Chapter 2
Observer 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

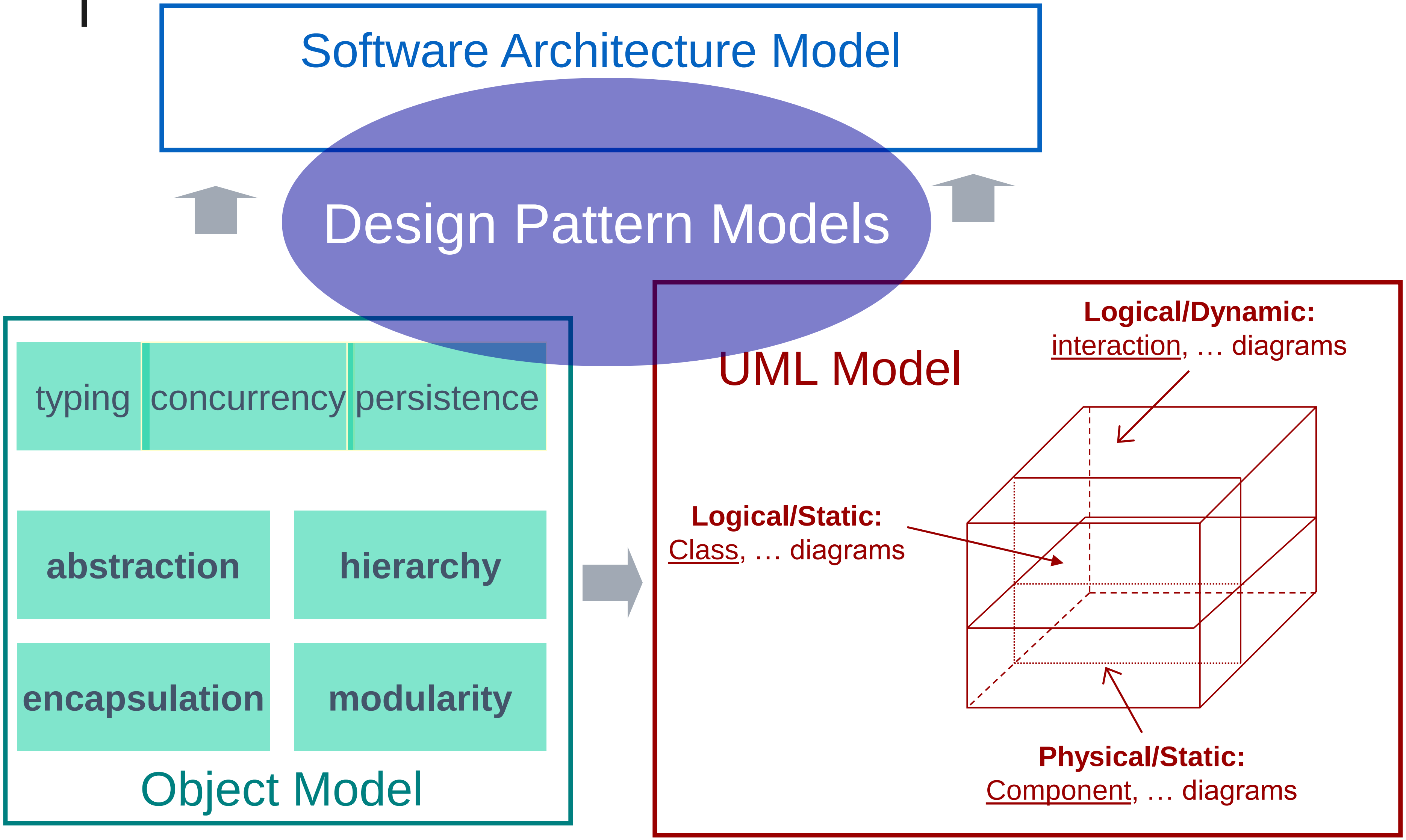
Design Patterns

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

Patterns

- Each Design Pattern presents a concrete solution schema for recurring design problems based on proven solutions
  - *Problem requirements* and *desired properties of a solution* are available
- Each Design Pattern provides concepts and specifications distinct from, but complementary to, those contained in the Object Model, UML, and Software Architectures (often tied to particular programming languages or frameworks)
  - Accounts for quality attributes
- Design Patterns document *what* models to develop and *how to* create them
  - Usually in terms of class, sequence, and other UML diagrams (e.g., see the Observer pattern)

