UML Diagrams

(Based on [Booch *et al.* 2005])

Yih-Kuen Tsay

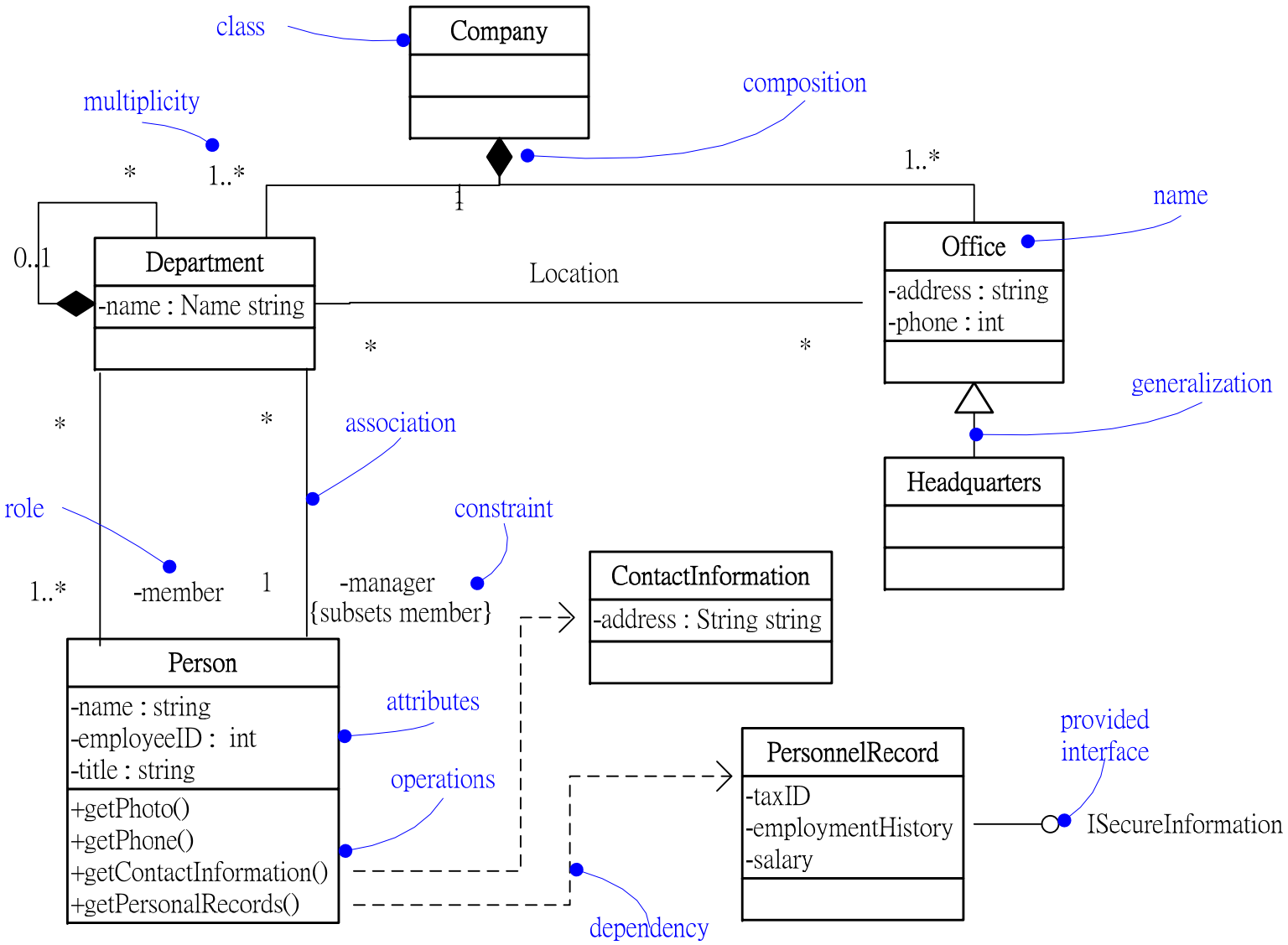
Dept. of Information Management

National Taiwan University

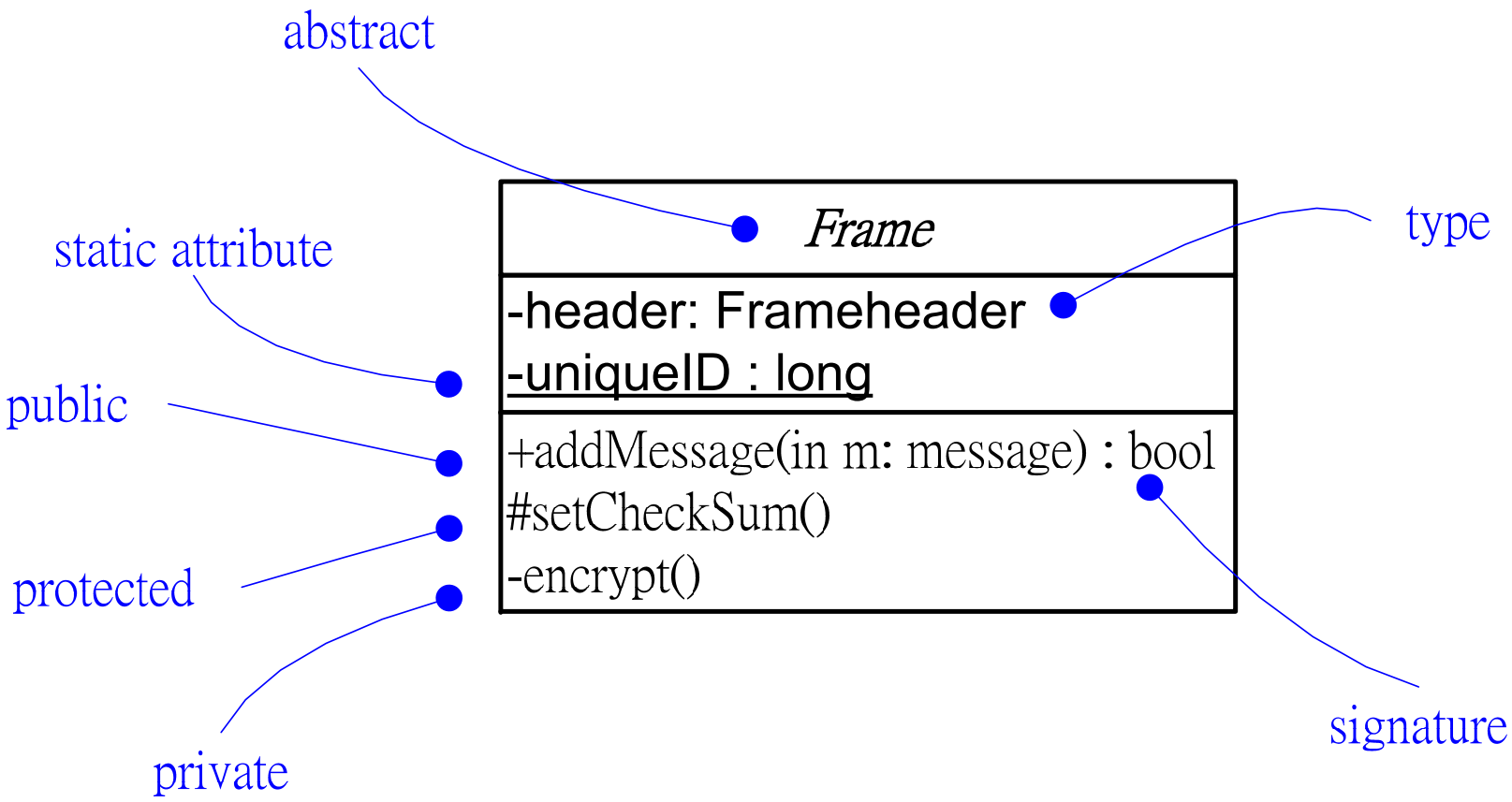
Outline

- Structural Diagrams
- Behavioral Diagrams
- Advanced Structural Modeling
  - Object Diagrams
  - Components
- Advanced Behavioral Modeling
  - State Machines
  - Processes and Threads
  - Timing Constraints

