UML for embedded software (II)

課程：嵌入式系統與軟體工程

開發學校：輔仁大學資工系

范姜永益

How to Describe Behavior?

Behavior Modeling

- An Introduction to Behavior Modeling
- Sequence Diagrams
- Collaboration Diagrams (UML 2.0: Communication Diagrams)
- Statechart Diagrams
- Activity Diagrams

An Introduction to Behavior Modeling

- All systems have a *static structure* (documented by class diagrams) and *dynamic behavior* (documented by dynamic diagrams).

- Dynamic behavior is described by:
  - Sequence Diagrams: describe how objects interact and communicate with each other, focusing on time sequence;
  - Collaboration Diagrams(*): describe how objects interact, focusing on space;
  - Statechart Diagrams(**): describe which states an object can have during its life cycle, and the behavior in those states; and
  - Activity Diagrams: show objects that perform work in terms of activities.

UML 2.0: (*) Communication diagrams; (**) State machine diagrams.

Introduction to Behavior Modeling (cont'd)

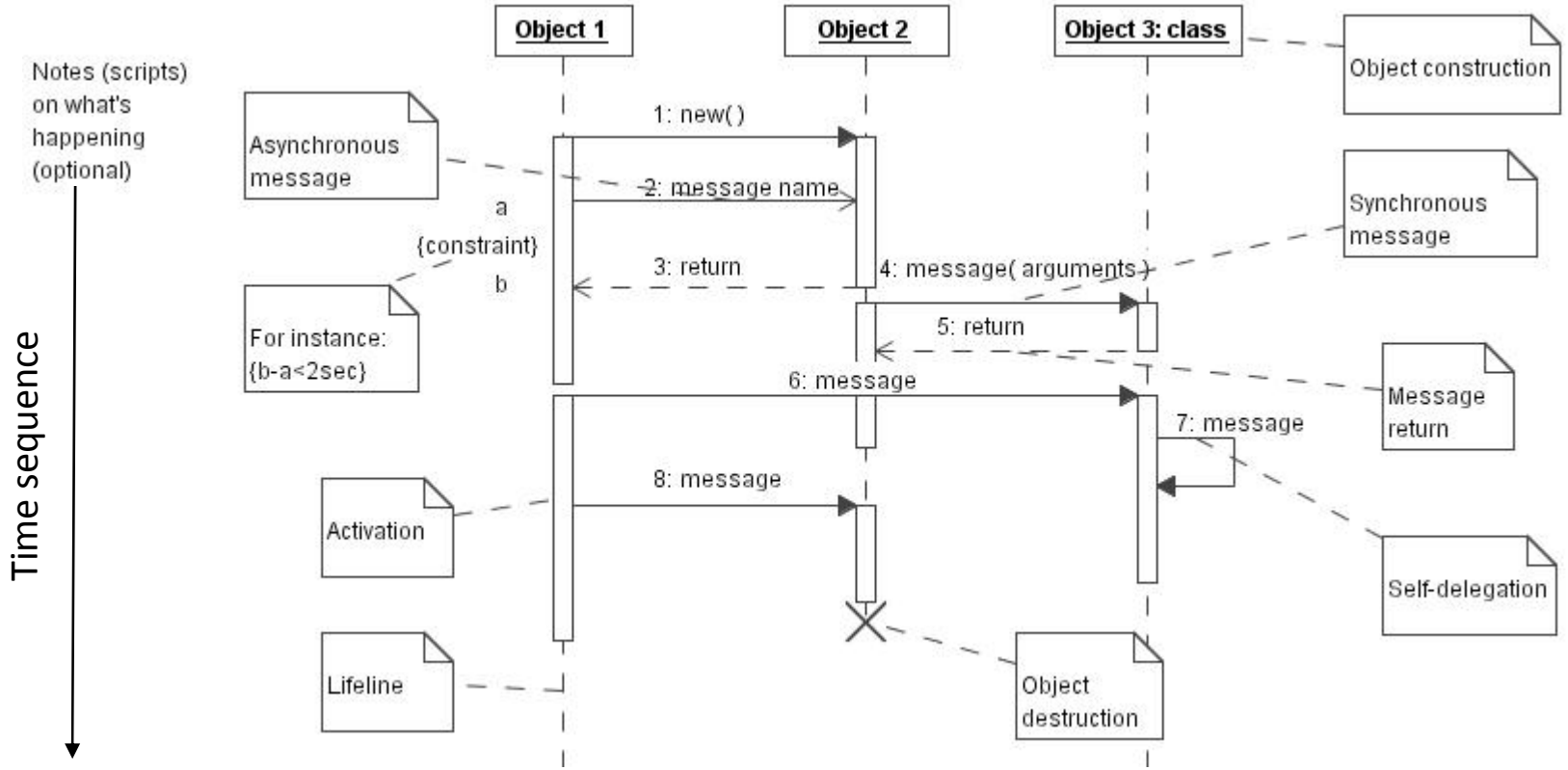
- **Interaction Diagrams** are used when modeling the dynamic aspects of a system. There are two way to use interaction diagrams [Booch et al. 1999]:
  - To model flows of control by time ordering
    - Sequence diagrams are used
  - To model flows of control by organization
    - Collaboration diagrams are used.

Sequence Diagrams

- A **Sequence Diagram** shows the chronological order of objects interactions.
  - The objects (not classes) and the messages that pass between them when an interaction occur.
    - Messages show one object interacting with another.
  - Two specific features of a sequence diagram:
    - **Object lifeline**: represents the existence of an object over a period of time, that is, objects may be created or destroyed during the interaction.
    - **Focus of control**: shows which object is currently active, that is, the activations of the objects.
  - Using sequence diagrams
    - To map functionality stipulated by use cases into classes
    - To help determine what classes to reuse, enhance

Sequence Diagrams (cont'd)

- Notations to create a sequence diagram.

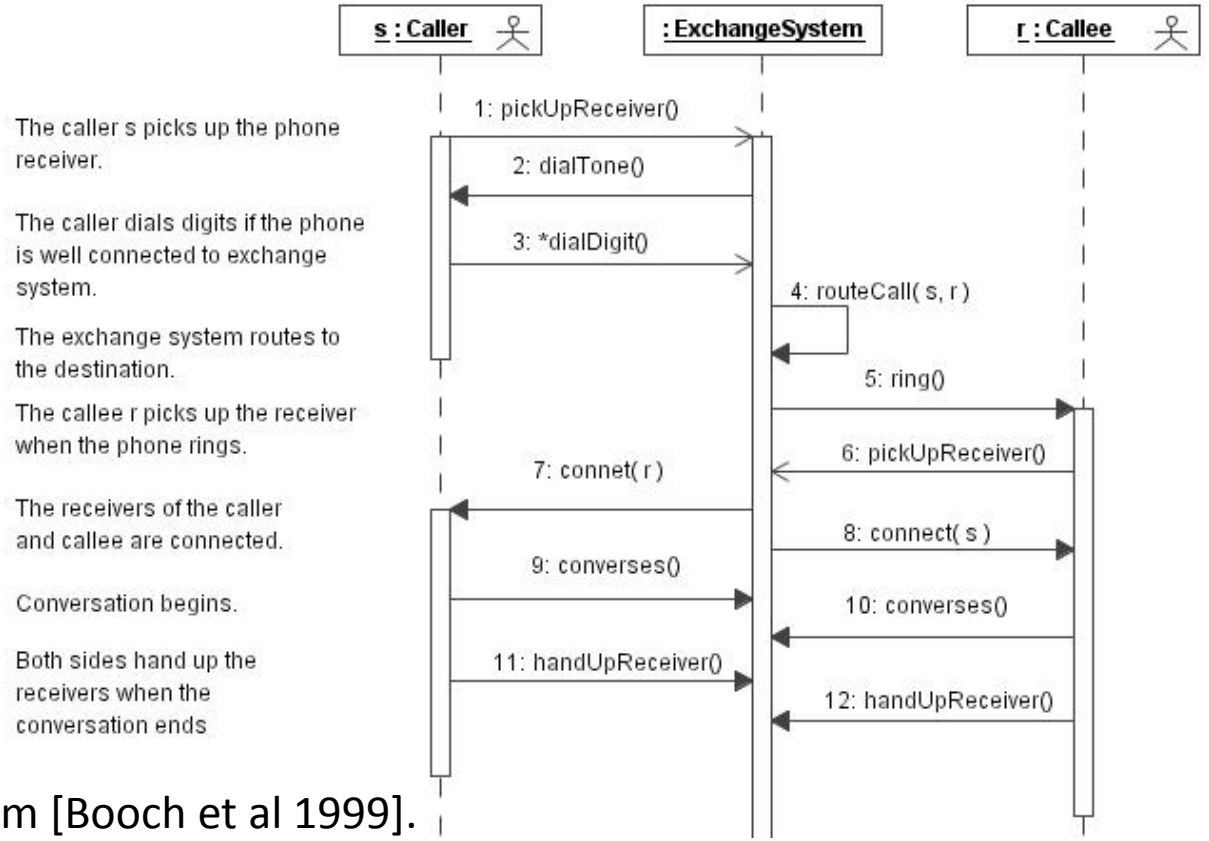


Sequence Diagrams (cont'd)

- Message names:
  - **Synchronous message**: the sender sends the message and waits for the receiver to return from the executing the message.
  - **Asynchronous message**: the sender sends the message and continually executes it, not waits for a return from the sender.
  - **Message return**: the receiver of the message returns activation of the sender's message.
  - **Object destruction**: The sender destroys the receiver.
  - **Activation (or focus of control)**: When a lifeline is executing a message.