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

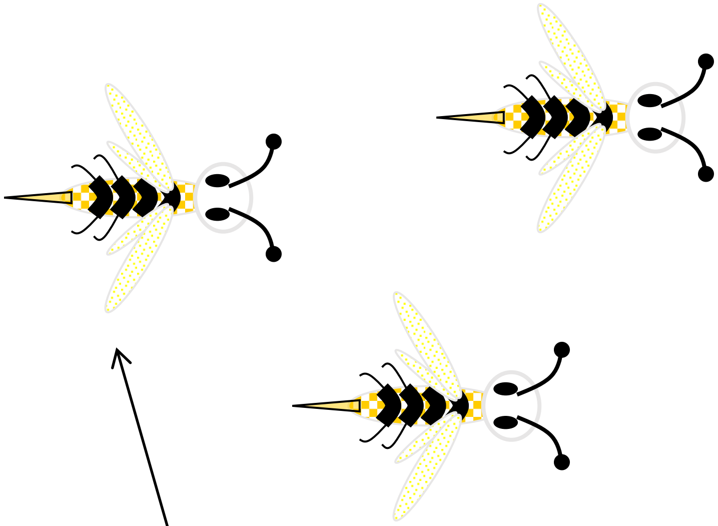
source: GoF, 1994

Observer Pattern

Observers (similar to subscriber)

Subject (similar to publisher)

Honeybees



- start feeding
- stop feeding

Flower

- open
- close

Describing a Pattern: Observer

- **Intent**
  - Define a **one-to-many dependency** between objects so that when one object (Subject) changes state, all its dependents (Observers) are notified and automatically updated.
- **Motivation**
  - Maintain consistency among a collection of cooperating classes.
  - Support loose coupling between what changes (subject) and what is affected (observers).

Describing a Pattern: Observer

Applicability

- When an abstraction has two aspects and one depends on the other. Encapsulation supports independent change in the subject and observer **independently** and thus supports reuse.
- When a change to one object (Subject) requires changing others (Observers) and we do not know how many observer objects need to be changed.
- When an object should be able to notify other objects without making assumptions about who these objects are.

Describing a Pattern: Observer (Cont.)