Class Diagram



Advanced Classes

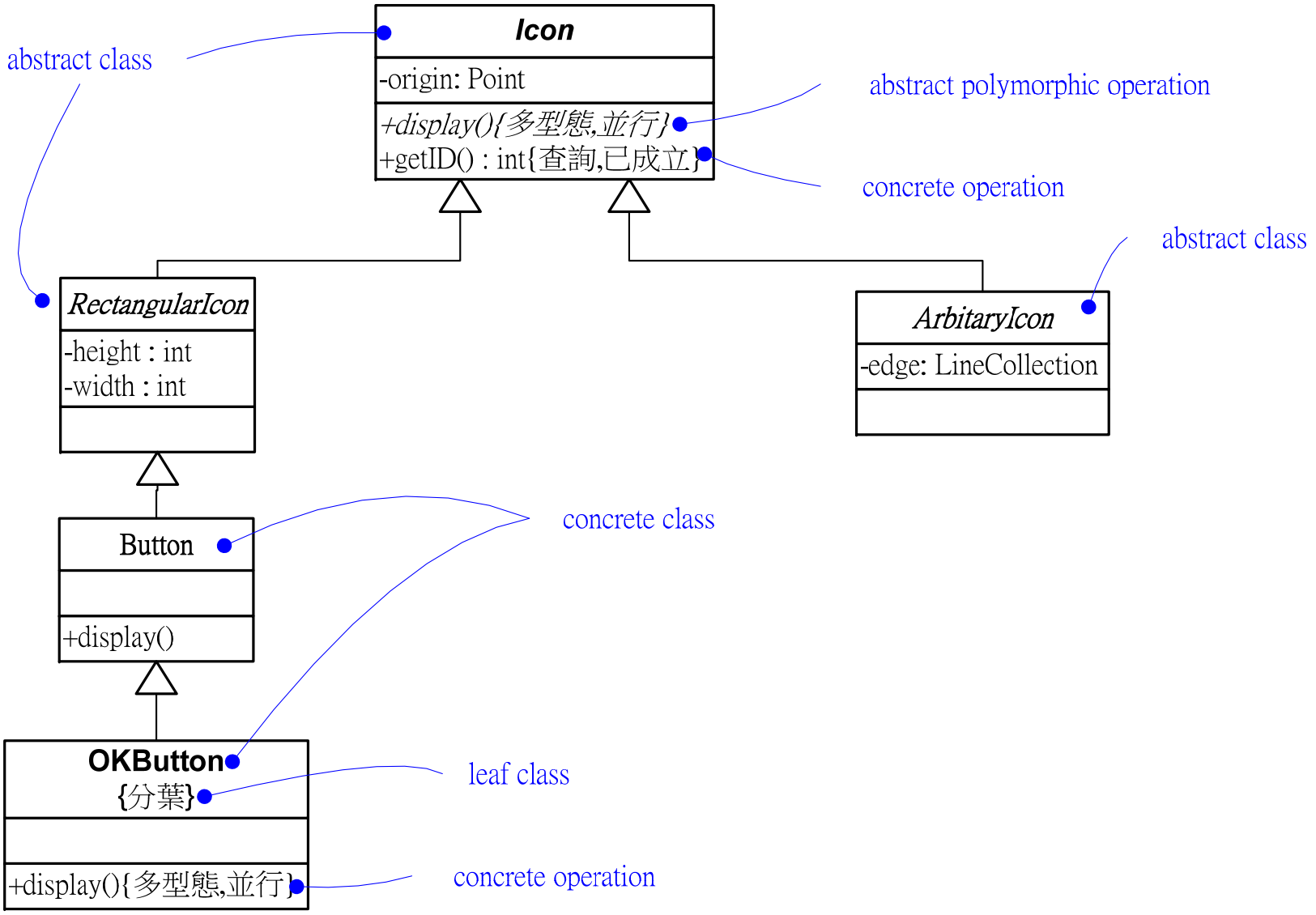


Classifiers

In general, those modeling elements that can have instances are called classifiers.

- Interface
- Datatype
- Signal
- Component
- Node
- Use case
- Subsystem

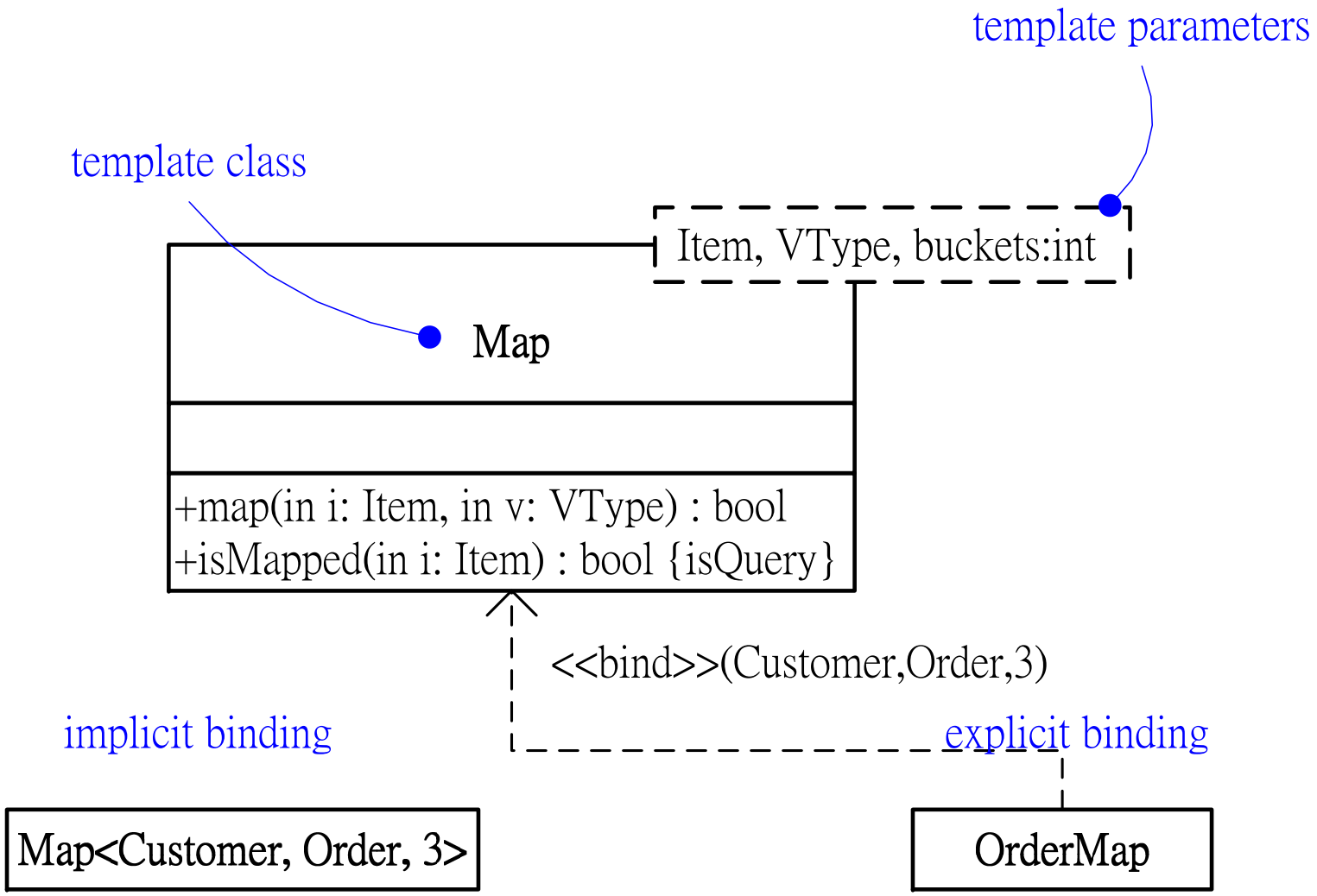
Abstract and Concrete Classes



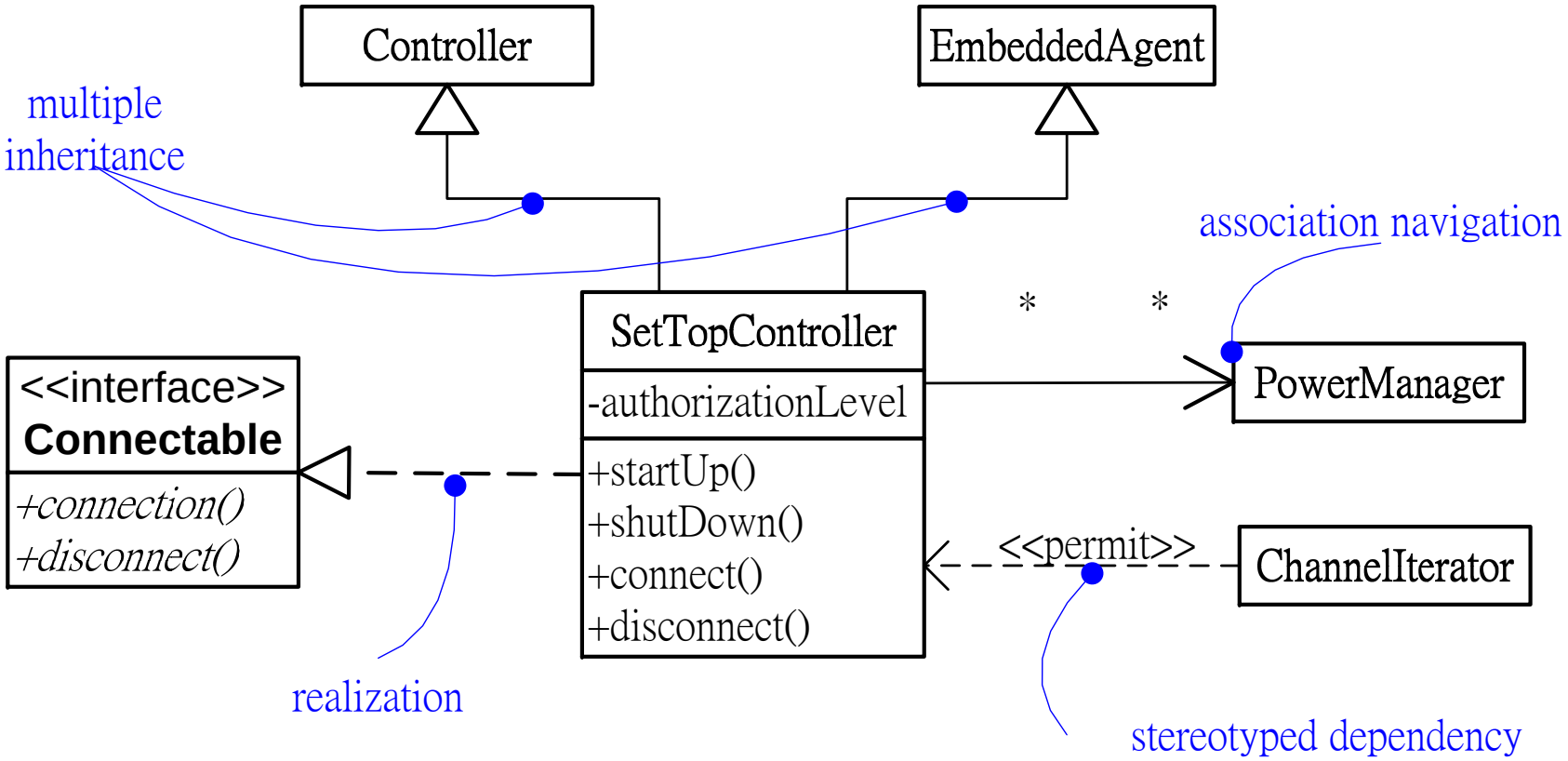
Properties on Operations

- **query**: no side effects
- **sequential**: relying on the callers to do the coordination
- **guarded**: all calls sequentialized (by the object)
- **concurrent**: concurrency control enforced
- **static**: like a global procedure

Template Classes



Advanced Relationships



Advanced Relationships (cont.)