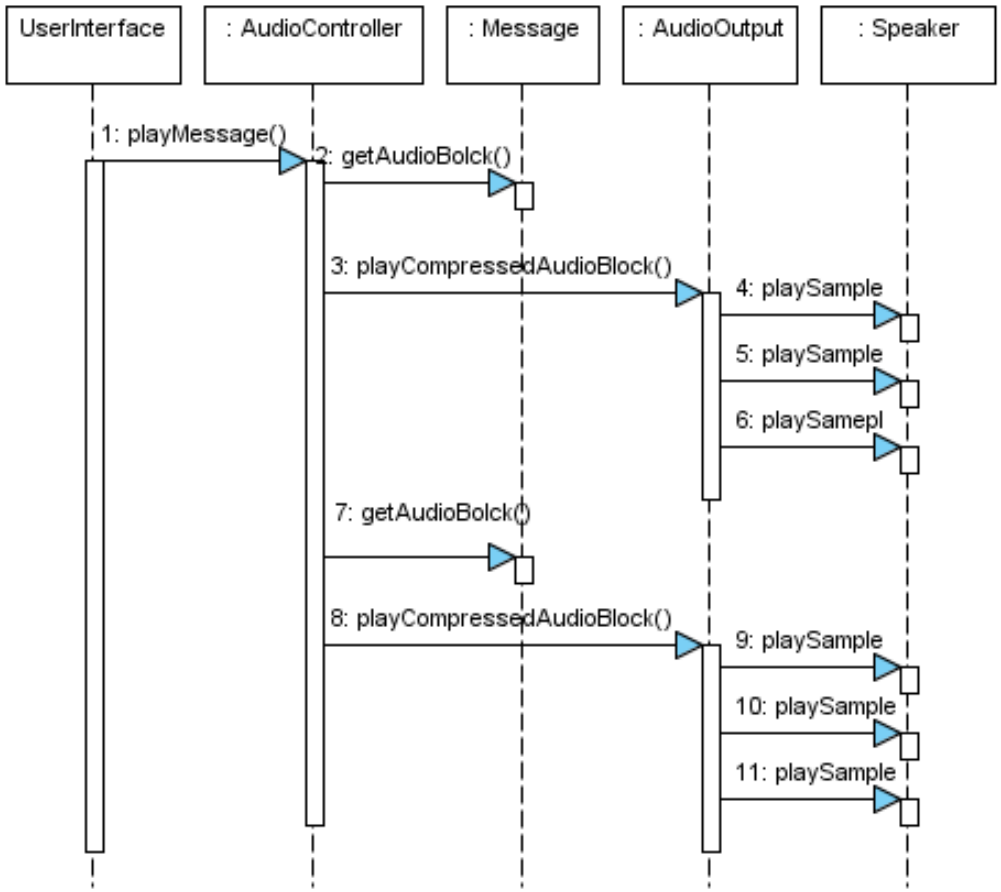
Sequence Diagrams (cont'd)

- Example: A simple sequence diagram of Phone Call(*).



(*) Adapted from [Booch et al 1999].

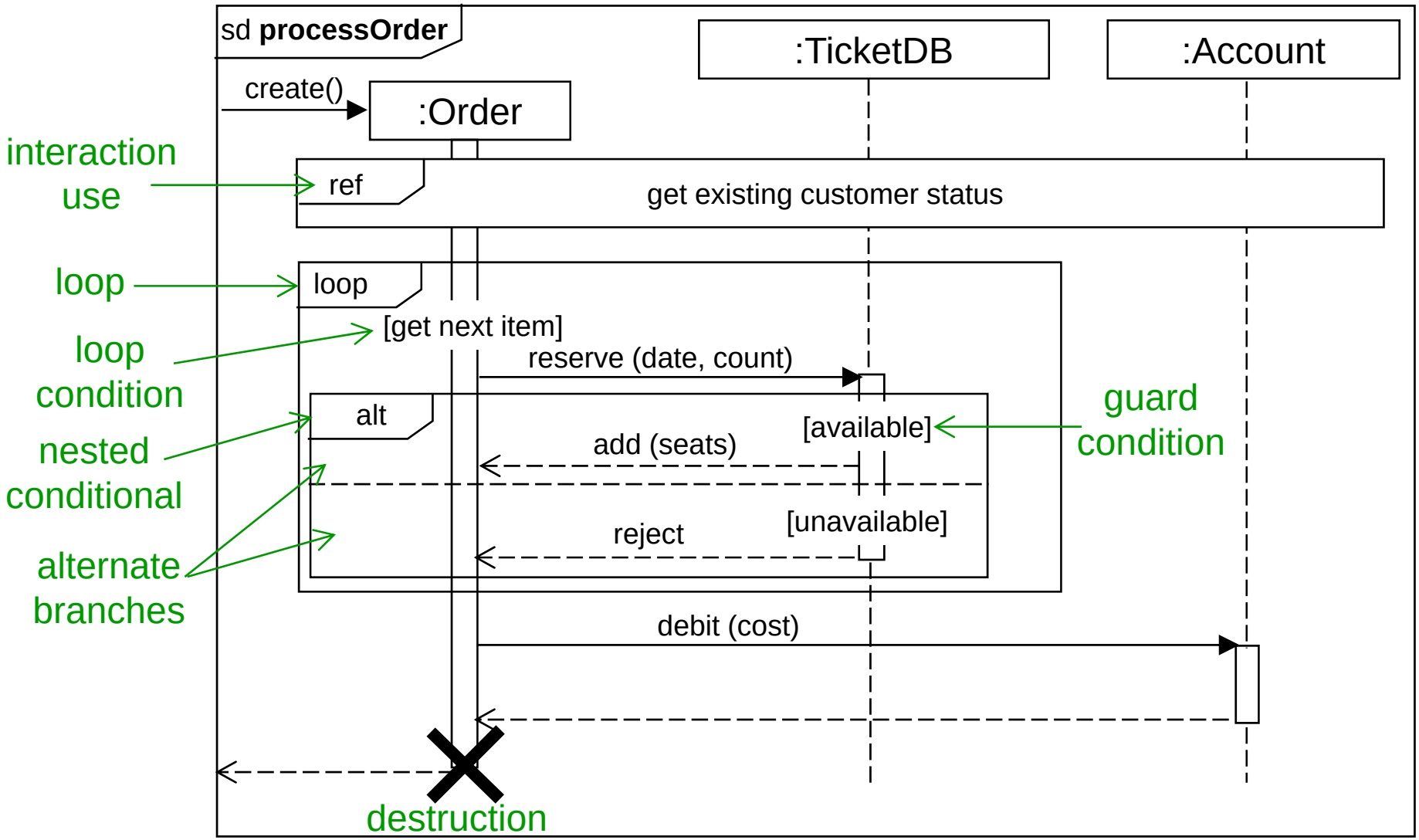
Audio Output to Speaker



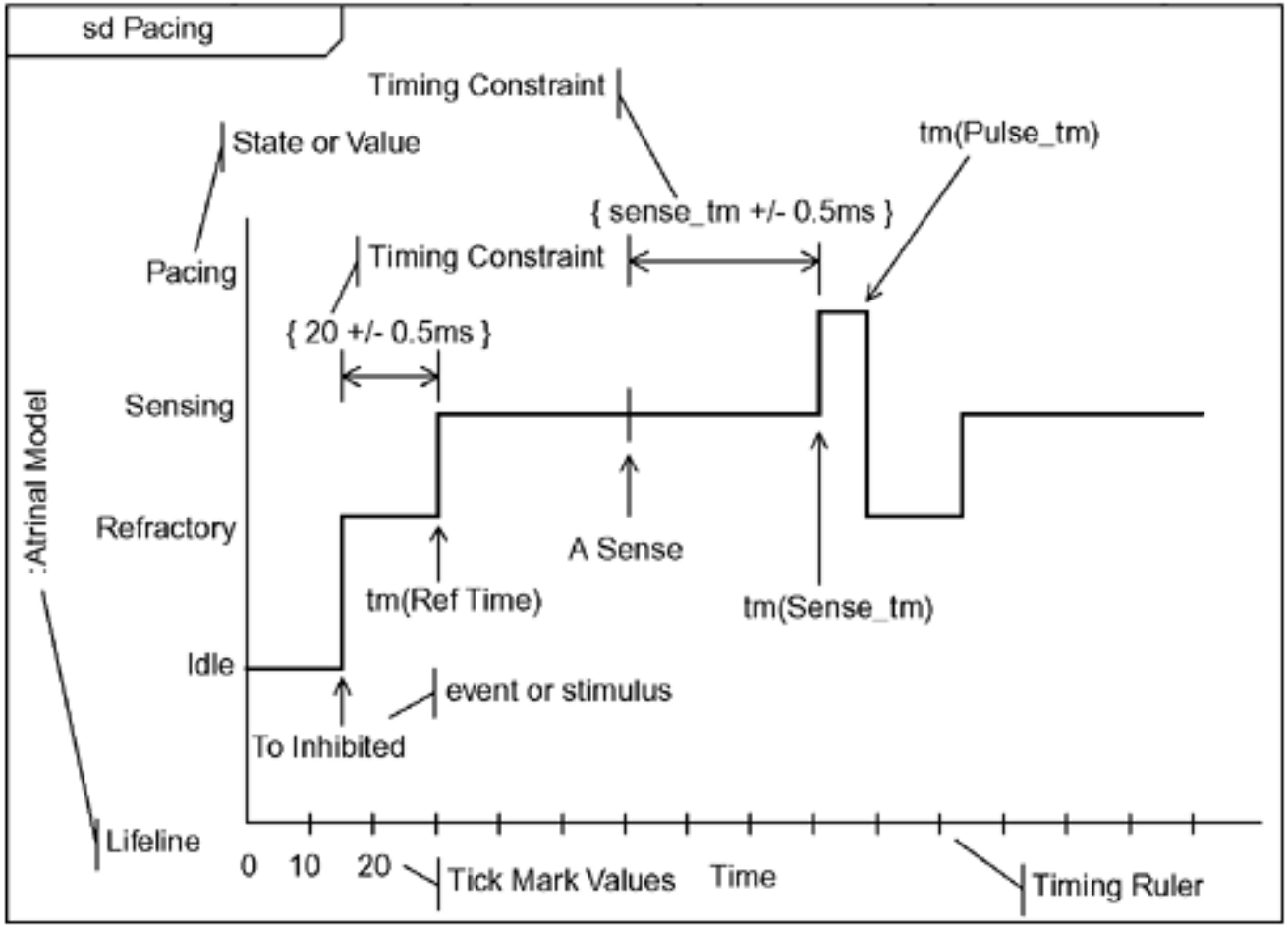
Decomposing Sequence Diagrams

- Sequence diagrams suffer from scalability problems when there are any
  - Lifelines
  - Messages
- UML 2 provides the ability to decompose sequence diagrams both vertically and horizontally
  - Lifelines may be decomposed into interactions (separate sequence diagrams)
  - Interaction fragments may be referenced on another diagram