Participants

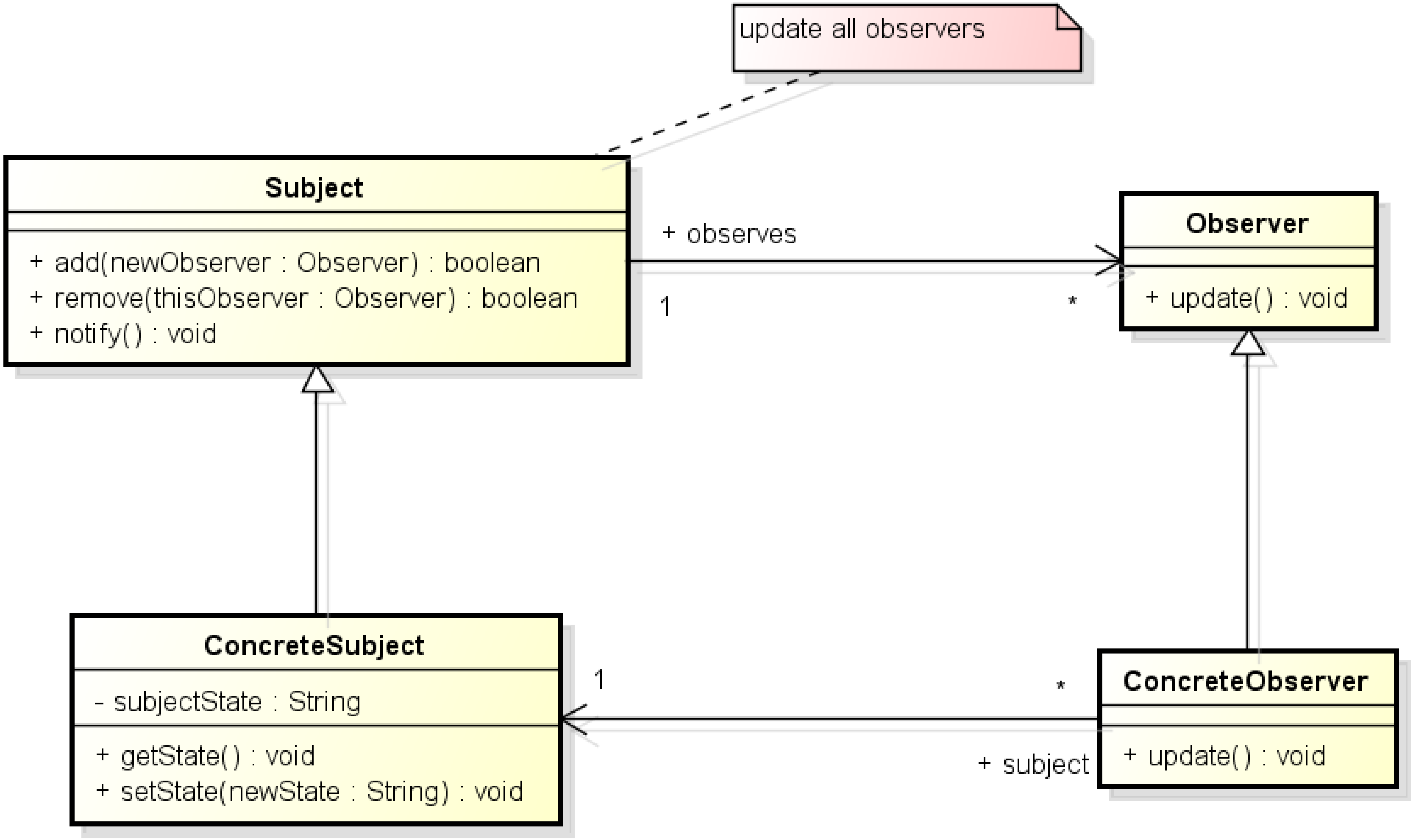
- Subject
    - knows its observers
    - provides an interface for adding/deleting Observer objects
- Observer
    - defines an interface (with an update method) for the Observer objects to be notified when changes occur in the Subject

Describing a Pattern: Observer (Cont.)

