SEG3101 (Fall 2010)

Behavioral Modeling

Gregor v. Bochmann, University of Ottawa

Based on Powerpoint slides by Gunter Mussbacher
with material from:
K.E. Wiegers, D. Leffingwell & D. Widrig, M. Jackson, I.K. Bray, B. Selic,
Volere, Telelogic, D. Damian, S. Somé 2008, and D. Amyot 2008-2009

Table of Contents

- An introduction to modeling is already given in the slides on "Introduction to Requirements Analysis and Specification"
- Structural modeling is discussed in separate slides.

- Here we discuss a four selected approaches for modeling behavioral aspects of requirements. For the last three approaches, we discuss the UML notations in detail.
  - Structured Analysis
  - UML Activity Diagrams, also Use Case Maps (see separate slides)
  - UML Sequence Diagrams
  - UML State Diagrams
- We also give an overview of UML version 2 and discuss for each of these approaches, how a model can be used for analysis (validation, verification – functional and non-functional) and implementation development.
- Get the habit of analysis – analysis will in time enable synthesis to become your habit of mind. [1]

[1] Frank Lloyd Wright (1867 - 1959)


WALLY, ARE YOU DONE WITH YOUR PROJECT YET?
I'LL BE DONE NEXT WEEK.
YOU'VE SAID, "NEXT WEEK" FOR SEVEN WEEKS IN A ROW. WHAT MAKES YOU THINK I'M GOING TO BELIEVE IT THIS TIME?
THE FIRST SIX TIMES?
Dilbert.com DilbertCartoonist@gmail.com
10-12-09 © 2009 Scott Adams, Inc./Dist. by UFS, Inc.

Structured Analysis

Structured Analysis

- Data-oriented approach
  - Based on analysis of information flow
- Models
  - Dataflow Diagram (DFD) – flow of information in system
  - Entity Relationship Diagrams (ERD) – describe data
  - Data Dictionary (DD) – define all data elements
  - State Diagram – describe state-based behavior
- Mainly used for information systems
  - Extensions have been developed for real-time systems
- Analysis consists of modeling current system (can be a manual system)
  - New system derived from understanding current system
  - What if there is no current system?

Popular Approaches (at least once upon a time…)

Structured Analysis is historically important. Here are some of the more popular versions:

- Structured Analysis and Design Technique (SADT) by Doug Ross
- Structured Analysis and System Specification (SASS) by Yourdon and DeMarco
- Structured System Analysis (SSA) by Gane et Sarsan
- Structured Systems Analysis and Design (SSADM)
- Structured Requirements Definition (SRD) by Ken Orr
- Structured Analysis / Real Time (SA/RT) by Ward and Mellor
- Modern Structured Analysis by Yourdon

Structured Analysis – Methodology (SASS Steps)

1. Analysis of current physical system
   - DFD to show current data flow through the organization
   - Shows physical organizational units or individuals (could be called “agents”)
2. Derivation of logical model (existing problem domain)
   - Logical functions replace physical agents
3. Derivation of logical model of proposed new system
   - DFD modified to reflect system boundaries and updated organization of the environment
4. Implementation of new system
   - Some architectural alternatives are considered

Example: Yacht Race Results – Analyze current system (1)

- Elicitation plan:

Interview with primary contact, sailing secretary of Dartchurch sailing club, Dave Rowntree. To be held at their place on 6/6/00.

From phone conversation, we know they are after a 'cheap' (!) PC-based system to calculate results of yacht races.

Establish basic problem.

Establish role of DR – is anyone else involved?

Investigate financial aspects.

How (in outline !) does it work now?

What are the current problems?

What are they looking to achieve?

- Elicitation notes:

Interview with David Rowntree, sailing secretary, Dartchurch sailing club (DSC) 6th June.

Basic scenario - They have an old spare PC, reckons they could use it to help work out race results. Currently all done by hand.

DR well experienced in the process (frequently worked out results, etc. himself). Probably knows as much as anyone here but Jim Lock bit of a technical whizz if needed.

Cagey about money side - really wants to know "how much it would cost?". Advised that after today's session, we'll give quotes for basic system and "all singing" one.