- Stereotypes for dependency
  - Among classes and objects (in class diagrams): **bind, derive, permit (friend), instanceOf, instantiate, powertype, refine, use**
  - Among packages: **import, access**
  - Among use cases: **extend, include**
  - In state machines: **send**
  - In subsystems and models: **trace**

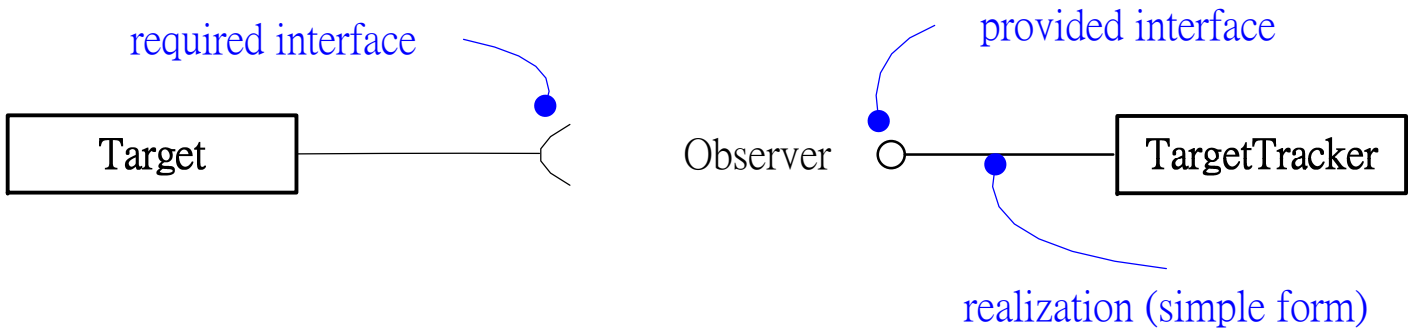
Advanced Relationships (cont.)

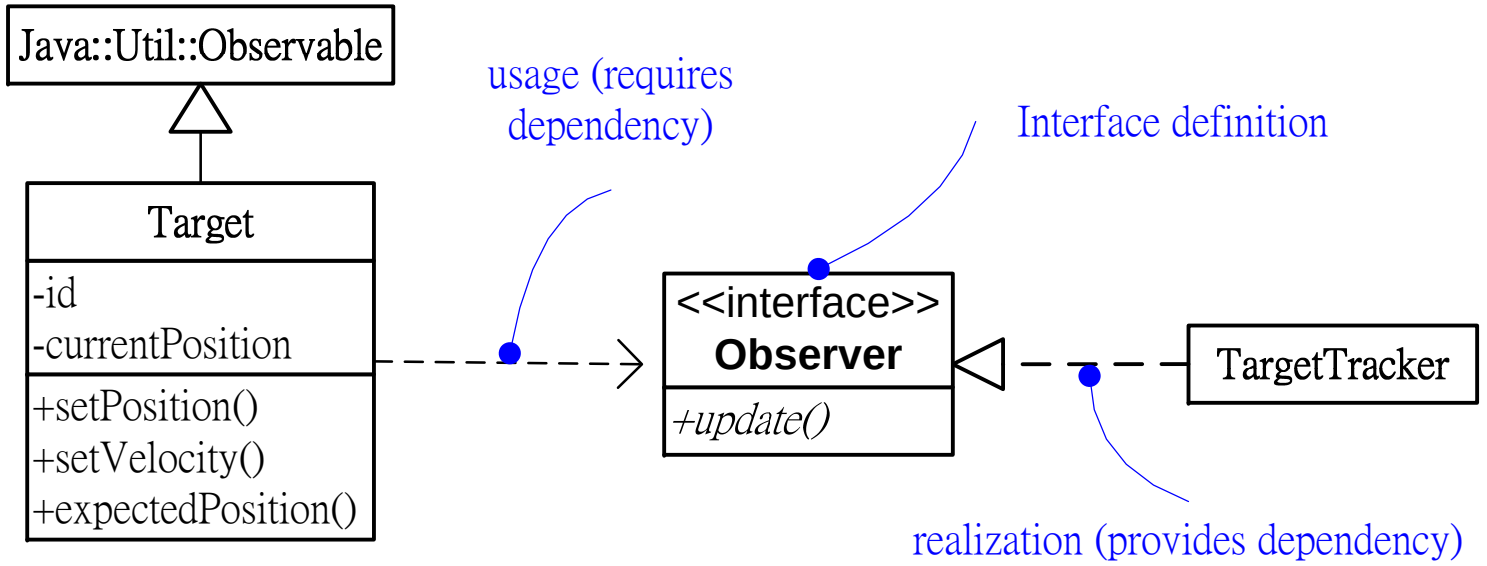
- A stereotype for generalization:
  - **realization**
- Constraints for generalization:
  - **complete**: all children have been specified
  - **incomplete**
  - **disjoint**: subtypes are incompatible
  - **overlapping**

Advanced Relationships (cont.)

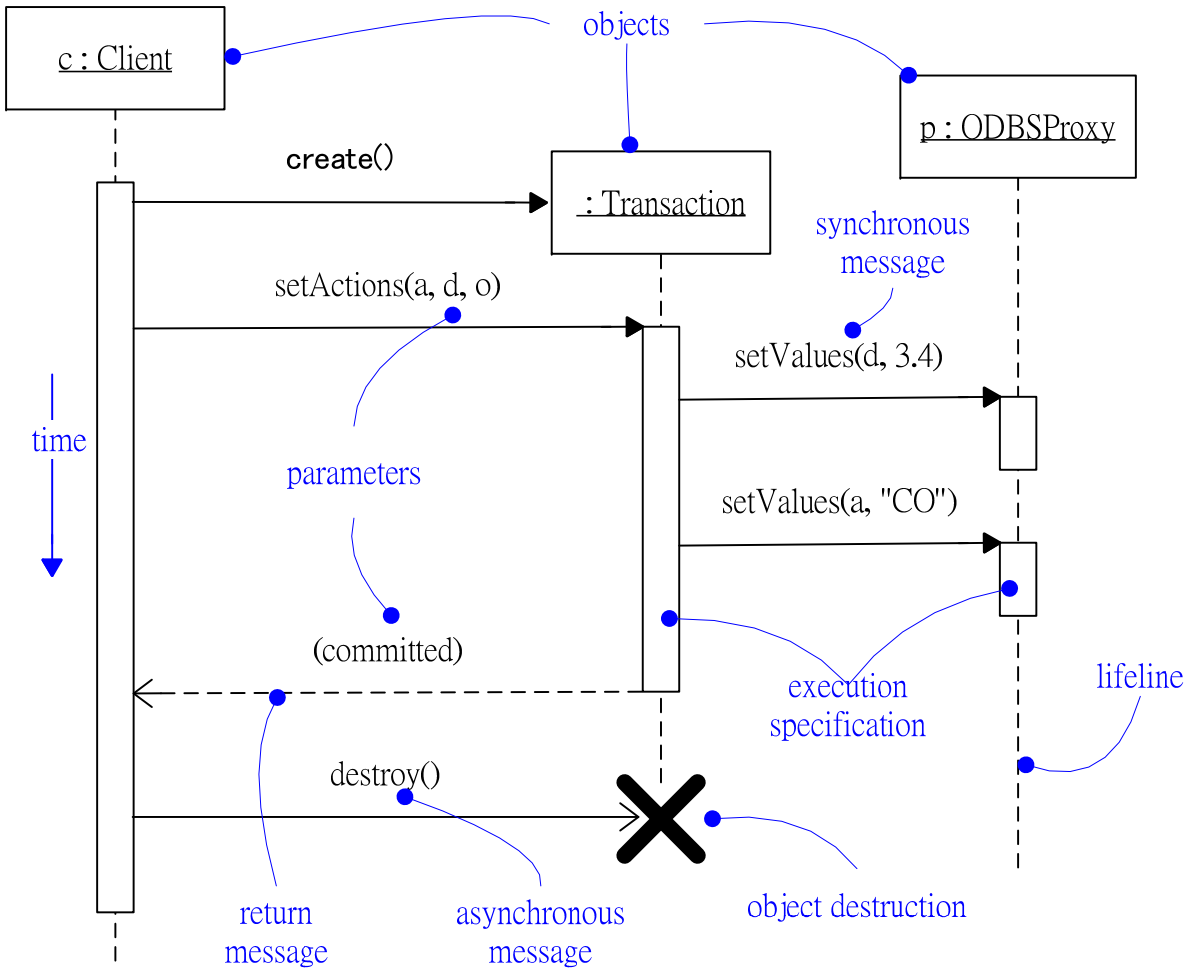
- Properties of association
  - Navigability
  - Visibility
  - Qualification
  - Interface specifier (obsolete in UML2)
  - Composition
  - Association classes
  - Constraints: **ordered, set, bag, ordered set, list** or **sequence, readonly**

