, J. Vlissides, Addison Wesley, 1994. [commonly referred to as GoF]*
- *Pattern-Oriented Software Architecture, Volume 4, A Pattern Language for Distributed Computing, F. Buschmann, K. Henney, D. C. Schmidt, Wiley, 2007*
- *Pattern-Oriented Software Architecture: A System of Patterns, F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, John Wiley & Sons, 1996.*
- *Thinking in Patterns with Java, B. Eckel, Mindview.net, 2002.*
- *astah, UML software tool, astah.net, 2021*