Source: Bray, 2004

Example: Yacht Race Results – Analyse current system (2)

Current system:-

Sailors enter boats on race sheet prior to race (at least 1 hour).

The "OOD" (Officer of the Day!) takes race sheet out on "committee boat" to start race (or sheets - one per race and they often have several starting one after the other). Some "race officers" (proper name for OOD!) allow entries at the start (not really allowed but it's up to them). At end of race RO enters finish times of boats on race sheet. (No start time per boat - all start at same time.) Details on sheet are:- boat name, sail number, class, helm name (the helm is the one sailing it). Mostly optional - only really need sail number and class.

Back at club, RO works out results and enters on results sheet (see attached copy). (May leave it to class captains (CCs).) Easy for OD (one design classes) tedious for handicap classes (which are?!). Some races for one type of boat, others for different types! For 2nd sort, finish times adjusted by handicaps. For ODs, just put in finishing order. (So don't actually need finish times for those (!) - just positions.)

Handicap is a multiplier for elapsed time (ET = finish time – start time). Based on RYA (Royal Yachting Association) lists (copy on race notice board). RO can refer to list but usually knows. Example - start at 12:00, finish at 13:14:22 (to nearest second, note!) - elapsed time = 1:14:22. Convert to seconds - 4462, then elapsed time = finish time x 1000/handicap. (And there's another, similar, way of doing it - worry about that later!)

Results are written up on results sheet and posted on race notice board. May be right after race (more or less) or may be a week later (if RO is lax) (avoiding delays is a potential plus here).

Boats may not finish a race - could be DNF (did not finish) or RTD (retired). (What's the difference? - we needn't worry!)

Quantities:- about 200 boats in club. Each class, 2 or 3 races per week, about 10 boats per race (max, say 30). 7 classes of boat but some classes are "open". (Same as handicap classes!) Different classes of boat may enter handicap races. (Eh?????) So, some classes of race are for one class of boat (the OD classes), others are for several classes of boat (handicap classes). Actually about 20 classes of boat. (7 classes of race!!)

Interested in system printing out race entry forms with race details. ROs can then enter results into system and it will print out result sheets.

(There are some "oddities" regarding the race results sheet:-

- Elicitation notes (suite)

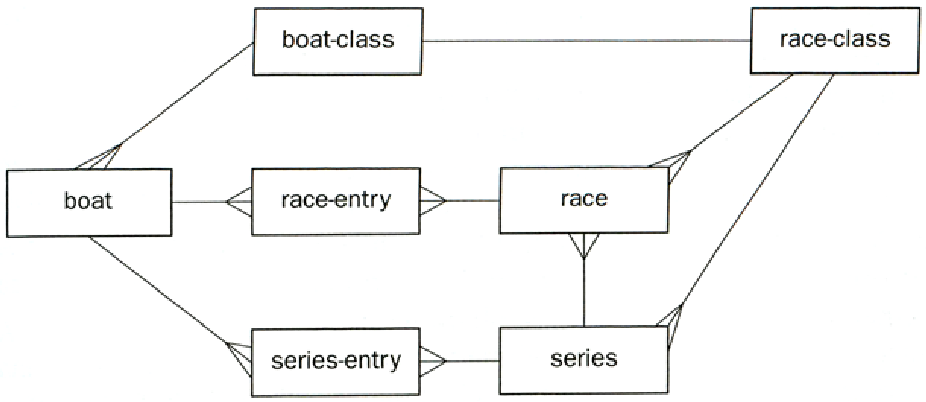
"Recall no." is just another name for "sail number"
The "PYN" column is used for PY and TMF handicaps
The corrected time is seconds (not hrs, mins and secs)
The "overall" column isn't used.

Otherwise, it's just fine!)

And there are series of races but out of time - follow up next week 10:00 Thursday - same place.