Realizations

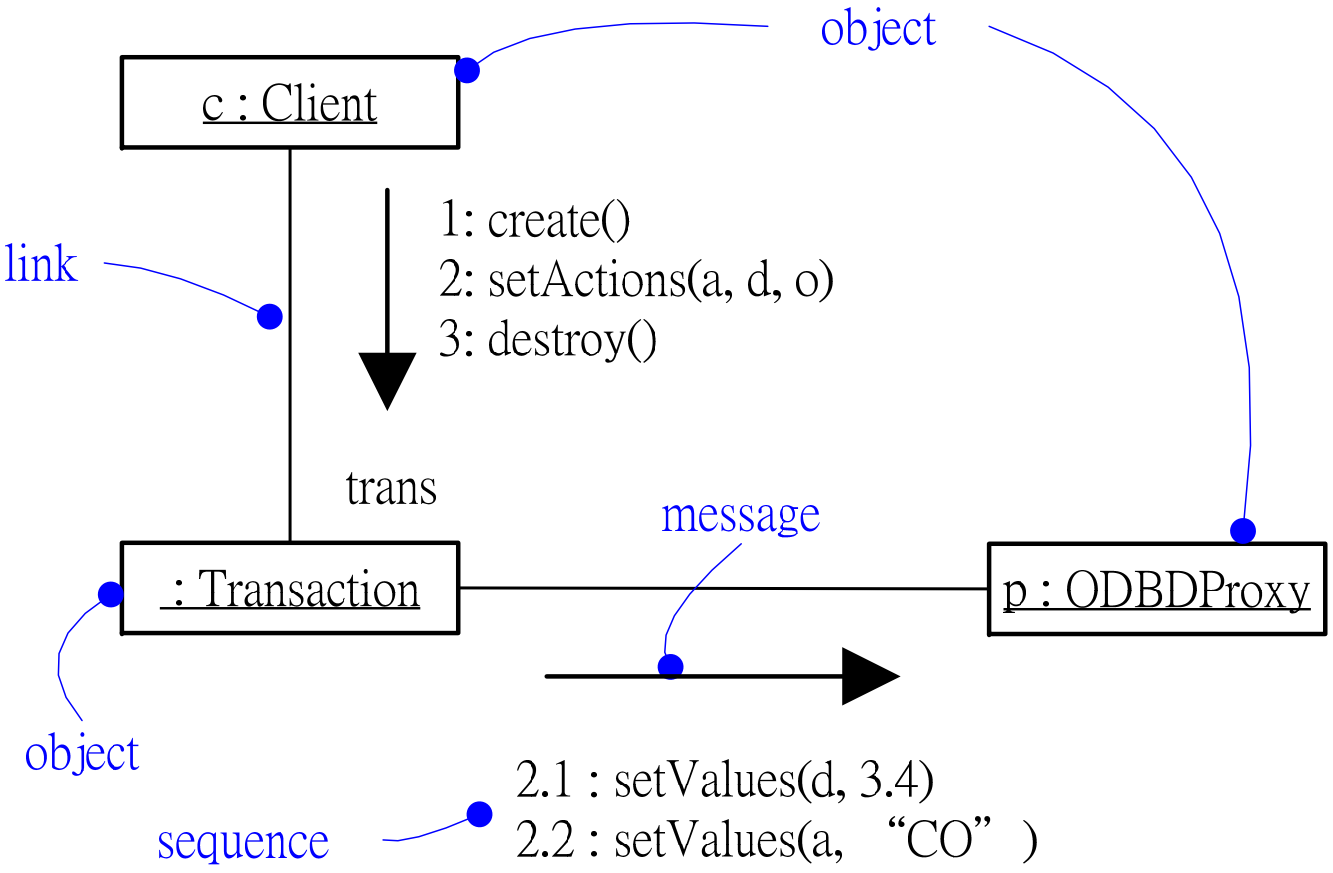




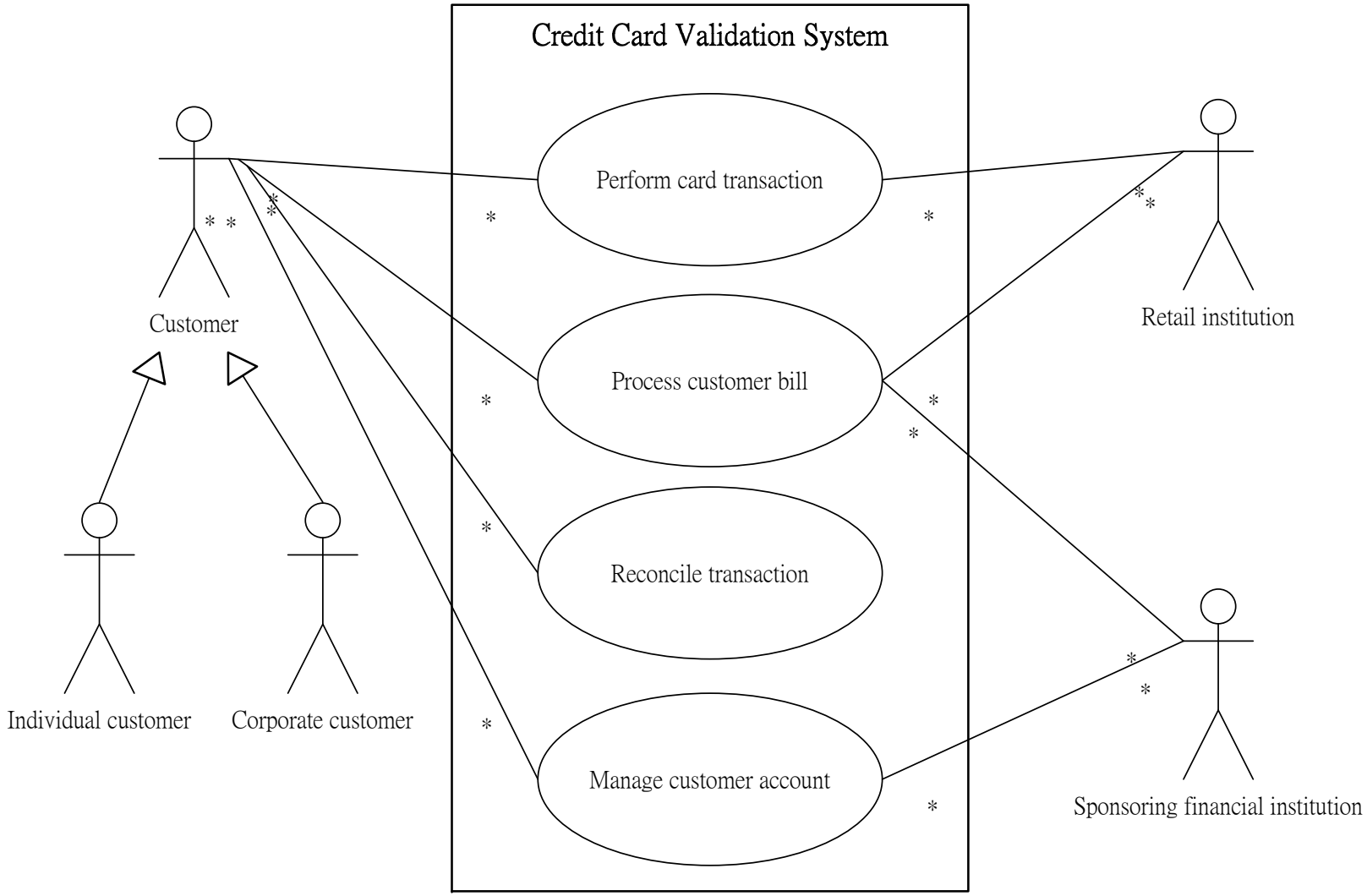
Sequence Diagram



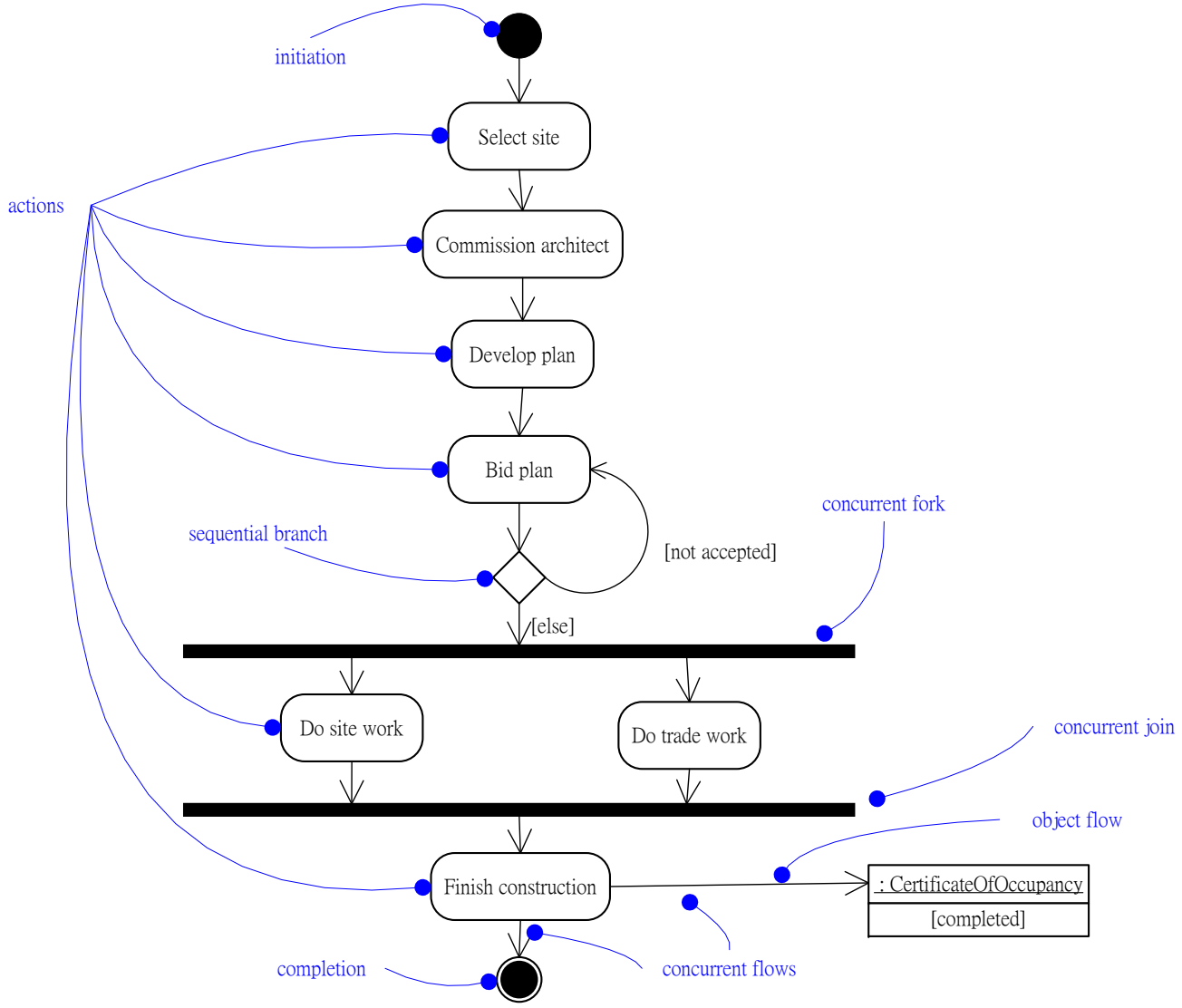
Communication Diagram



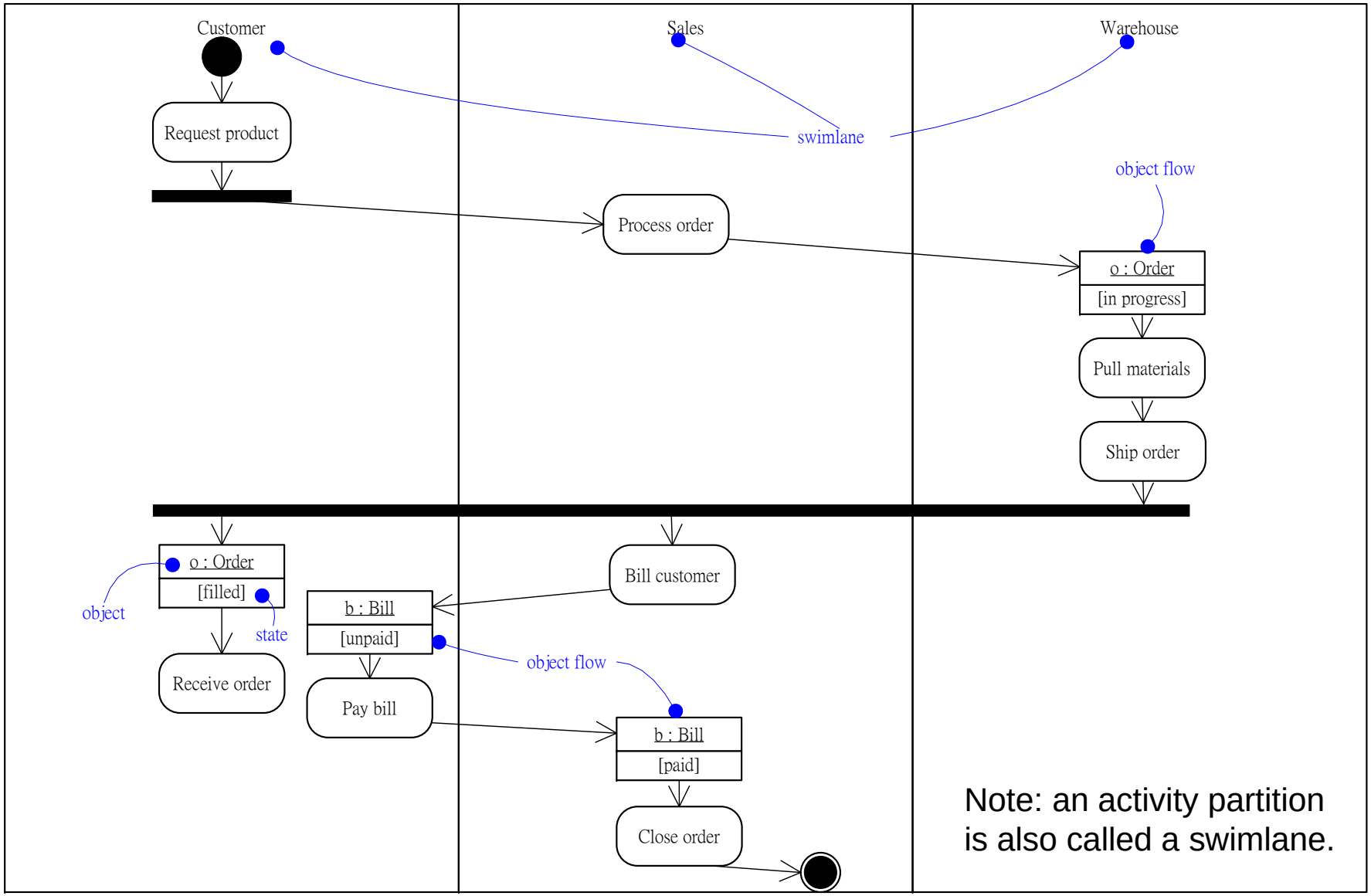
Use Case Diagram



Activity Diagram

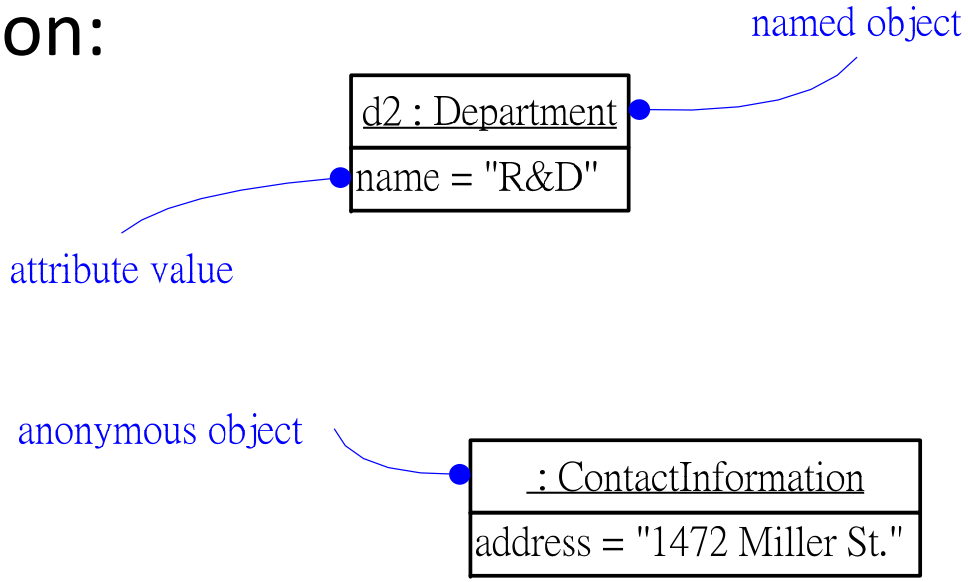