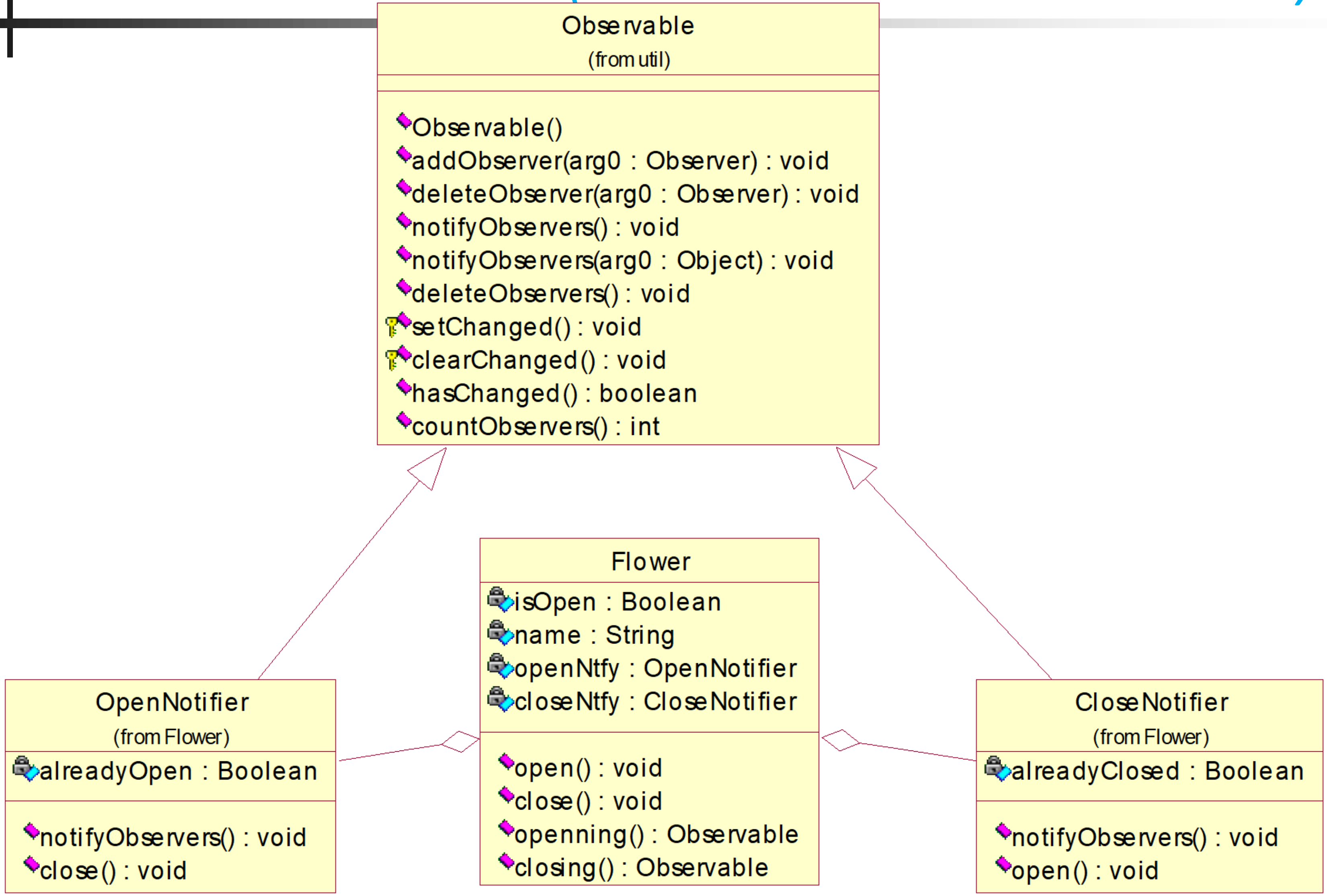
Participants

- **ConcreteSubject**
  - stores state of interest to concreteObserver objects
  - sends a notification to its observers when its state changes
- **ConcreteObserver**
  - maintains a reference to a ConcreteSubject object
  - stores state that should stay consistent with the subject’s
  - implements the Observer interface (update method) to keep its state consistent with the subject’s