Example: Yacht Race Results – Analyse current system (3)

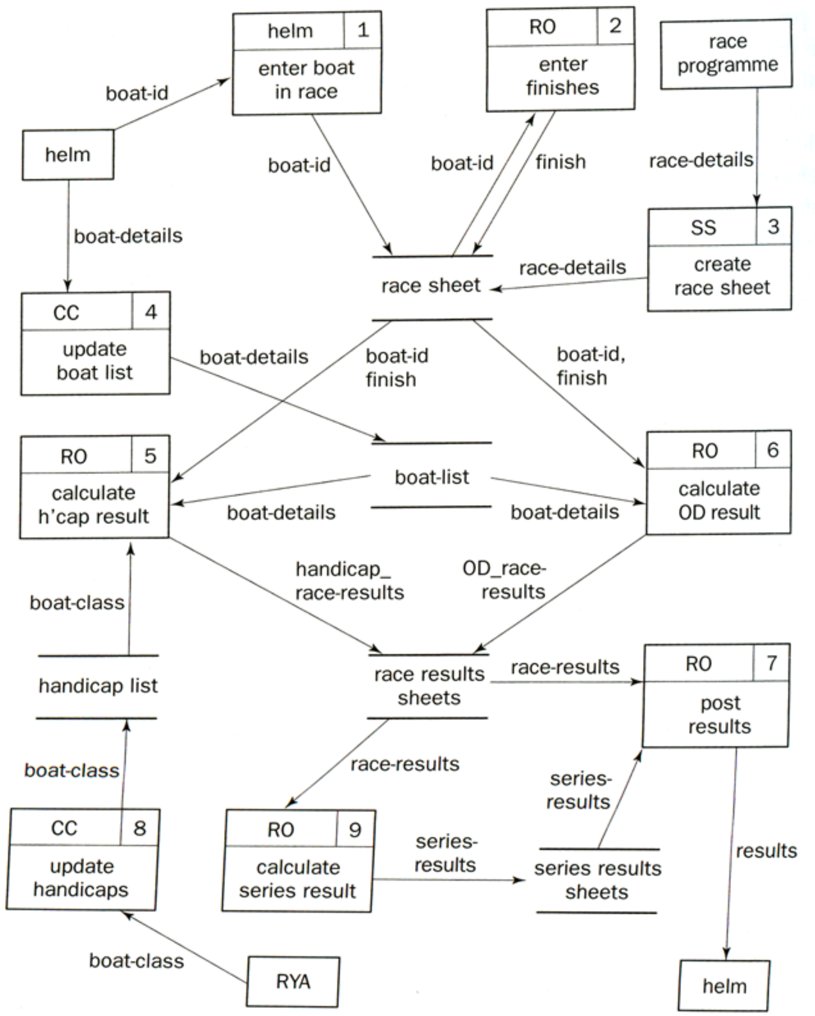
Domain model (ERD)



Data Dictionary (DD)

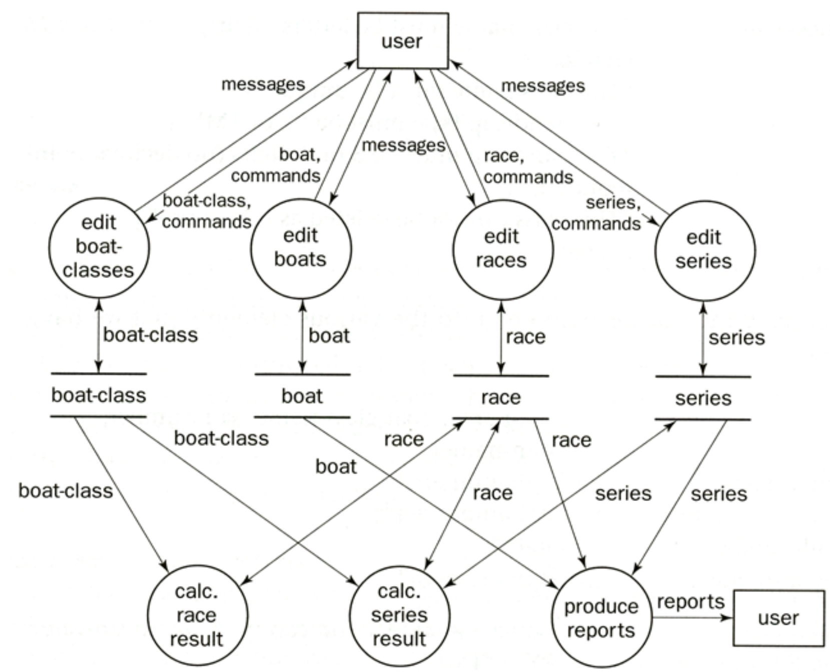
helm-name	::= {alphanumeric}^{25};
	::= (* the person who sails the boat *)
boat-class	::= boat-class-name, handicap;
	::= (* a particular type of boat *)
handicap	::= handicap-type, handicap-value;
	::= (* a way of compensating for different inherent speed when different boat classes race against each other *)
handicap-type	::= "PY" | "TMF";
handicap-value	::= ^{3}{digit}^{4};
race-details	::= race-class-name, race-date, start-time, [race-name] [course];
race-class	::= race-class-name, race-class-type, [handicap-type, minimum-handicap, maximum handicap];
	::= (* an indication of the boat-classes) (that may enter a race *)
race-class-name	::= boat-class-name | ^{1}{alphanumeric}^{25};
race-class-type	::= "one-design" | "handicap";
race-date	::= day, ":", month, ":", year;
start-time	::= hour, ":", minute;