Activity Partitions and Object Flows



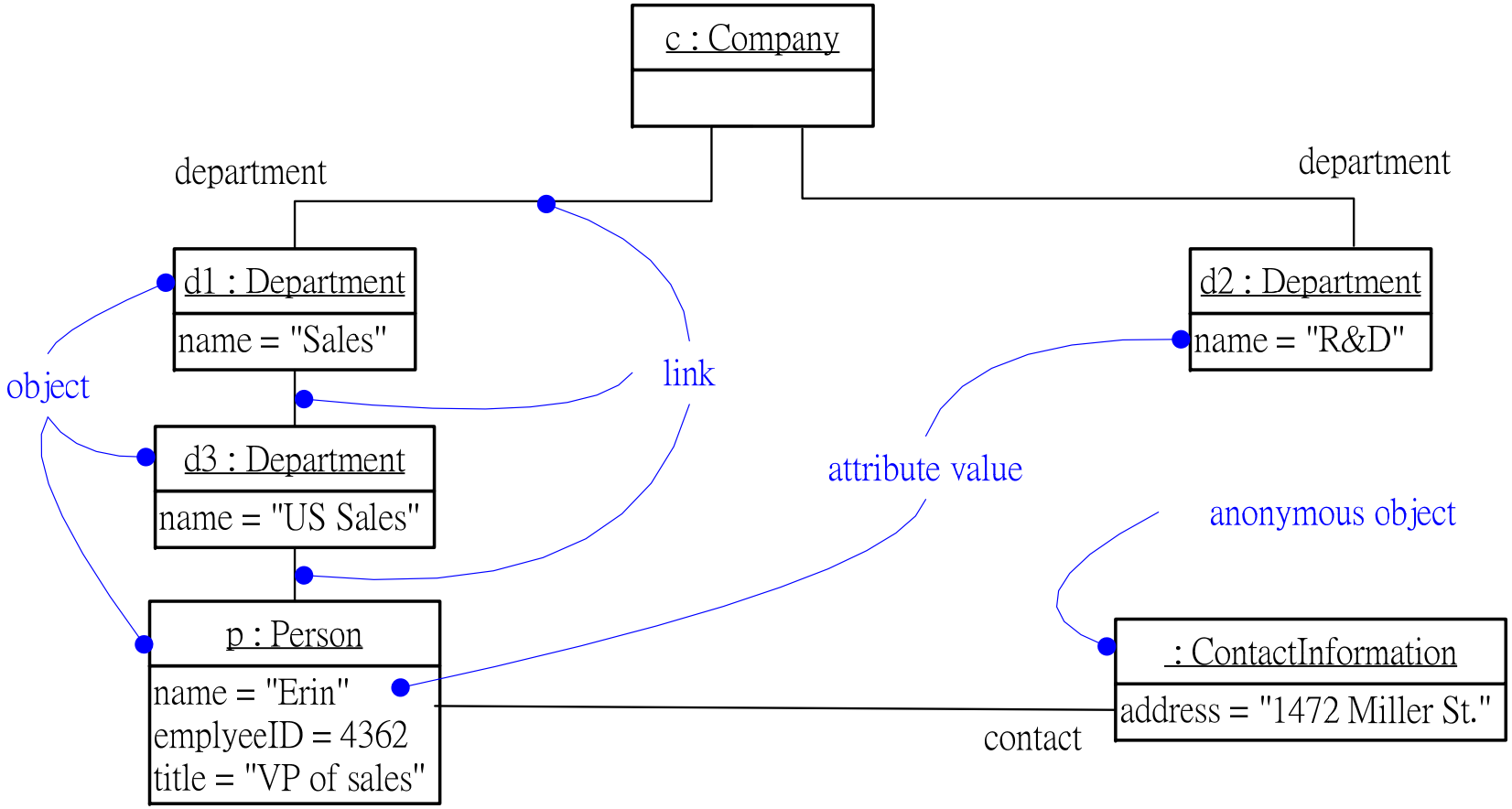
Instances

- An instance is a concrete manifestation of an abstraction
  - Set of operations (that can be applied to the instance)
  - State (that stores the effects of operations)
- Graphical notation:

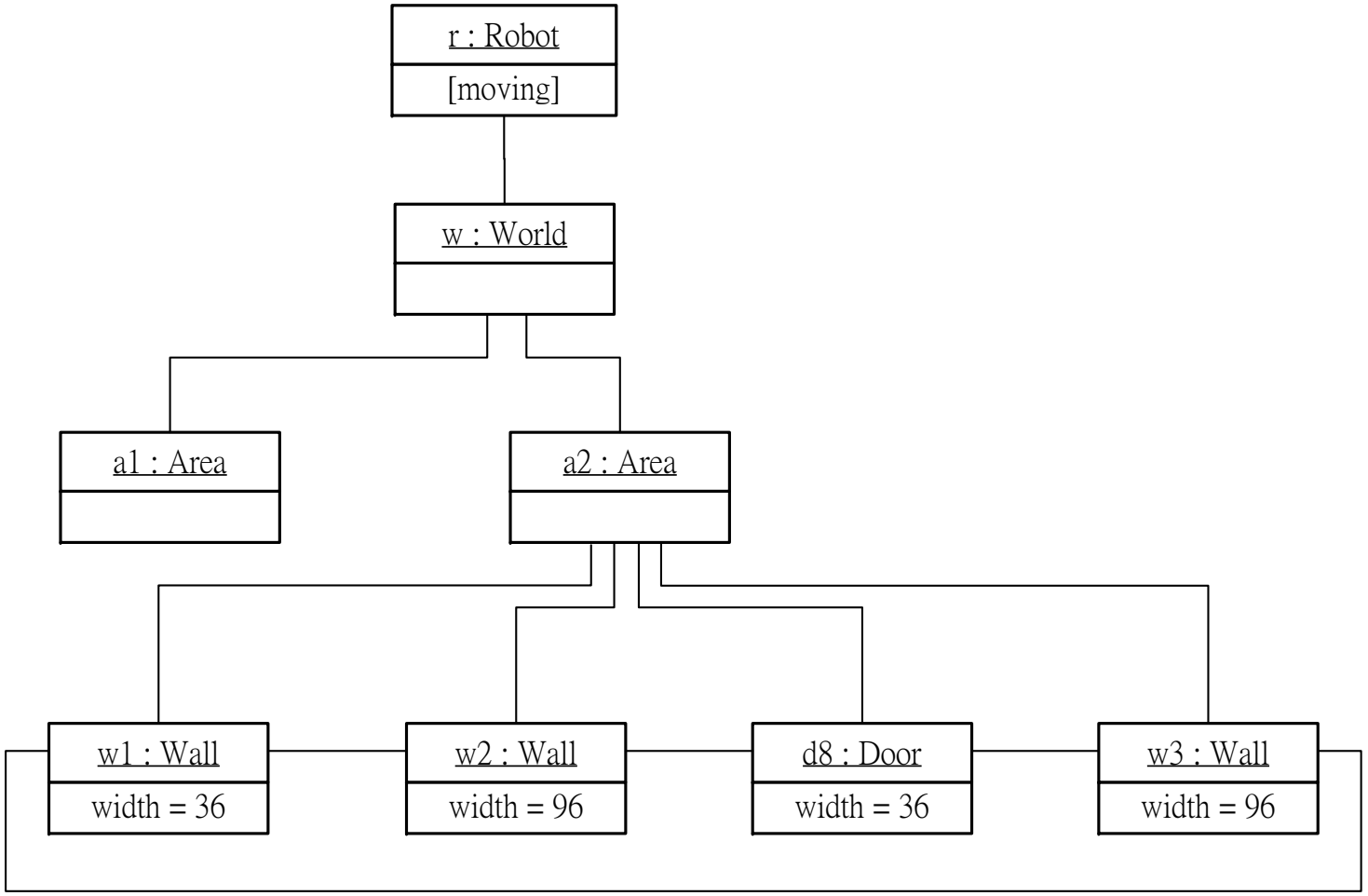


Object Diagrams

- An object diagram shows a set of objects and their relationships at a point in time.



Modeling Object Structures

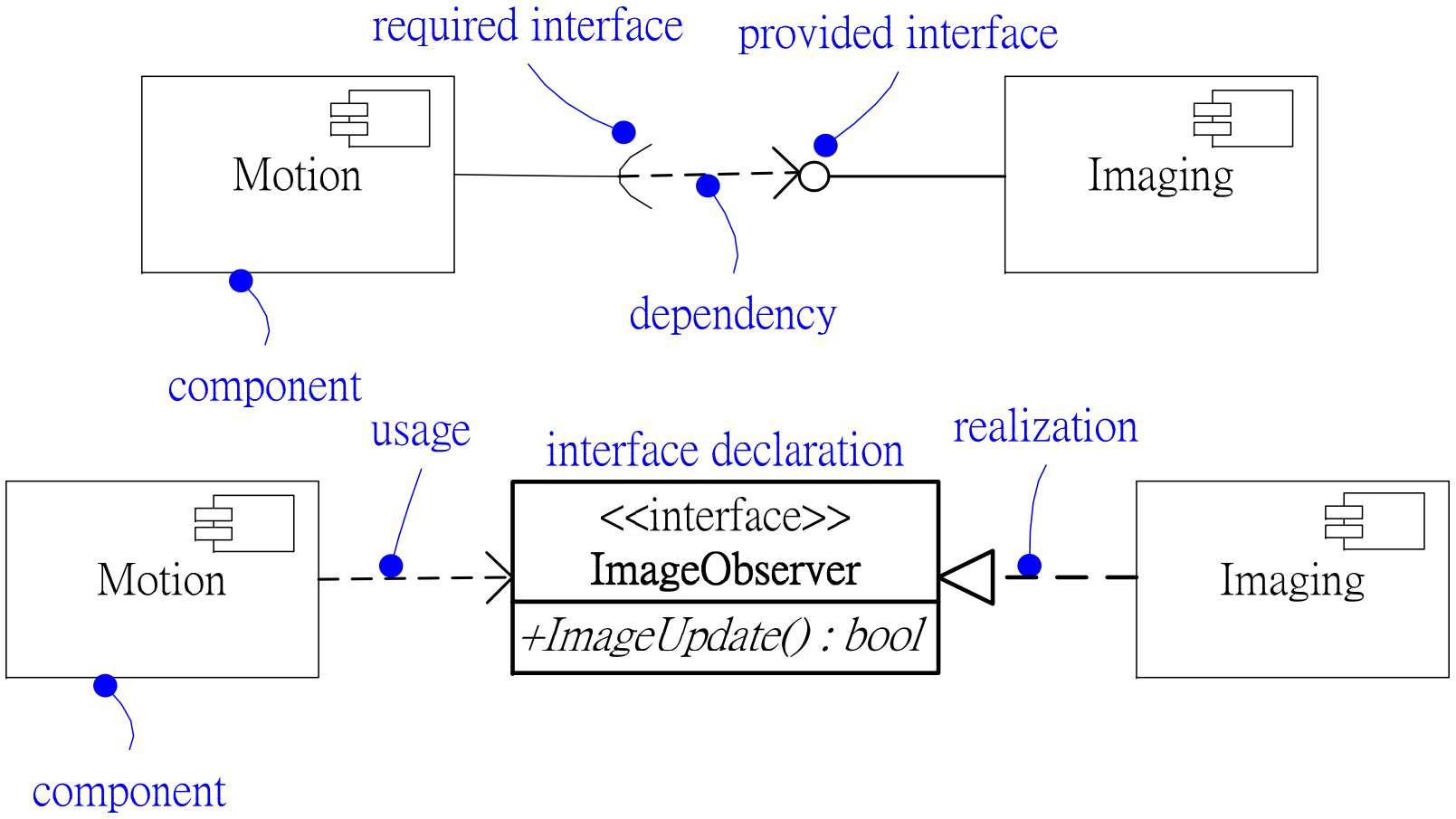


Components

- A component is a *logical*, *replaceable* part of a system that conforms to and realizes a set of *interfaces*.
- Relevant concepts
  - Interface: a collection of operations. Interfaces are the glue that binds components together.
  - Port: a window for accepting and sending messages
  - Internal structure: implementation of a component
  - Part: a unit of the implementation
  - Connector: a communication relationship between two parts or ports