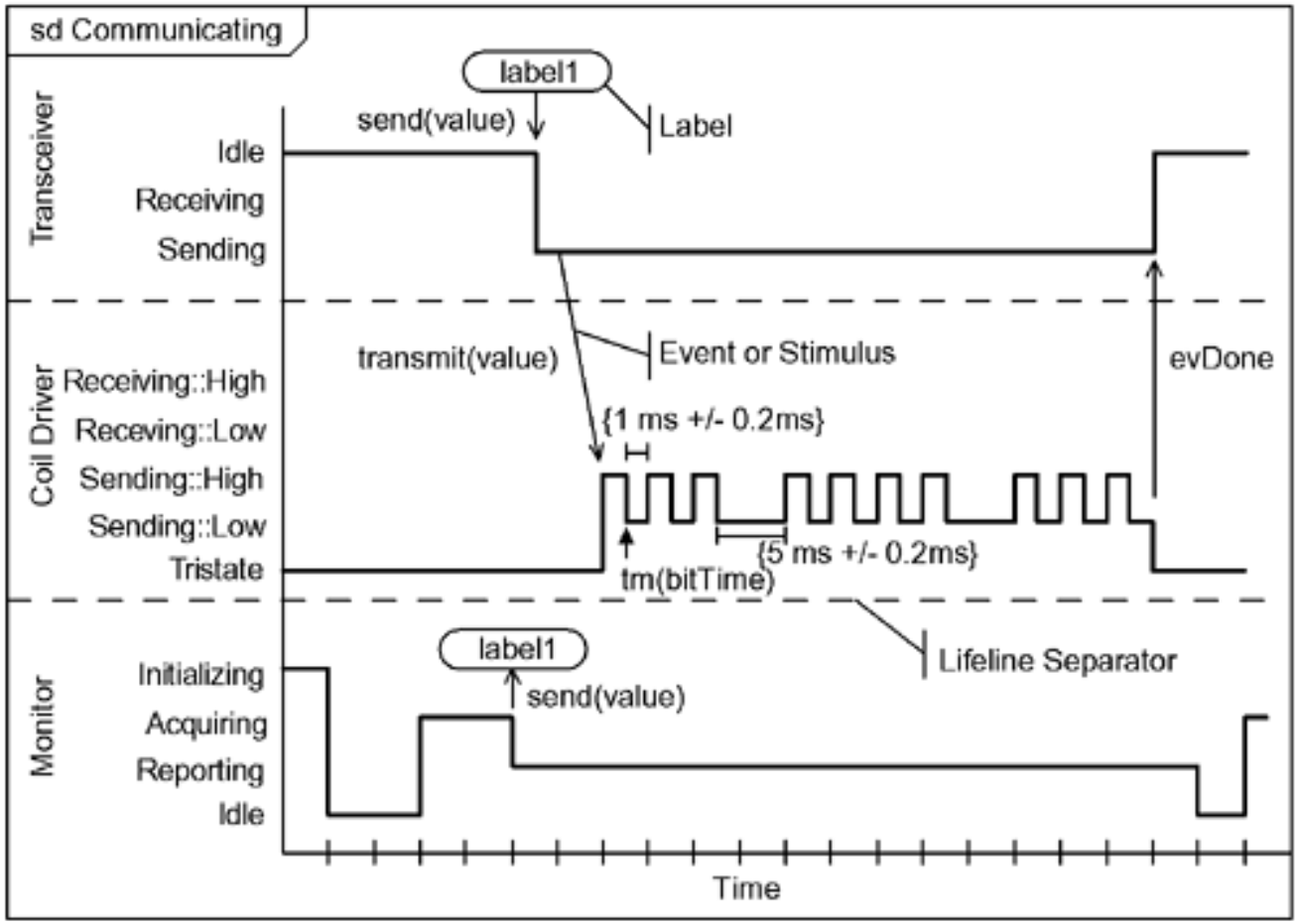
SEQUENCE DIAGRAMS



UML 2.0 Timing Diagrams



UML 2.0 Timing Diagrams

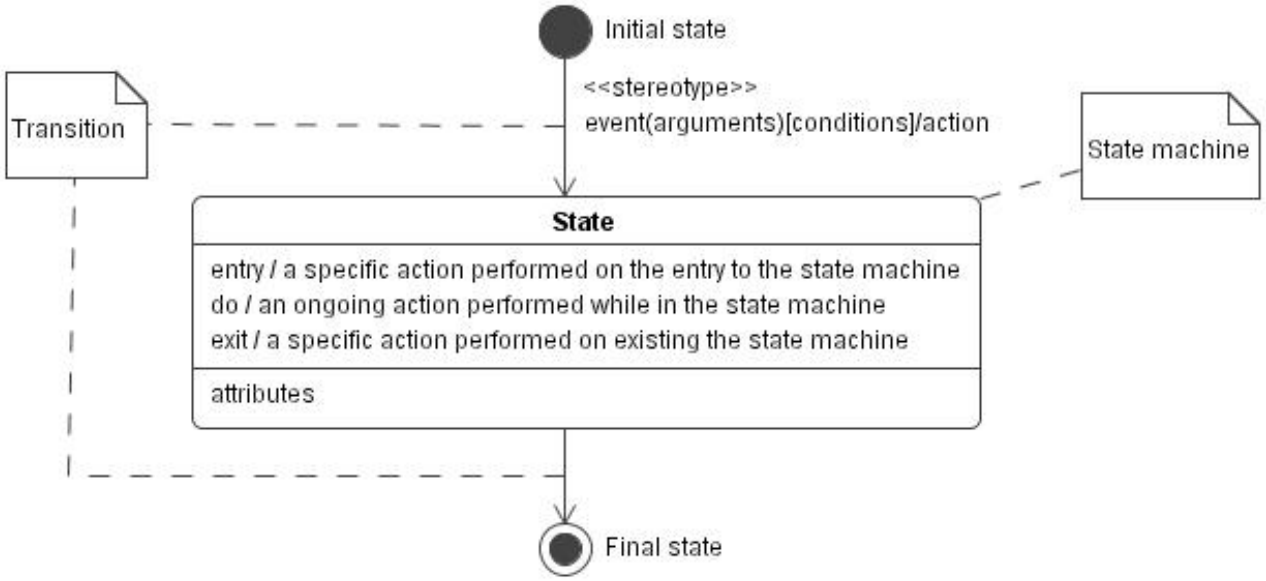


State Machine Diagrams

- A **State machine Diagram** shows a structured finite state machine that expresses an object lifecycle, that comprises:
  - **State**: represents a situation during the life of an object;
  - **Event:** represents an incident or indications that progression is happening;
  - **Transition**: specifies what new state is achieved under transition rule when an object receives a particular event;
  - **Action**: an executable atomic computation that results in a transition from one state and another, and cannot be interrupted.

Statechart Diagrams (cont'd)

- A **State Machine** specifies the sequences of states during the lifetime of an object in response to events, together with its response to those events.
  - A state machine: state, transition, initial state, final state



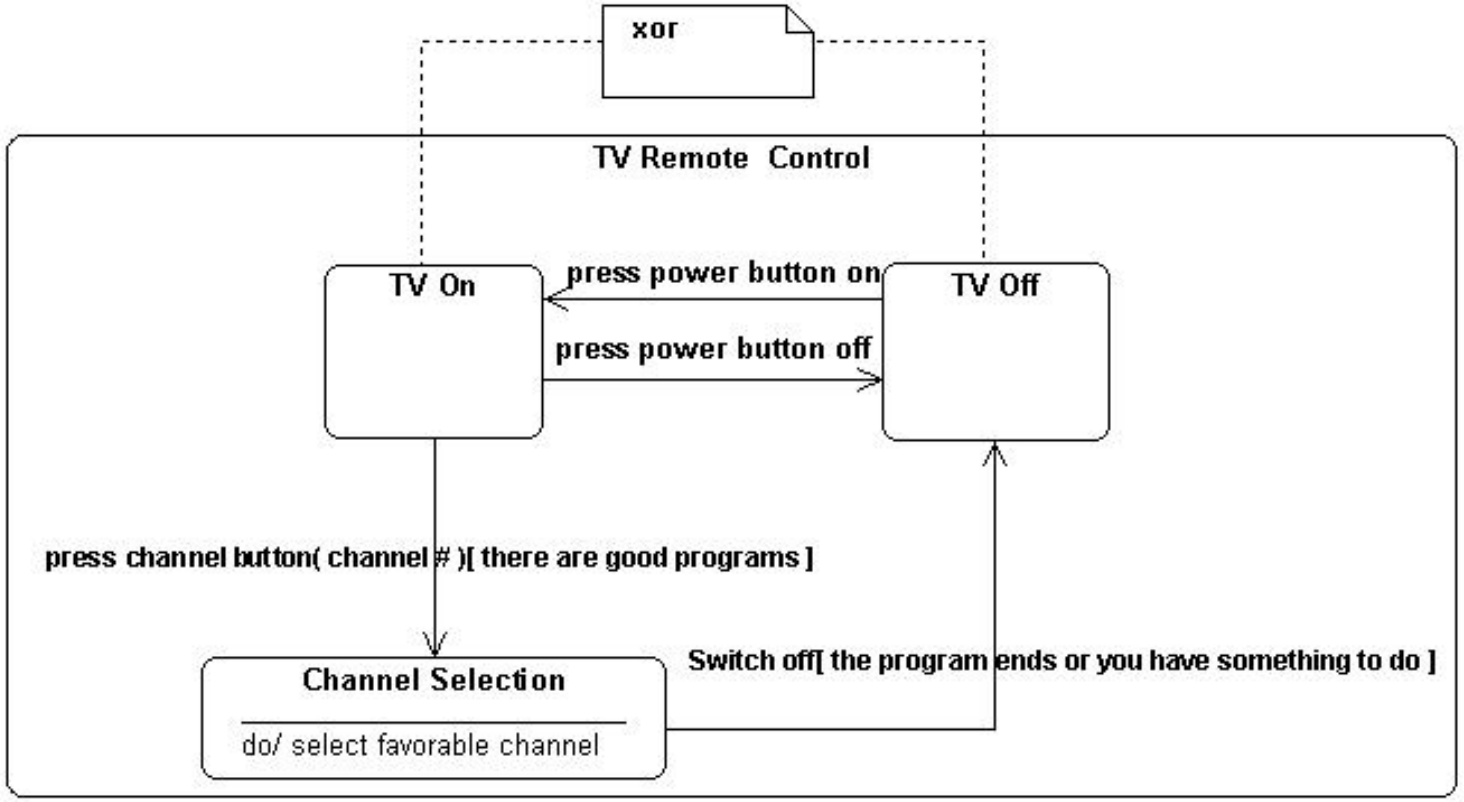
Statechart Diagrams (cont'd)