Example: Yacht Race Results – Analyse current system (4)



Left: SSADM Diagram showing data flow (arrows), functions (boxes with indication of agents), and stored data (between two horizontal lines). The simple boxes are agents or external data.

Below: A more modern notation is shown. Here functions are presented in circles. A different system structure is adopted in this diagram.

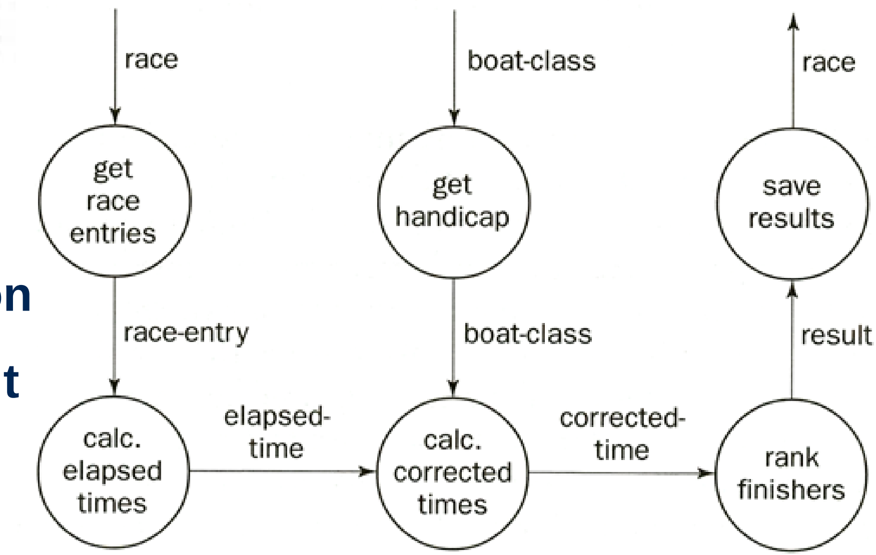


Example: Yacht Race Results – Analyse current system (5)