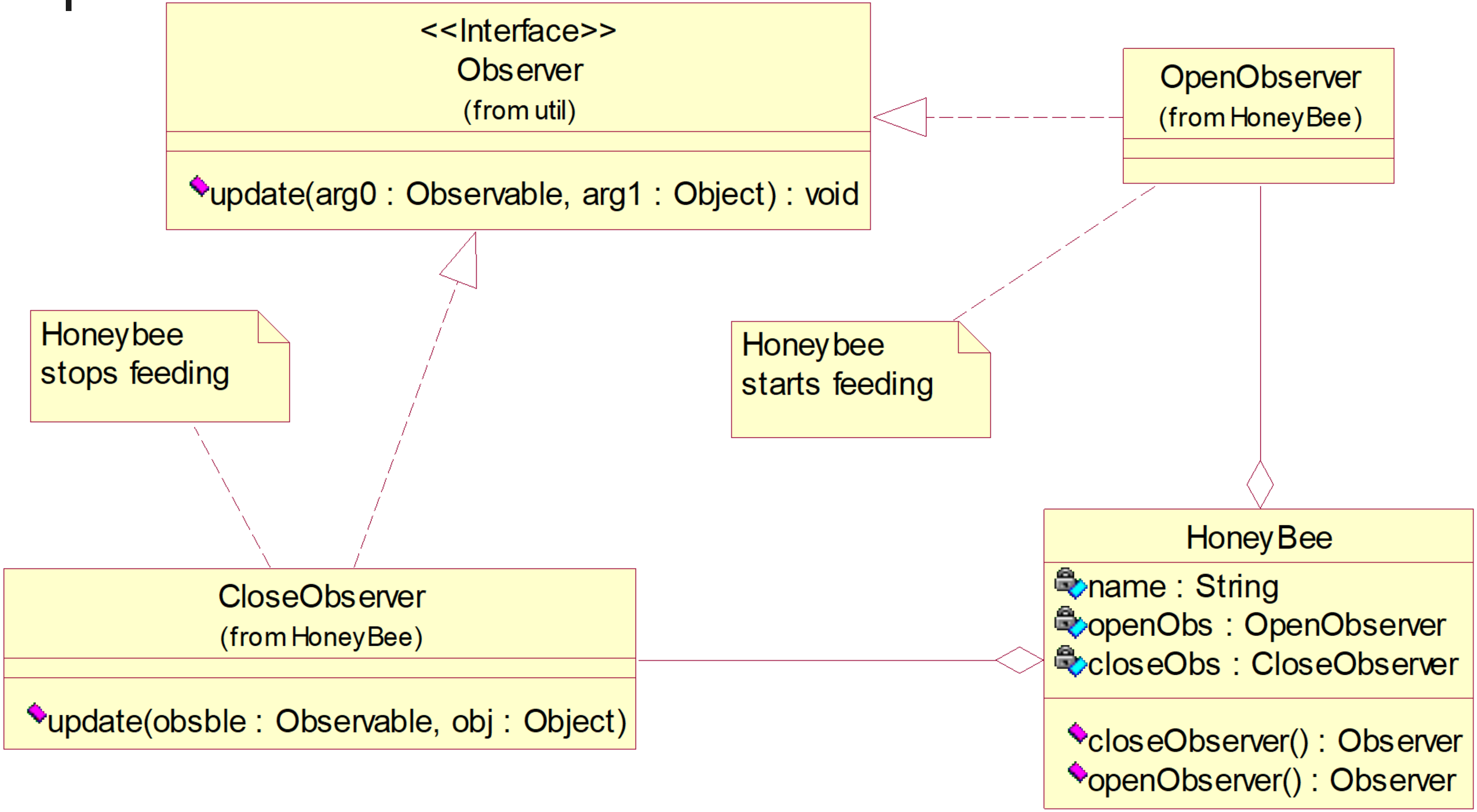
Describing a Pattern: Observer (Cont.)



An Observer Design Pattern Example (Flower is Observable)