- One state machine for each class (a class may be seen as a state machine). For example:

ServiceAgreement
serviceLevel : ServiceLevelEnum saPrice : Double saDescription : String
<<event>> requestAgreement() <<event>> negotiation() <<event>> conformedAgreementDraft() <<event>> signingAgreementDocument()

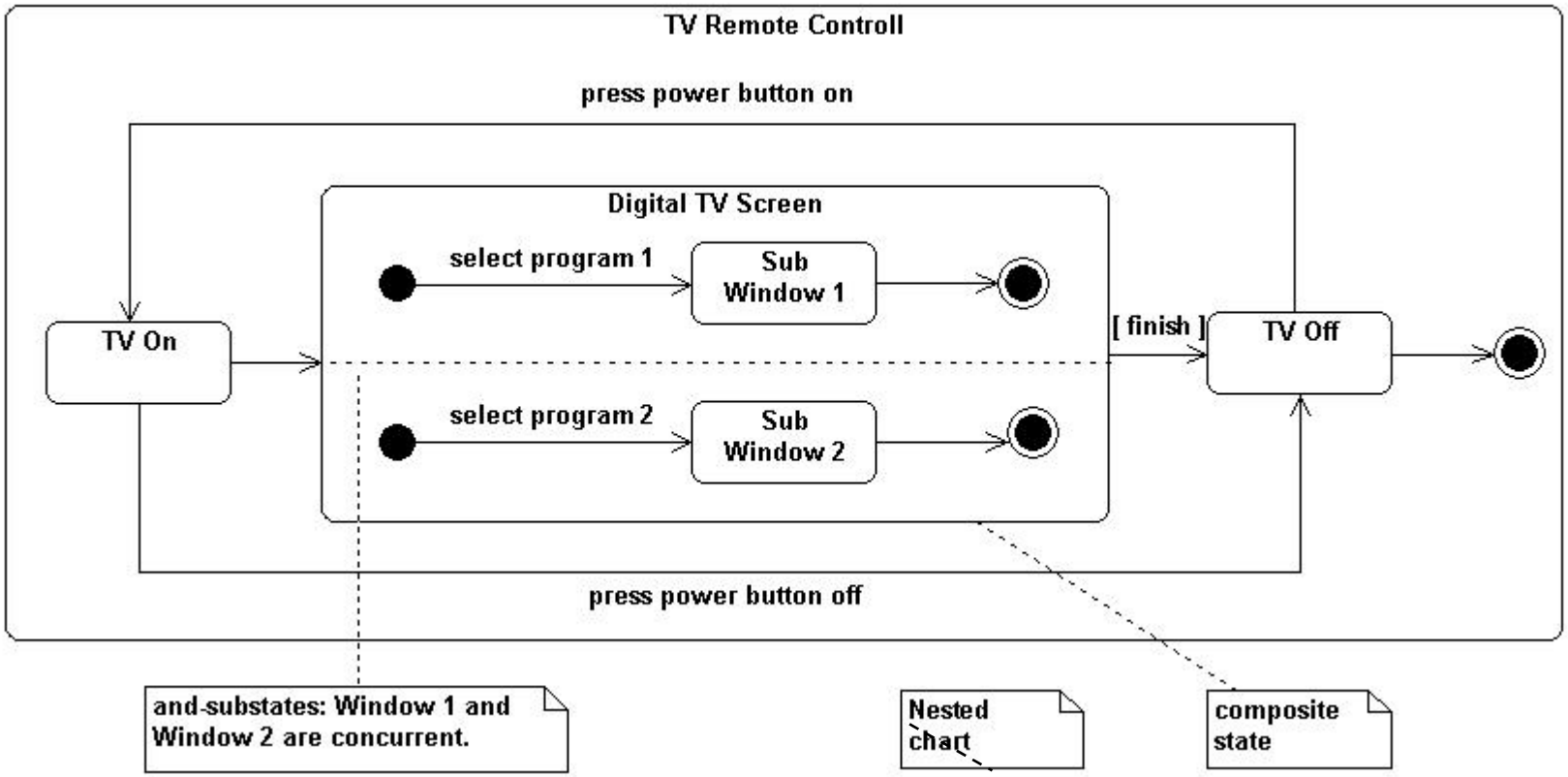
Statechart Diagrams (cont'd)

- Generalization: or (xor) substates



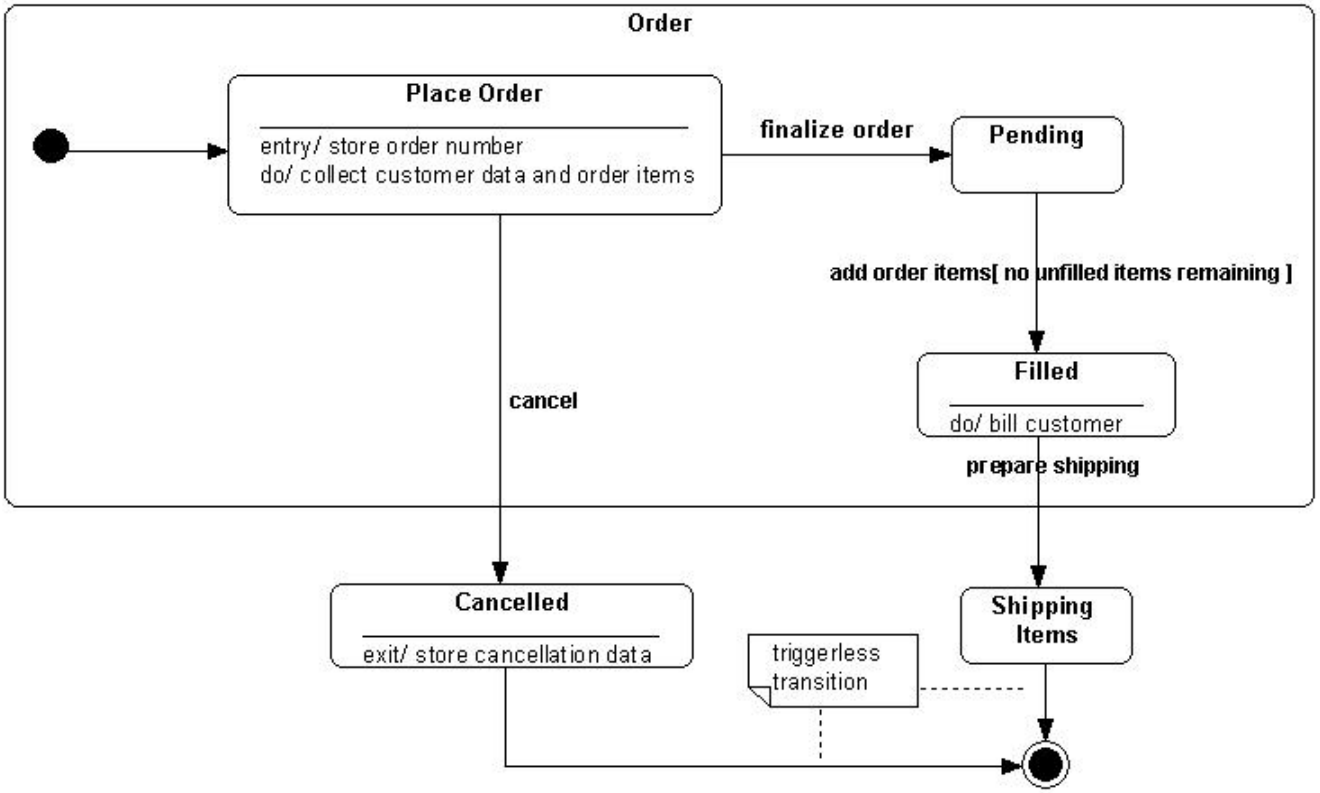
Statechart Diagrams (cont'd)

- Aggregation: and-substates (concurrent substates)



Statechart Diagrams (cont'd)

- Example: Order class



Types of Events

- UML defines 4 kinds of events
  - **Signal Event**
    - Asynchronous signal received e.g. evStart
  - **Call Event**
    - operation call received e.g. opCall(a,b,c)
  - **Change Event**
    - change in value occurred
  - **Time Event**
    - Absolute time arrived
    - Relative time elapsed e.g. tm(PulseWidthTime)

Handling Transitions

- If an object is in a state S that responds to a named event evX, then it will act upon that event
- It will transition to the specified state, if the event triggers a named transition and the guard (if any) on that transition evaluates to TRUE. It will execute any actions associated with that transition
- Handle the event without changing state if the event triggers a named reaction and execute all the list of actions associated with that reaction