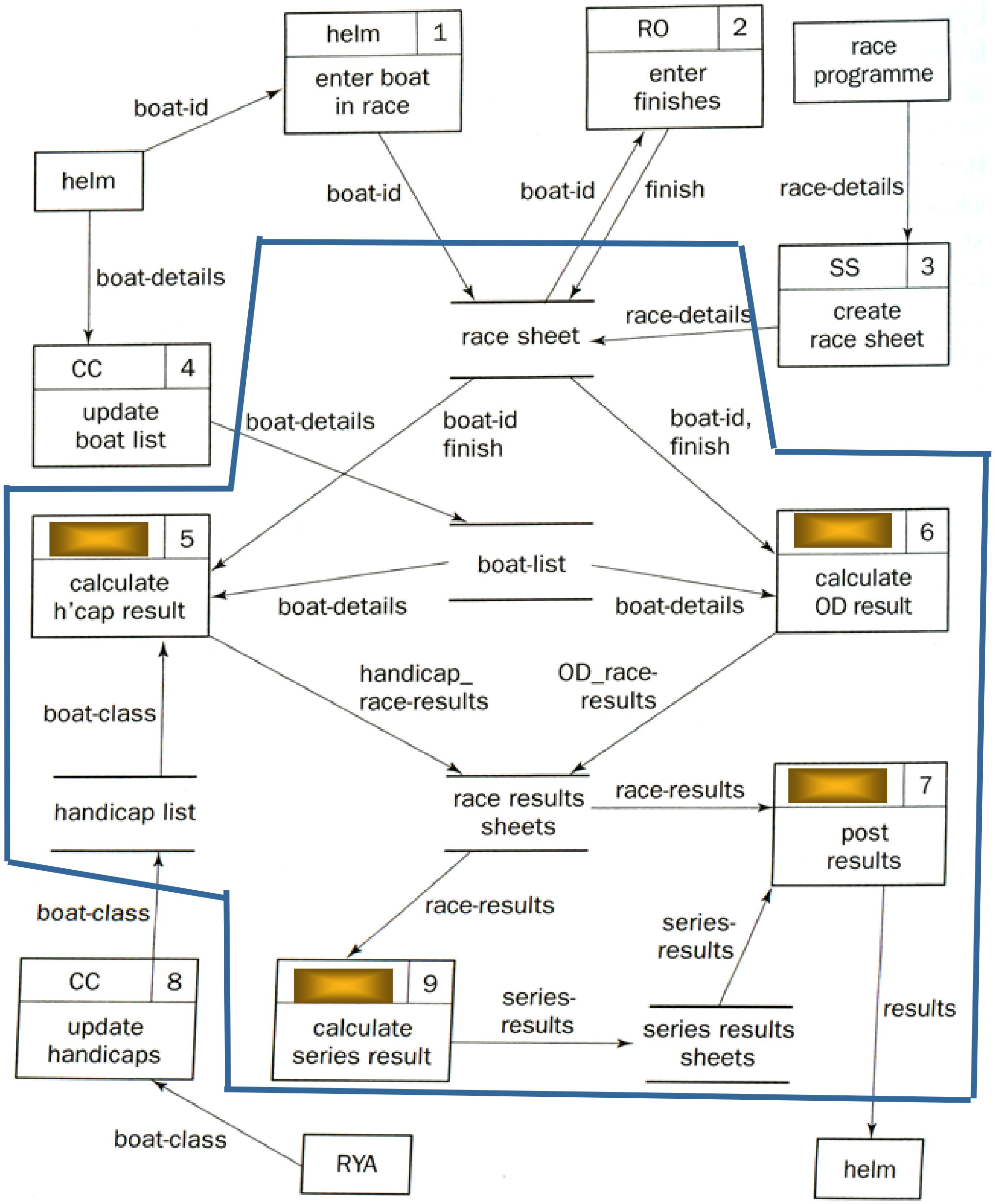
Definition of the function Calculate-handicap-result

```
calculate-handicap-result
get race details
for each race entry
        case finish of
            finish-time
                if handicap-type = PY then
                    corrected-time := elapsed-time × 1000 / handicap-value
                else (*TMF handicap*)
                    corrected-time := elapsed-time × handicap-value
                insert into sorted list by corrected-time
            other (* DNS etc. *)
                add to end of sorted list
```

Refinement of the function Calculate-series-result



Example – Define logical view and new system scope



Notes:

The blue line defines the boundary of the system-to-be with its environment. The yellow functions are performed by the system.

The diagram defines implicitly the system interfaces.

It also suggests an internal design for the system-to-be. In this case, no revision of the system structure has been proposed for the new system.