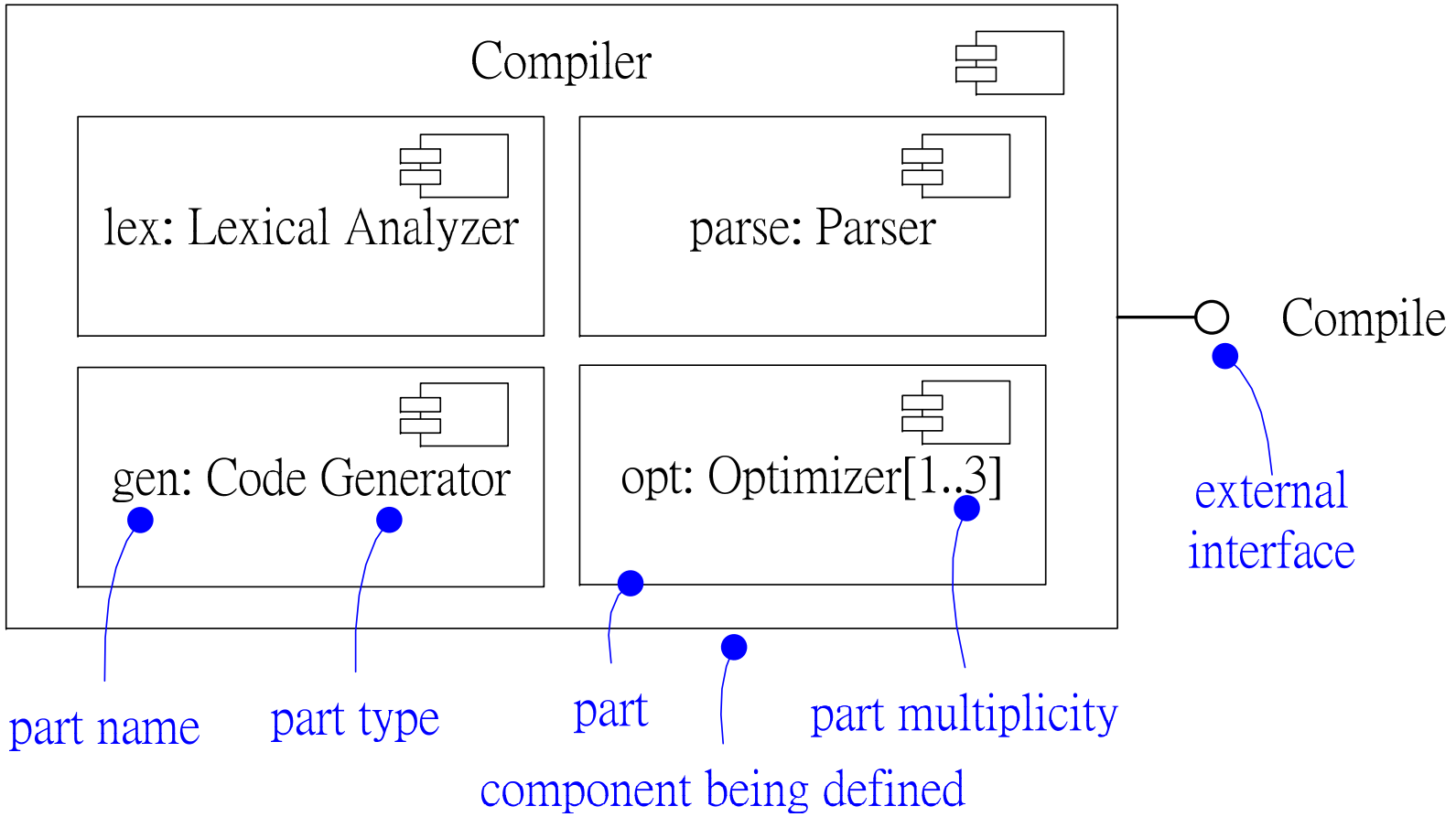
Component Diagrams

- Components are bound by interfaces.



Component Diagrams (cont.)

- Components can be nested.

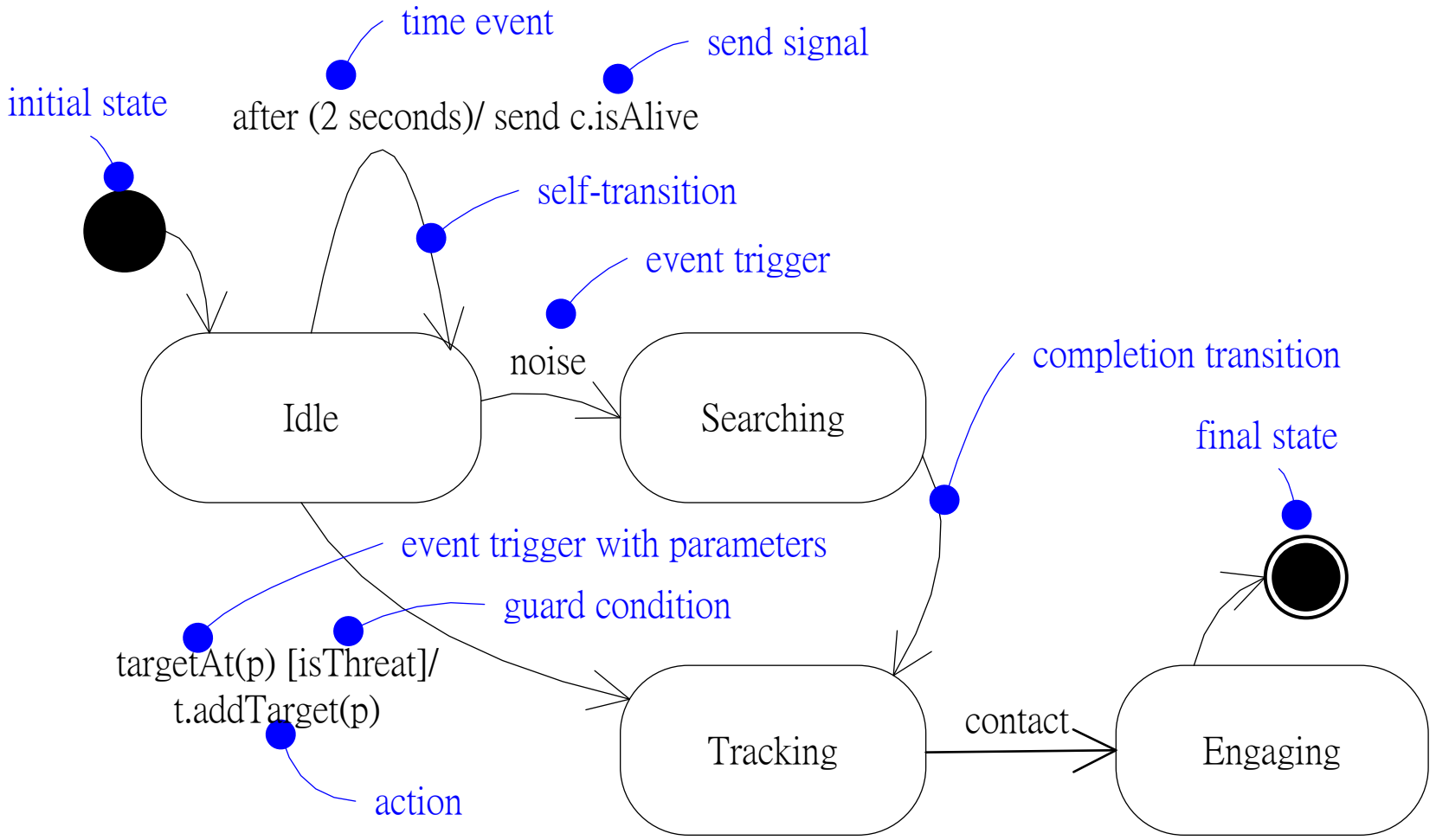


Events and Signals

- "Things that happen" are called events.
- They are used to model the occurrence of a stimulus that changes the state of a system.
- Events may include
  - Signals,
  - Calls,
  - The passing of time, or
  - A change in state
- Events may be synchronous or asynchronous.

State Machines

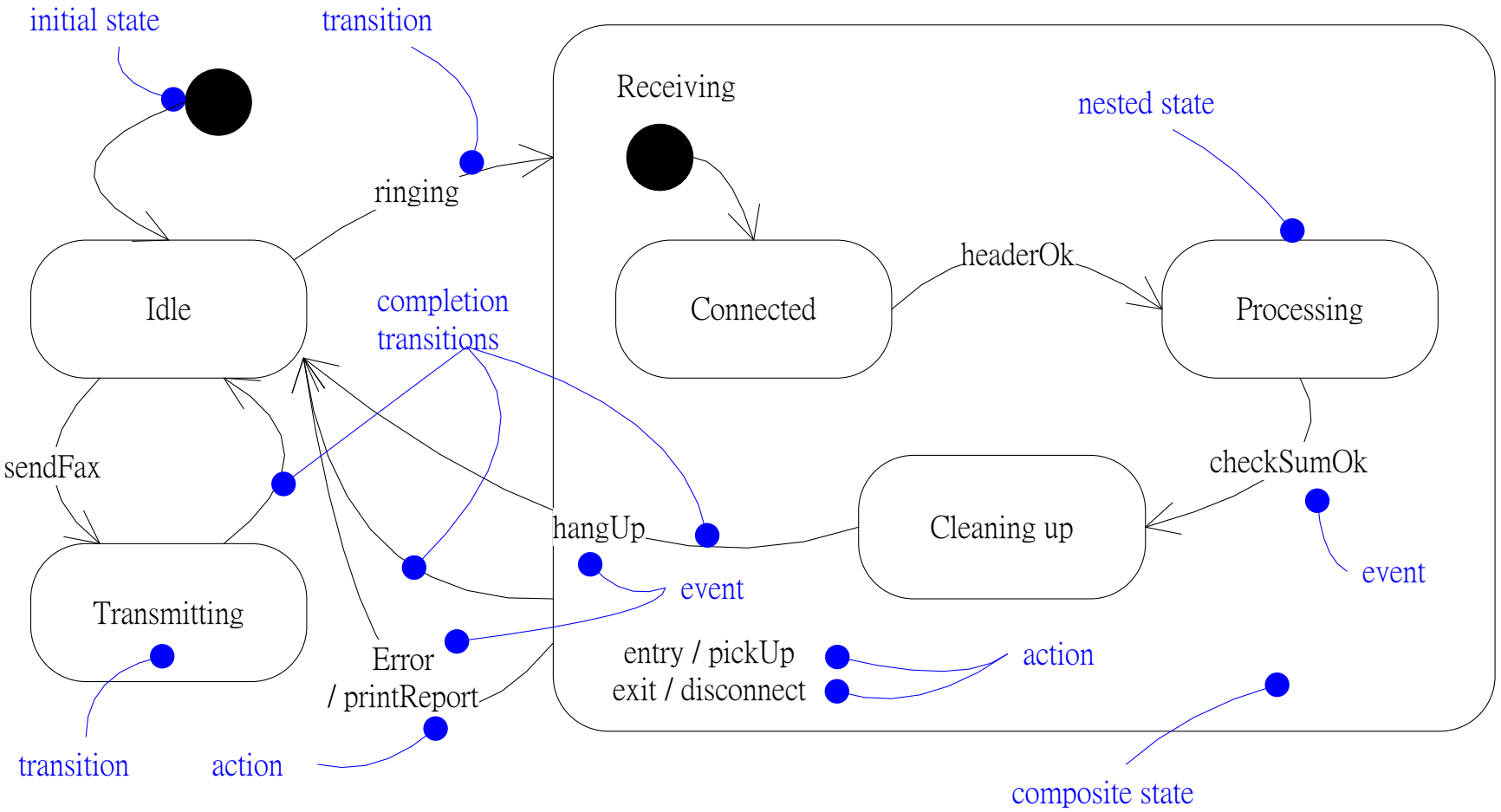
- A state machine models the lifetime of an object.