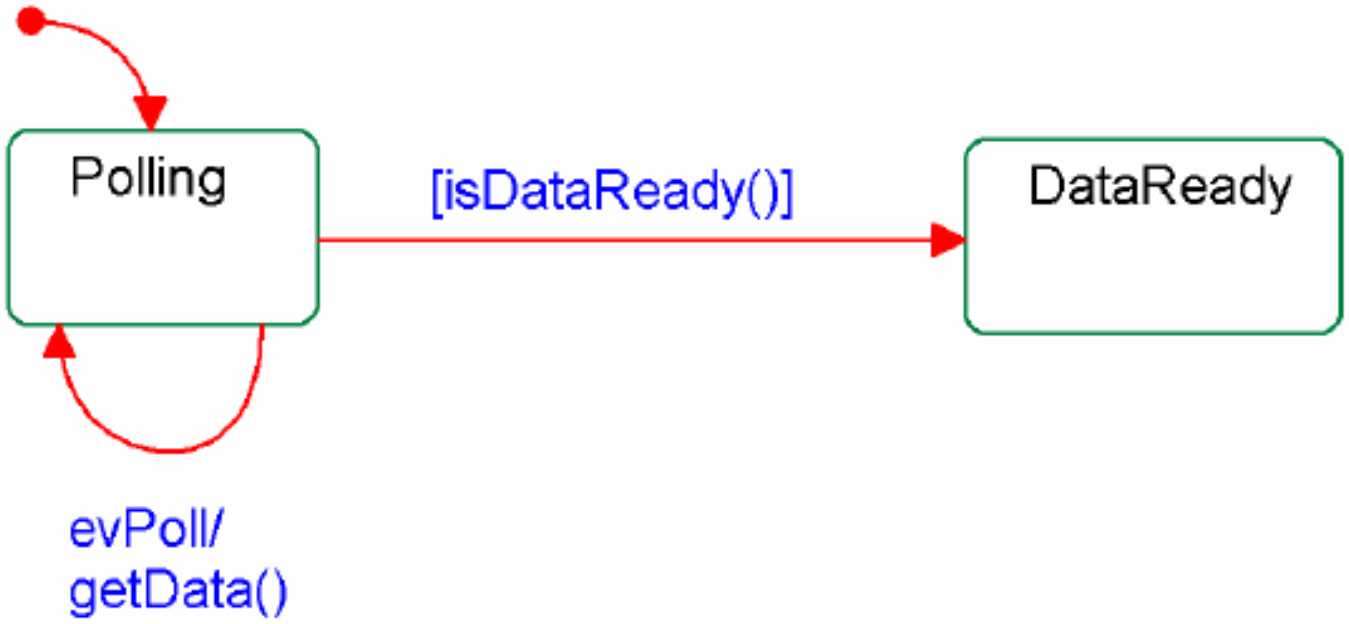
Transitions: Guards

- A **guard** is some condition that must be met for the transition to be taken
- Guards are evaluated prior to the execution of any action.
- Guards can be:
  - Variable range specification ex: [cost<50]
  - Concurrent state machine is in some state [IS_IN(fault)]
  - Some other constraint (preconditional invariant) must be met

Actions

- Actions are run to completion
  - Normally actions take a short period of time to perform
  - They may be interrupted by another thread execution, but that object will complete its action list before doing anything else
- Actions are implemented via
  - An object's operations
  - Externally available functions
- They may occur when
  - A transition is taken
  - A reaction occurs
  - A state is entered
  - A state is exited

Null-triggered Transitions

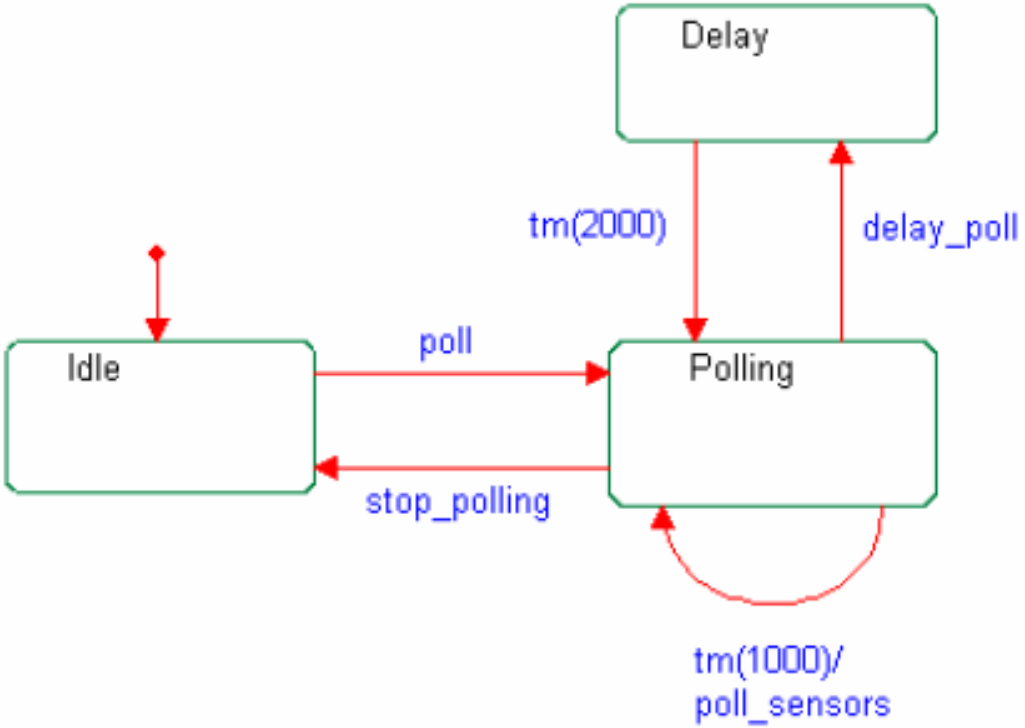


Timeouts

- When an object enters a state, any Timeout from that state are started
- When a Timeout Expires, the State machine receives the expiration as an event
- When an object leaves a state, any timeout that was started on entry to that state are cancelled
- Only one timeout can be used per state, nested states can be used if several timeouts are needed

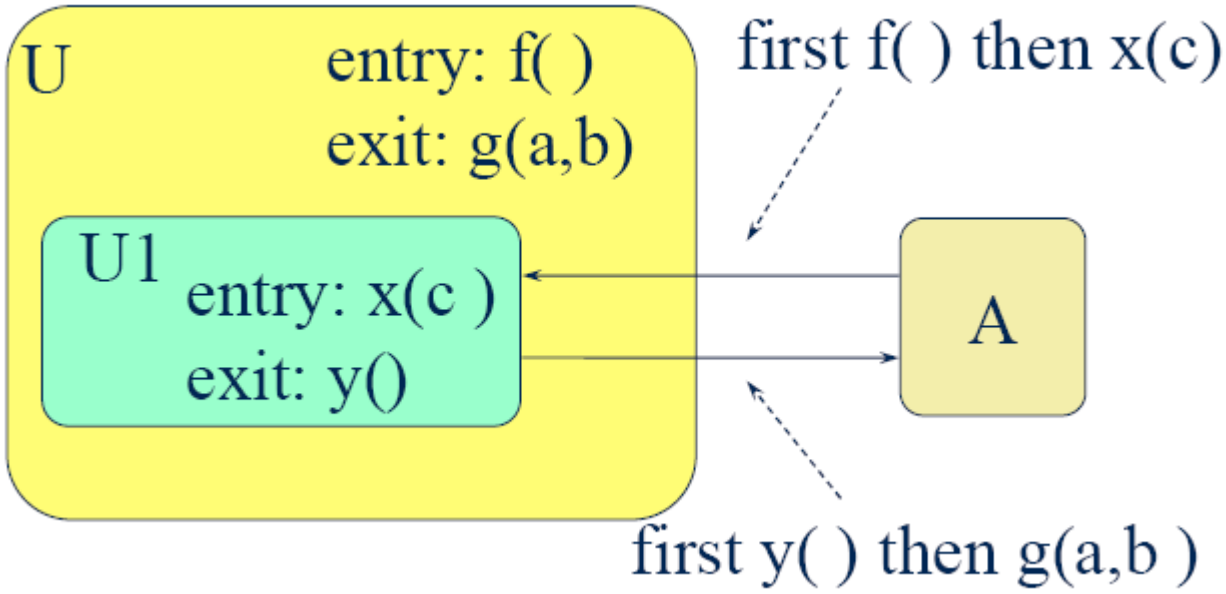
Timeouts

- Timeouts example:



Order of Nested States

- Execute from outermost – in on entry
- Execute from innermost – out on exit



Activity Diagrams

- An **Activity** is an ongoing non-atomic execution within a state machine.
  - An activity is interruptible.

- An **Activity Diagram** "shows the flow from activity to activity, addressing the dynamic view of a system." [Booch et al. 1999]
  - An activity diagram is "a special case of a state machine that used to model processes involving one or more classifiers (*)." [OMG-UML v1.4]
  - An activity diagram is used to model the flow of activities in a procedure.

(*) A model elements that describes behavior and structural features, e.g., class, use case, interface, etc.

Activity Diagrams (cont'd)

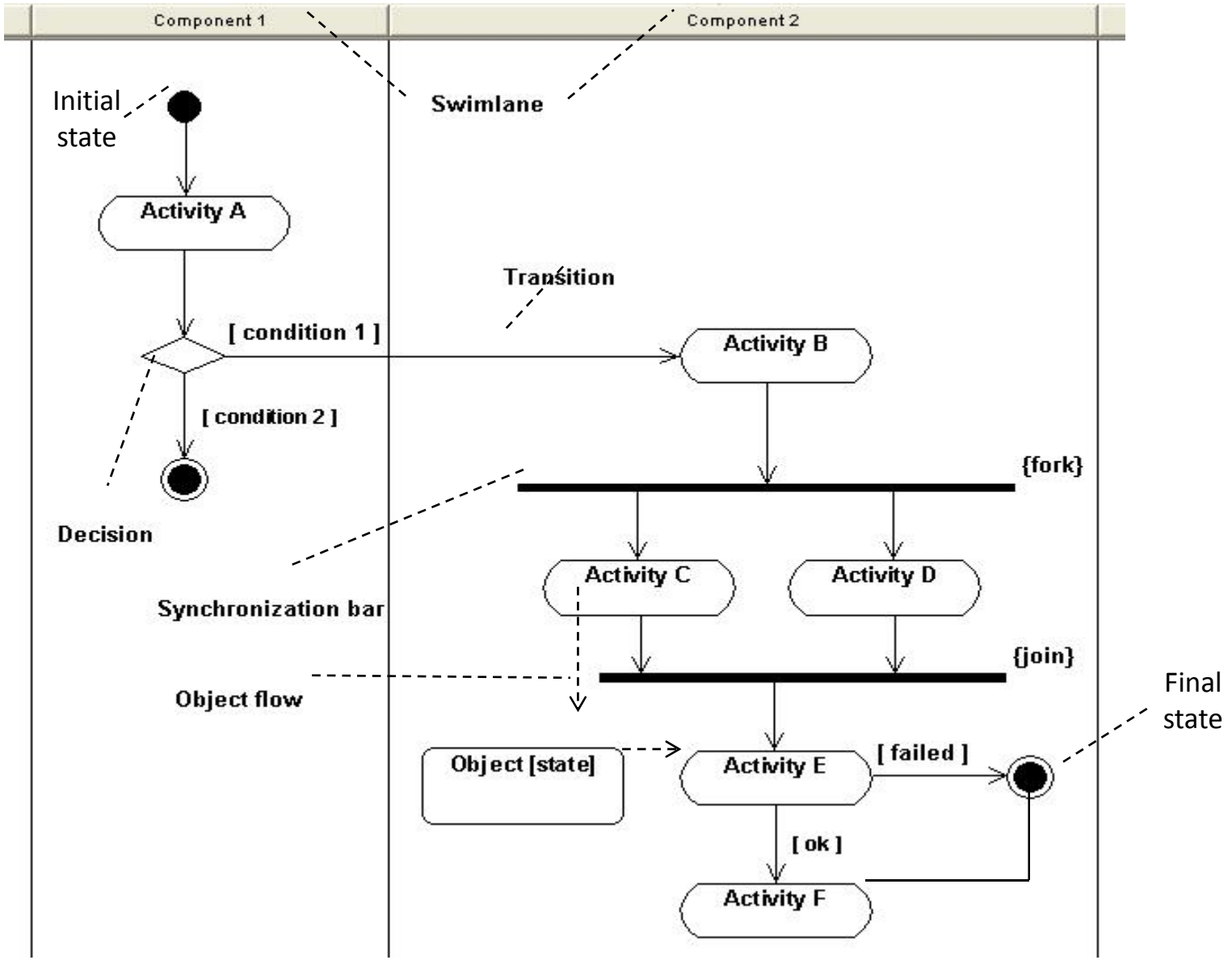
- An activity diagram consists of:
  - activity,
  - transition,
  - decision,
  - synchronization,
  - swimlanes,
  - initial state,
  - and final state.