Structured Analysis – Problems

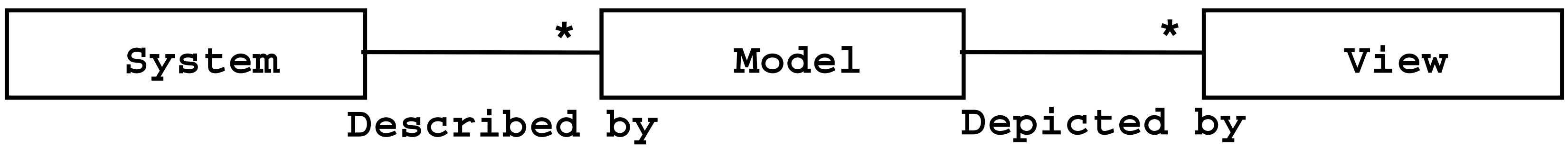
- Overemphasis on modeling (there's more to analysis!)
- Models the preexisting solution system (rather than the application domain)
- Essentially process-based models (encourages structural model of the preexisting system)
- Difficulty in integrating DFD and ERD models
- No explicit mention of requirements!
  - Implicit assumption that the preexisting system already meets the requirements apart from not being computer-based!
  - SSADM[1] eventually added the Problem/Requirements List (PRL)
- This assumption is carried through into design (new system inherits its basic structure from the preexisting system)
- Lack of a truly behavioral specification
  - Where are the process descriptions, à la SDL?

[1] Structured System Analysis and Design Methodology

Introduction to the Unified Modeling Language (UML)