Advanced States and Transitions

- Entry/exit effects
- Internal transitions
- Do-activities
- Deferred events
- Submachines
- Nonorthogonal vs. orthogonal states (sequential vs. concurrent states)
- History states (cf. static variables in C)
- Fork and Join

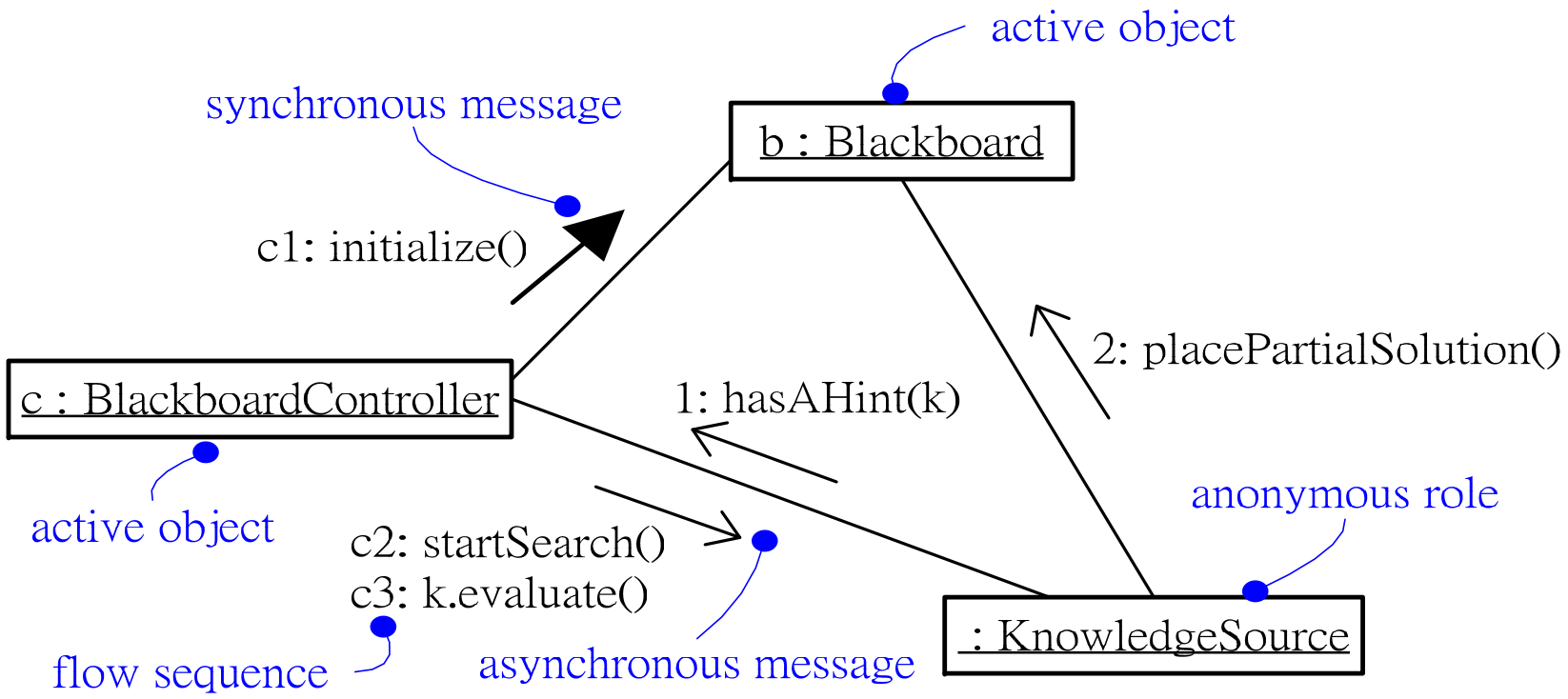
State (Statemachine) Diagrams



Processes and Threads

- Flow of control
  - Heavyweight: Process
  - Lightweight: Thread
- Active class/object (representing a process or thread)
- Communication
- Synchronization
  - Sequential
  - Guarded
  - Concurrent

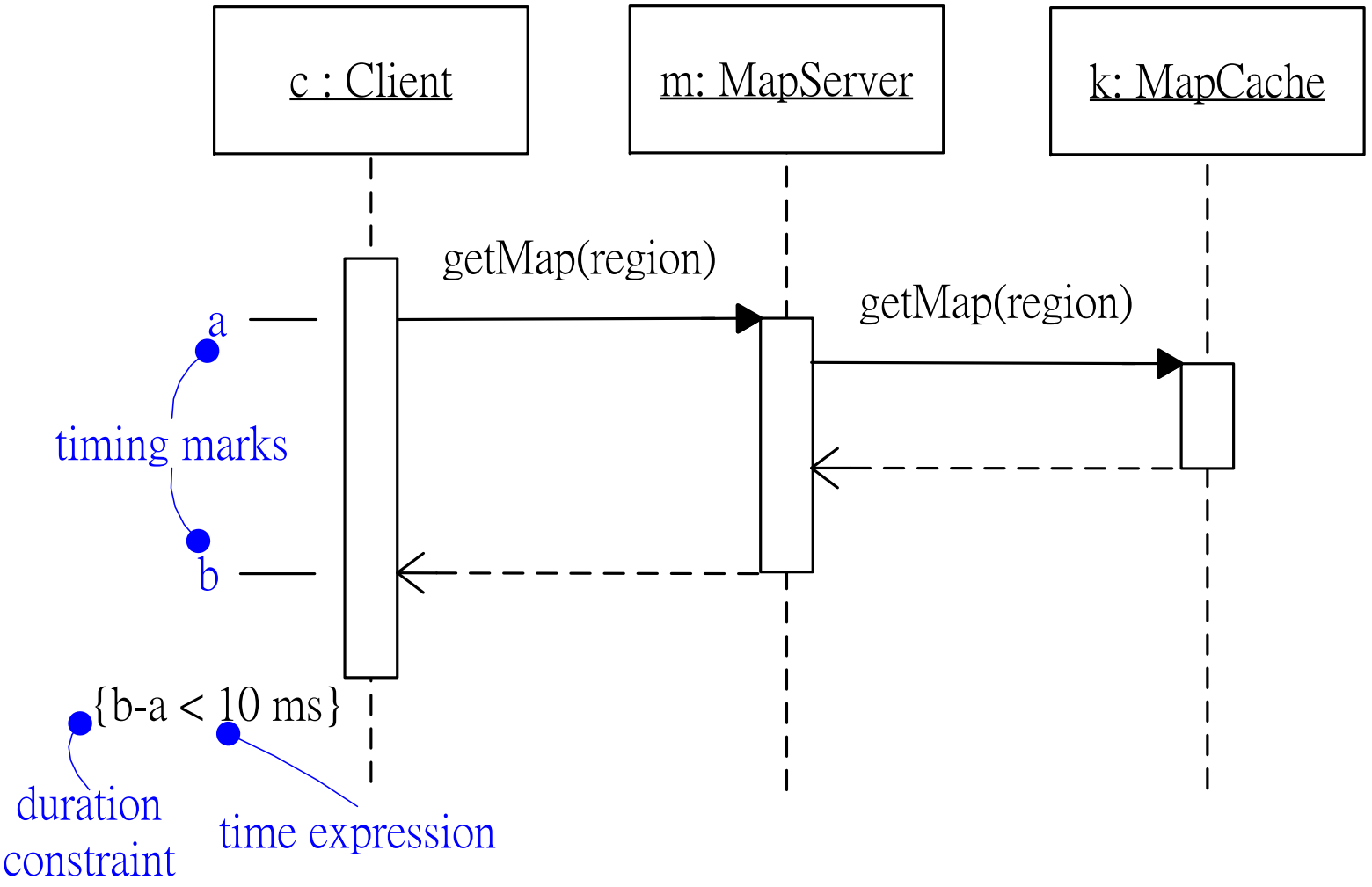
Communication



Timing Constraints

- Some systems may be time-critical.
- Even if not time-critical, meeting more stringent timing constraints is a good indicator of efficiency.
- Typical timing constraints:
  - Duration
  - Frequency

Duration Constraints



Improvements Made in UML 2.0

- Hierarchical decomposition of structures and support for component-based development:
  - Composite structure diagrams
- Hierarchical decomposition of behavior
- Improved integration between structural and behavioral models
- Support for executable models
  - fully integrated Action Semantics