Activity Diagrams

- Change in 2.0 to be based on token flow semantics
- Used when the primary means of transitioning from one state to another is upon completion of the previous activity not reception of an event
  - An activity diagram can be assigned to either a class, use case or an operation.
  - Useful for describing algorithms
- Each activity has a set of pins
  - Input pins bind input parameters to “local variables”
  - Output pins bind output parameters to “output variables”
  - An activity begins when input data appears on all input pins
  - When an activity completes, there is data on all the output pins
  - Activities are no longer triggered by events

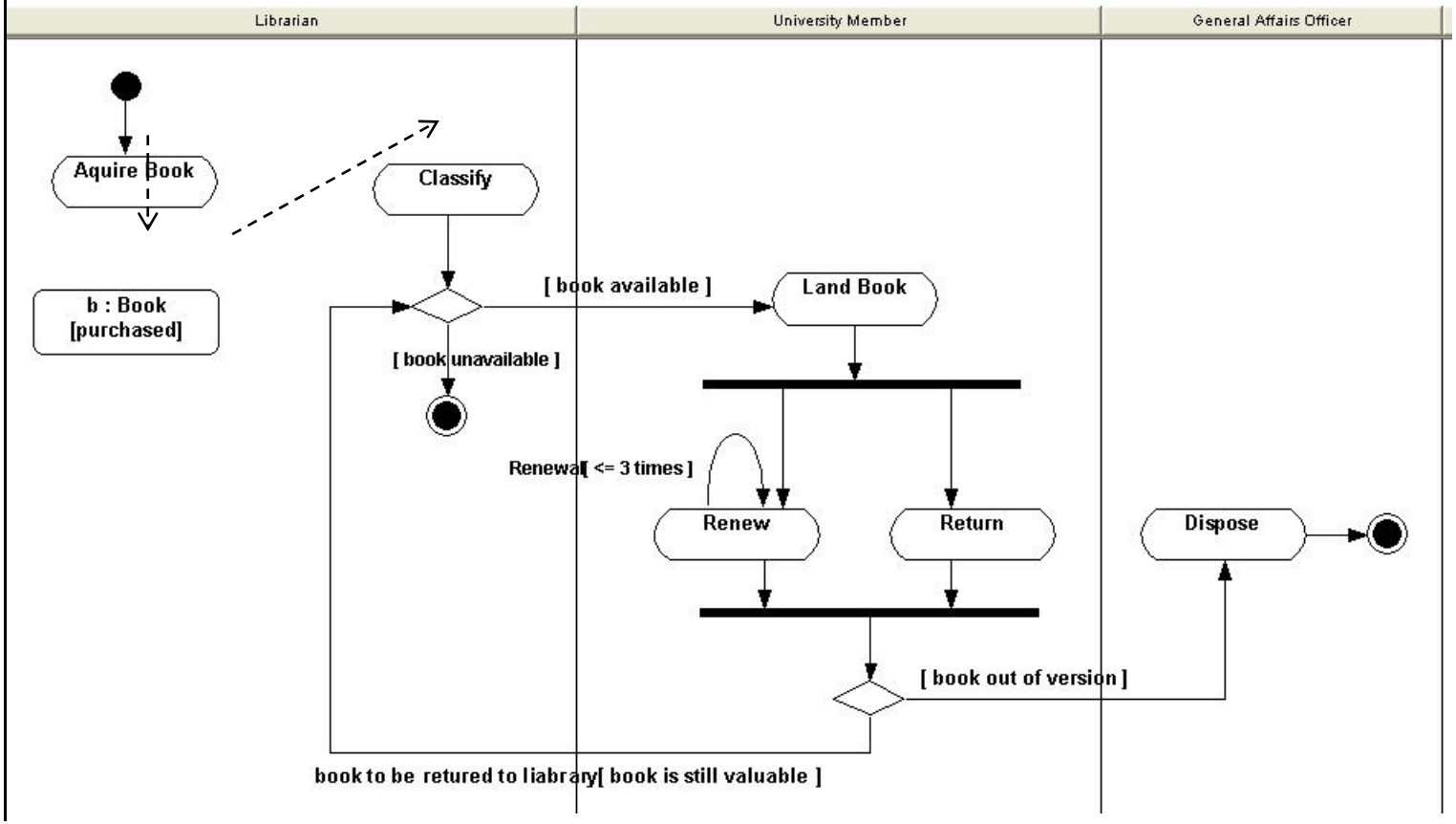
ACTIVITY DIAGRAMS (CONT'D)

- Activity diagram notations



ACTIVITY DIAGRAMS (CONT'D)

- Example: The life of a copy of book.



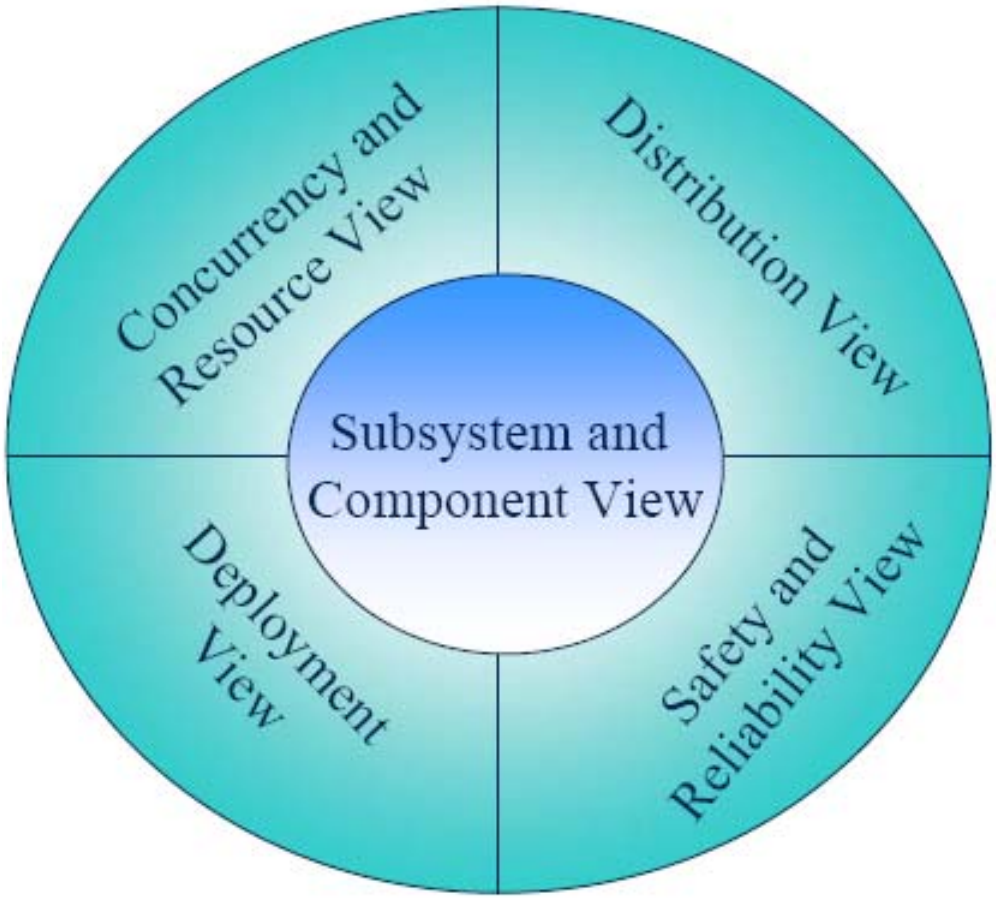
Activity Diagrams (cont'd)

- Activities and activity diagrams are associated either to:
  - a business process,
  - a class (a component),
  - an operation,
  - a use case (to describe the scenarios of a use case),

  that is, you can use activity diagrams to describe these elements.