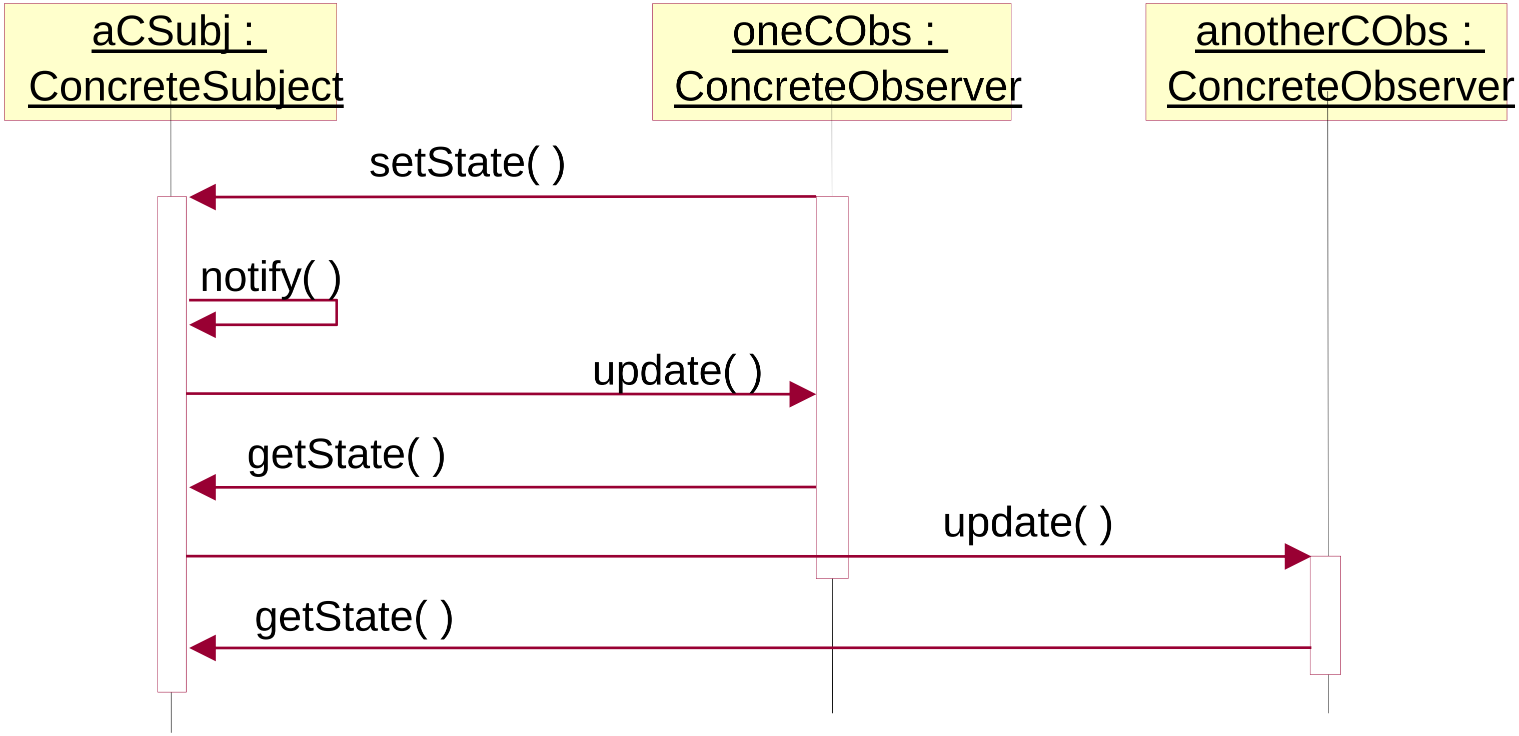
An Observer Design Pattern Example (Honeybee is Observer)



Describing a Pattern: Observer (Cont.)

Collaborations

- ConcreteSubject notifies its observers whenever a change occurs
- ConcreteObserver object may query the subject once it is informed about a change in the concrete subject



Describing A Pattern: Observer (Cont.)

- **Consequences (Benefits)**
  - Abstract coupling between Subject and Observer
    - subject knows only about a list of observers which conform to some defined interface of an abstract Observer class (or Interface)
  - Support for broadcast communication
    - Notification is sent to all interested objects – the subject does not care how many objects are interested in receiving the state update which in turn supports adding/deleting observers dynamically
  - Unexpected updates
    - Simple update protocol does not provide sufficient information about what is changed on the Subject, thus may require observers to discover what may have changed
- **Related patterns**
  - Mediator
  - Singleton

How to Use a Design Pattern

- Read the pattern once through for an overview – Applicability and Consequences are important

- Study the Structure, Participants, and Collaborations sections

- Study the sample code to understand choices from going from design to implementation

- Choose names for pattern participants that are meaningful in for the application at hand (take into account context of the problem)

Classification of Design Patterns

Creational

- Purpose: handle creation of objects – separate the details of object creation and thus help keeping changes local to the objects (e.g., Singleton)

Structural

- Purpose: support design of objects to satisfy particular project constraints – objects are connected in a such a way that changes in the structure does not require changes in the connections (e.g., Façade)

Behavioral

- Purpose: support objects to handle specific types of actions – encapsulate details of processes (e.g., Observer)

How to Use a Design Pattern

- Define the classes including interfaces and other classifiers
- Define application-specific names for operations in the pattern
- Implement the operations to carry out the responsibilities and collaborations in the pattern

Selecting a Design Pattern

- Study show design patterns can solve design problems (design patterns help identify suitable objects that have the right level of granularity and help specify object interfaces)
- Understand design patterns Intent section
- Study how patterns interrelated
- Understand patterns that have similar purposes
- Examine causes of redesign

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.com, 2014*