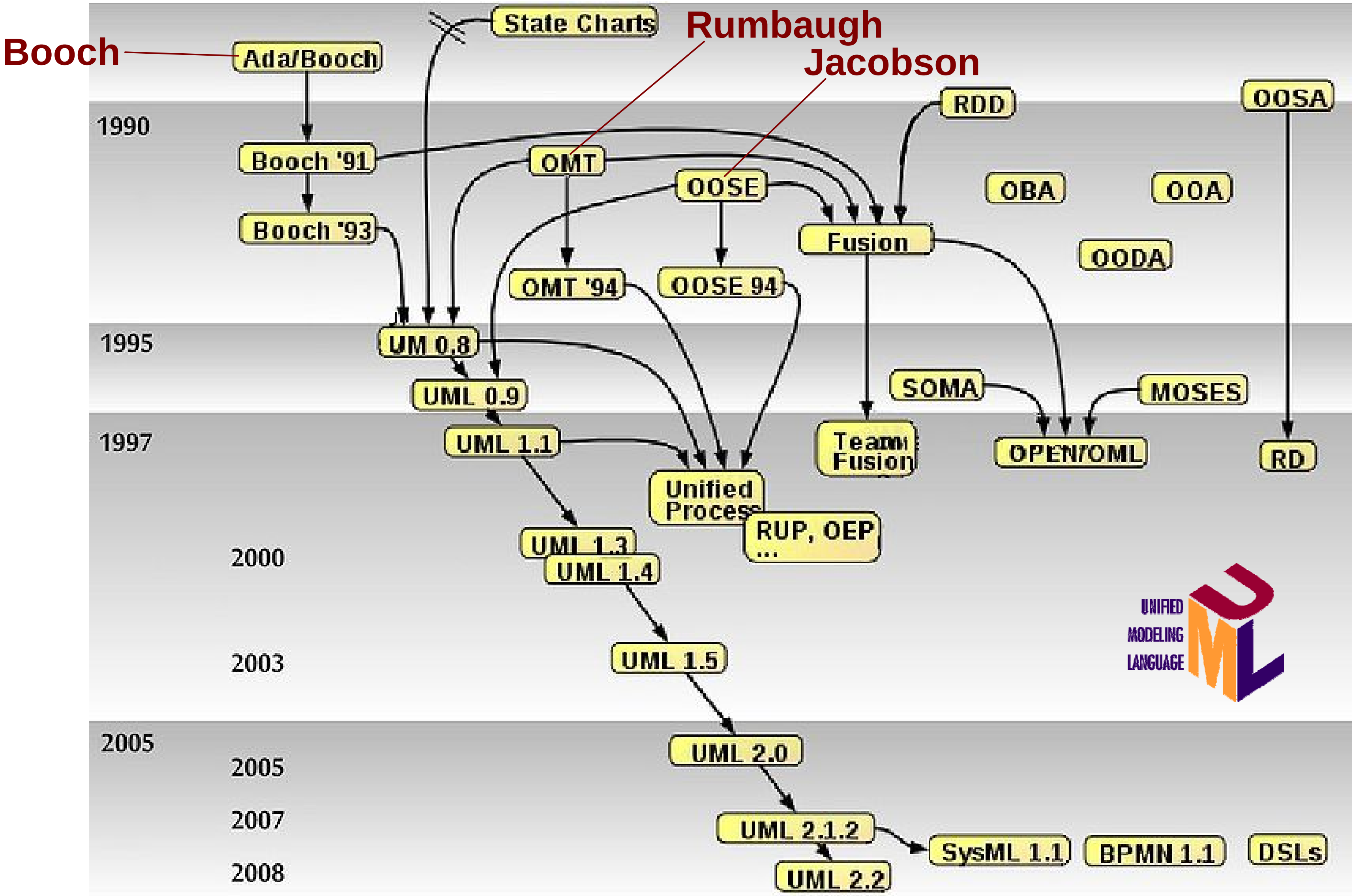
Systems, Models, and Views

- A model is an abstraction describing a subset of a system (filtering out unimportant details)
- A view depicts selected aspects of a model
- A notation is a set of graphical and/or textual rules for depicting views
- Views and models of a single system may overlap each other – examples:
  - System: Aircraft
  - Models: Flight simulator, scale model
  - Views: All blueprints, electrical wiring, fuel system



- Programming language vs. model

History of UML



Source: http://en.wikipedia.org/wiki/Unified_Modeling_Language

UML 2.x

- Object Management Group (OMG) standard
  - Version 2.0 released in 2005
  - Current version is 2.3 (May 2010)
  - http://www.omg.org/uml/
- Some key points (new in Version 2)
  - Restructuring of the metamodel
    - Infrastructure (semantics) and superstructure (notation)
  - New or modified diagrams
  - Simpler and more powerful profile mechanisms
  - Diagram exchange format (between UML tools)
  - OCL 2.0 (Object Constraint Language – for input/output assertions, invariants, etc. (resembles first-order logic)

Thirteen Diagram Types in UML 2.x

- Few changes
  - Use case, object, package, deployment diagrams
- Major improvements but less relevant to requirements engineering
  - Component and communication (collaboration) diagrams
- Major improvements and interesting for requirements engineering
  - State machine (integration of SDL as a profile), class, activity (complete re-write of the semantics), and sequence diagrams
- New
  - Timing, interaction overview, composite structure diagrams

Classification of Diagram Types

- According to UML Reference Manual
  - Structural
    - Class, object, composite structure, component, and use case diagrams
  - Dynamic (that is, describing dynamic behavior)
    - State machine, activity, sequence, communication, timing, and interaction overview diagrams
  - Physical
    - Deployment diagrams
  - Model Management
    - Package diagram

Most Relevant for Requirements Engineering

- Use case diagram
  - Use cases structuring
- Class diagram
  - Domain modeling
- Activity diagram (concepts much related to concepts of Use Case Maps)
  - Workflow and process modeling
- Sequence diagram
  - Modeling of message exchange scenarios
- State machine diagram
  - Detailed behavioral specification