Architecture

Physical Architectural Views

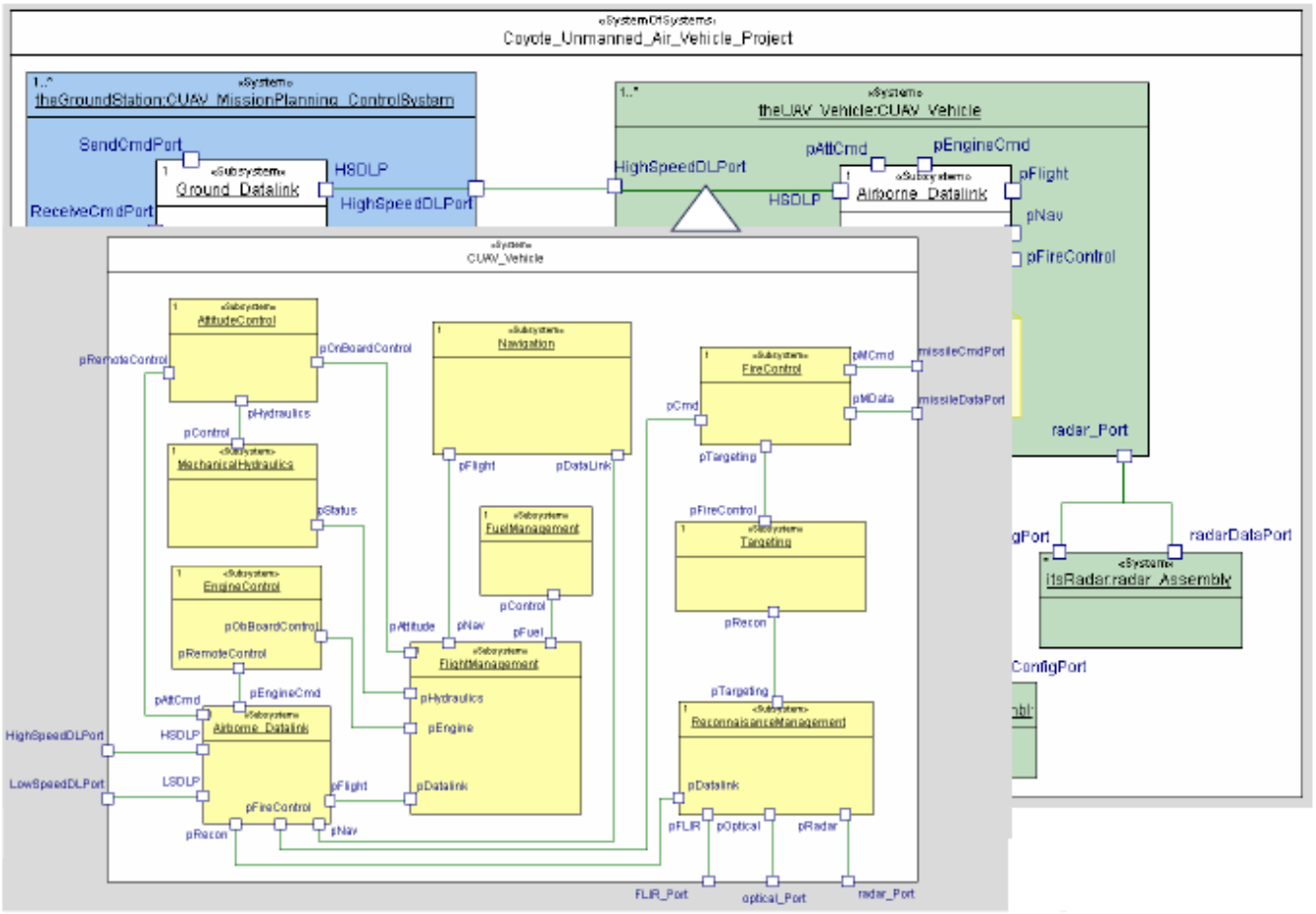


Physical Architectural Views

- Construct architectural design models
  - Subsystem Model
  - Concurrency Model
  - Distribution Model
  - Safety and Reliability Model
  - Deployment Model

- Capture with
  - Class Diagrams
  - Package Diagrams
  - Component Diagrams
  - Deployment Diagrams

Subsystem Architecture



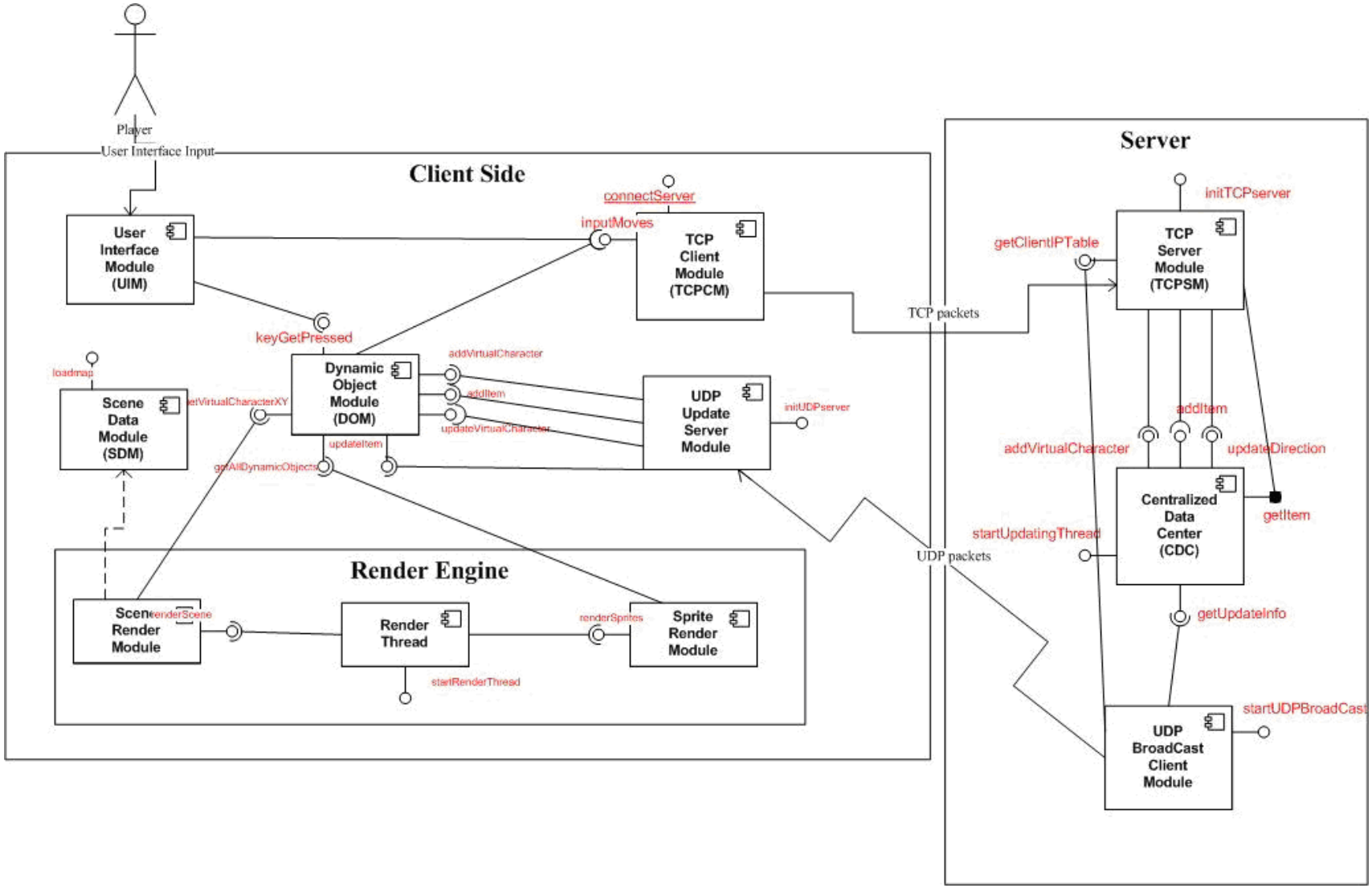
Subsystem and Component View

- A component
  - is the basic reusable element of software
  - organizes objects together into cohesive run-time units that are replaced together.
  - provides language-independent opaque interfaces
  - a metasubtype of (structured) Class
- A subsystem
  - is a large object that provides opaque interfaces to its clients and achieves its functionality through delegation to objects that it owns internally
  - contains components and objects on the basis of common run-time functional purpose
  - a metasubtype of Component

Distribution Architecture

- Distribution model refers to
  - Policies for distribution objects among multiple processors and communication links, e.g.
    - Asymmetric distribution (dedicated links to objects with a priori known location)
    - Publish-Subscribe
    - CORBA and Broker symmetric distribution
  - Policies for managing communication links
    - Communication protocols
    - Communication quality of service management

Distribution Architecture



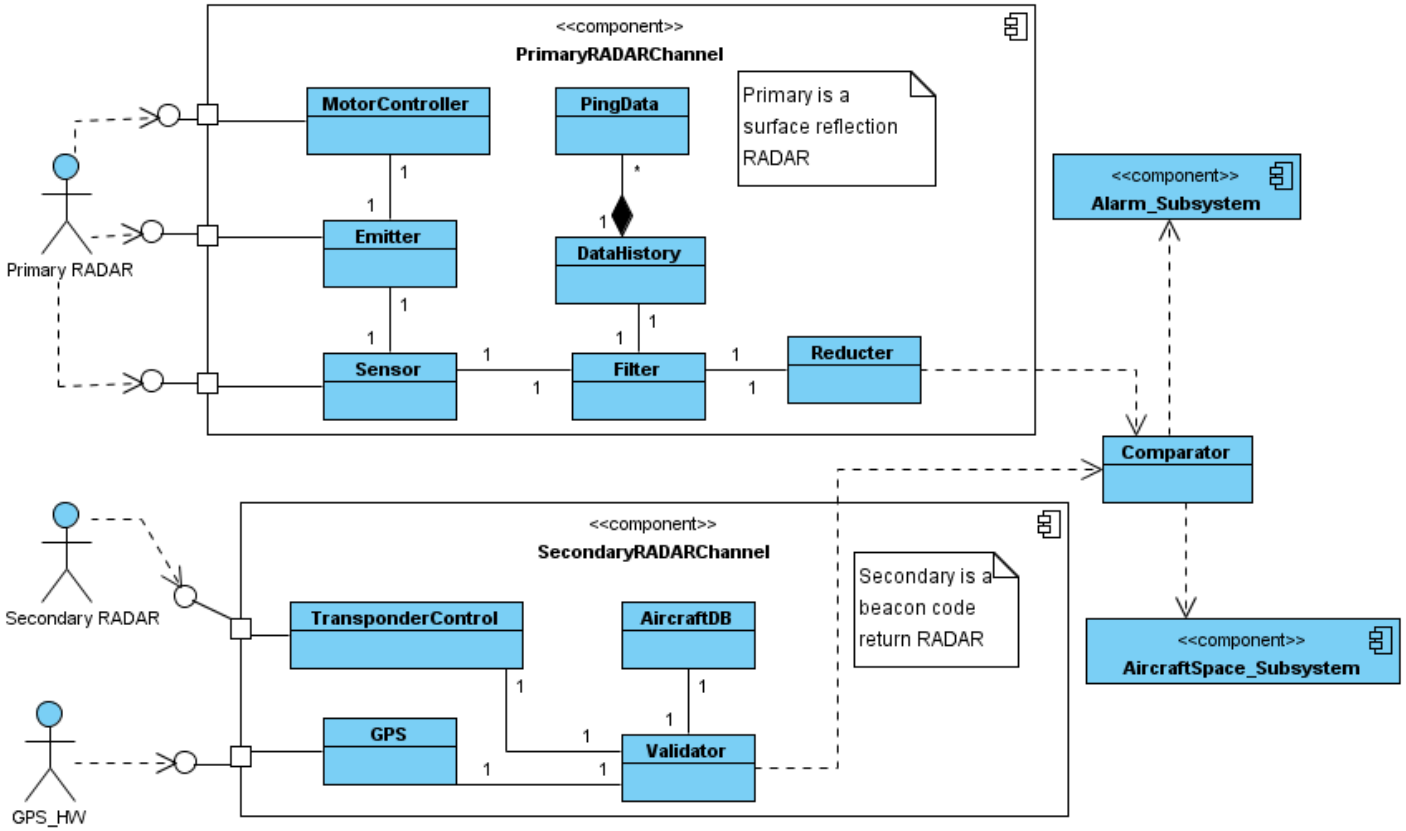
Safety and Reliability Model

- Safety and reliability model refers to the structures and policies in place to ensure
  - Safety
    - Freedom from accidents or losses (risk)
  - Reliability
    - High MTBF (Mean Time Between Failure)
    - Fault tolerance

- Safety and fault tolerance always require some level of redundancy

Safety and Reliability Architecture

Heterogeneous redundancy design



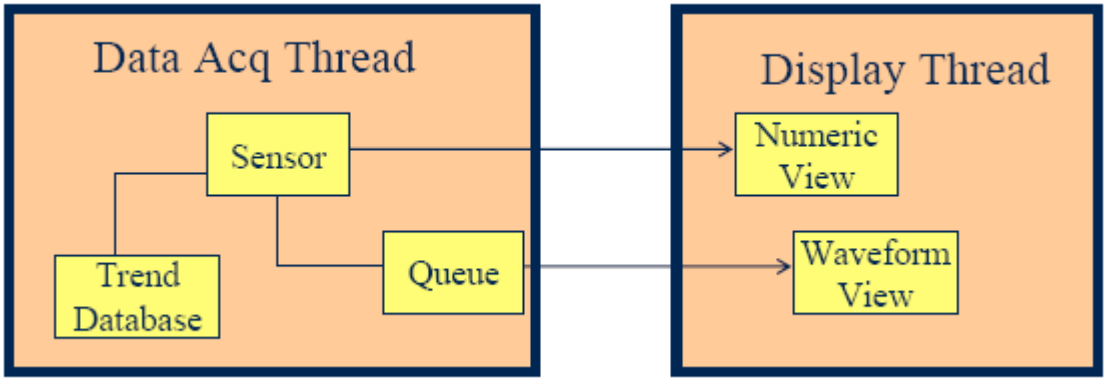
Concurrency Architecture

- Refers to
  - Identification of task threads and their properties
  - Mapping of passive classes to task threads
  - Identification of synchronization policies
  - Task scheduling policies

- Unit of concurrency in UML is the **«active»** object
  - «active» objects are added in architectural design to organize passive objects into threads
  - «active» objects contain passive semantic objects via composition and delegate asynchronous messages to them

Concurrency Model

- Active object is a stereotype of an object which owns the root of a thread
- Active objects normally aggregate passive objects via composition relations
- Standard icon is a class box with heavy line



Active Classes

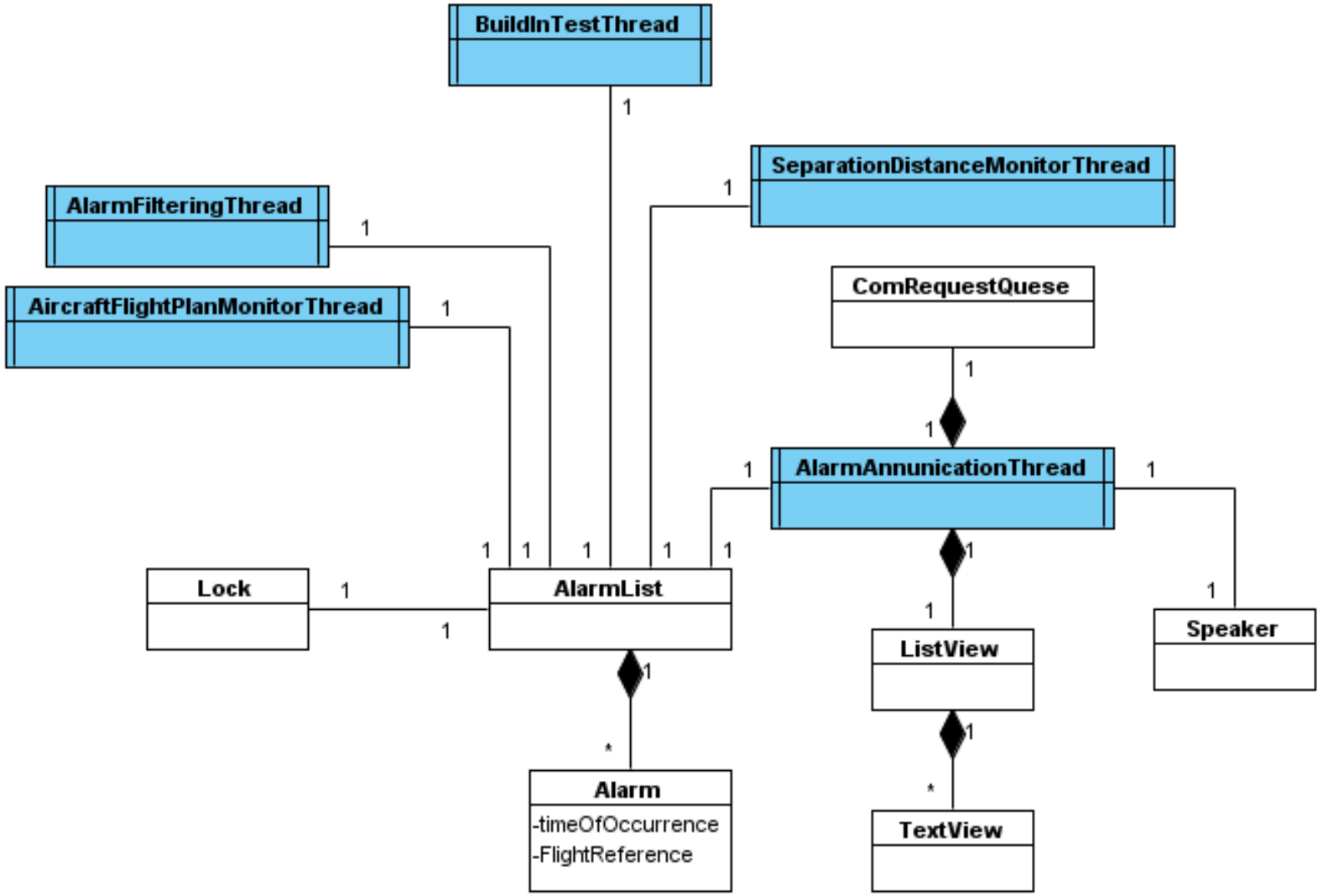
- In UML 1.x the unit of concurrency was called the Active Object, shown with a thick border

UML 1.x Active Object

- In UML 2.0 the notation has changed to double vertical lines, and is called an Active Class

UML 2.0 Active Class

Concurrency View



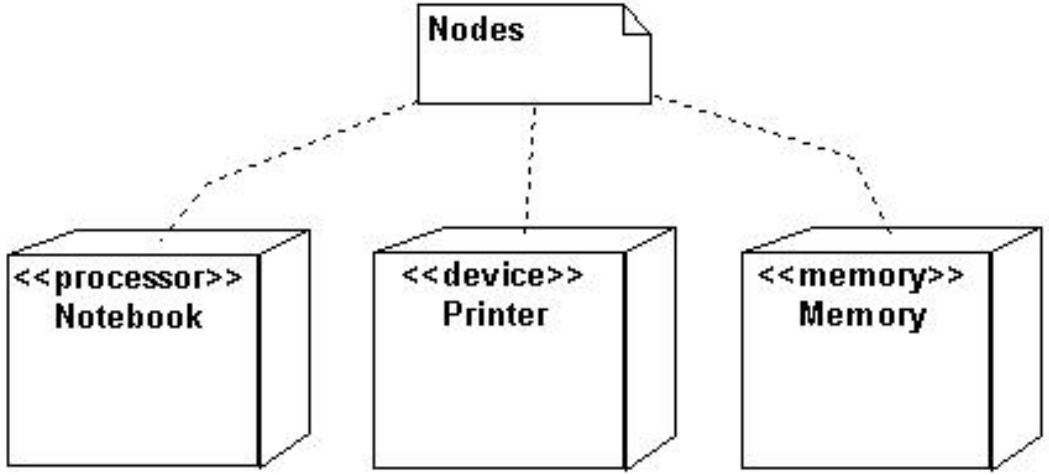
Deployment Architecture

- Maps software components and subsystems to hardware
  - Represents a device as a node
  - Iconic stereotypes are common

- Identifies physical connections among devices
  - May be buses, networks, serial lines, etc
    - Two primary strategies
  - Asymmetric
    - Design-time mapping of software elements to HW
  - Symmetric
    - Dynamic run-time mapping of software elements to HW

Deployment Diagrams

- A **Deployment Diagram** shows the configuration of runtime process nodes. It also addresses the static deployment view of a system.
- Representation of nodes.



Deployment Diagrams (cont'd)

- Examples:
  - Nodes and components.
  - Connections.