Activity Diagram

UML 2.x Activity Diagrams

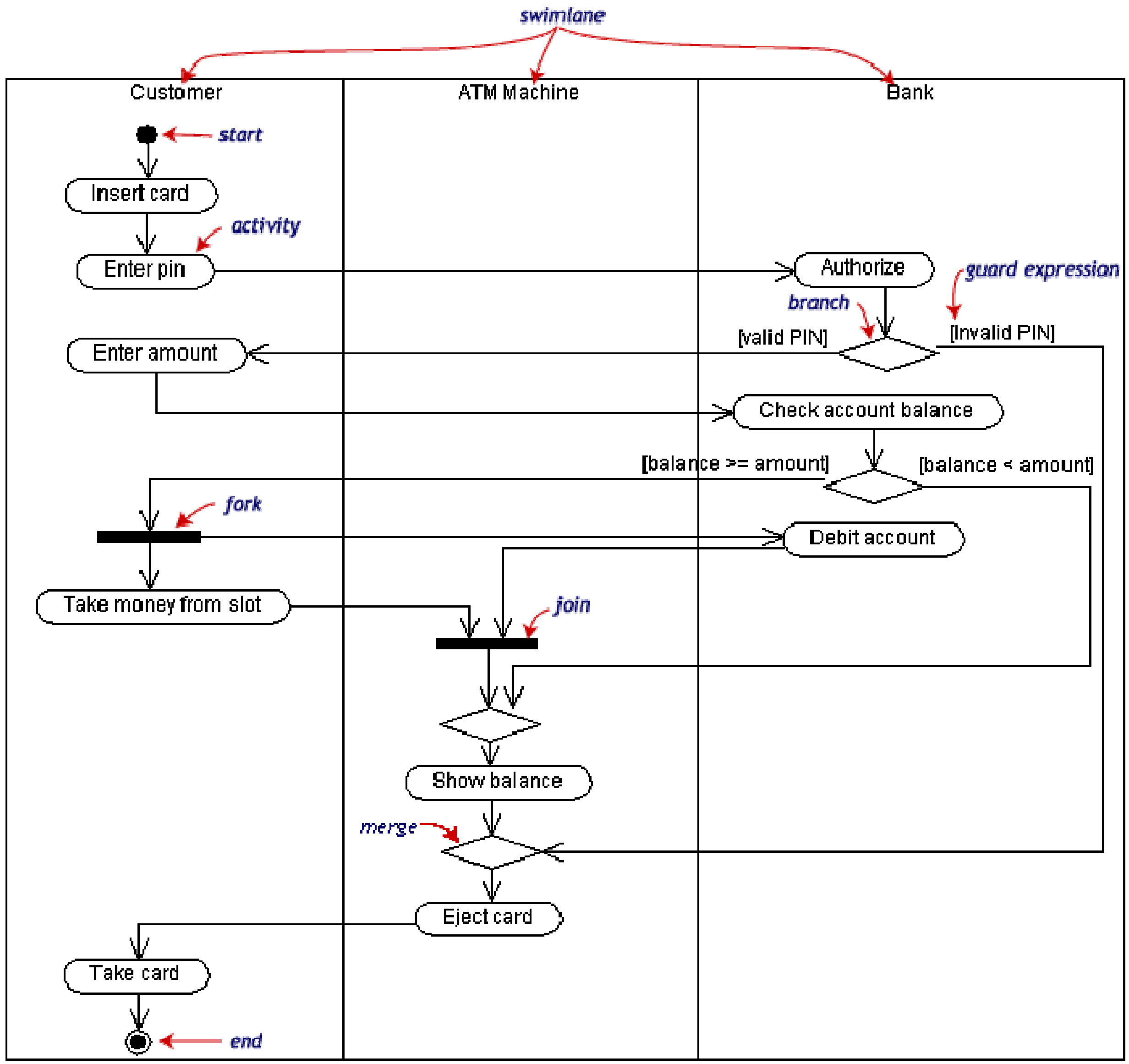
- An Activity Diagram models behavior in terms of sub-activities (actions) and data flow. Sometimes, the flow is simply control flow (a token without data).
- Actions are initiated because
  - The required input data (or control) tokens become available
    - because other actions finish executing, or
    - the action is the initial action and all required input data has been provided by the environment in which the activity diagram is executed
  - Some interrupting event occurs and the normal flow of control is changed
- The behavior of an action may be defined
  - Informally, by its name and an explanation
  - By input and output assertions about input and output data objects and the "state" of the system
  - By defining its behavior by a separate Activity Diagram

Activity Diagrams in UML version 2

- In UML version 1, the way the semantics of Activity Diagram was described, was confusing. (It was based on State Machines, which is not natural, and nobody liked it).
- In UML version 2, the meaning of Activity Diagrams has been explained (in a completely different manner). It is now much more easier to understand, and it is based on the tokens of Petri nets (which are used for modeling control or data flow tokens).
- There are also some interesting additions to the notation
  - Terminal node types, pins, partitions, exceptions

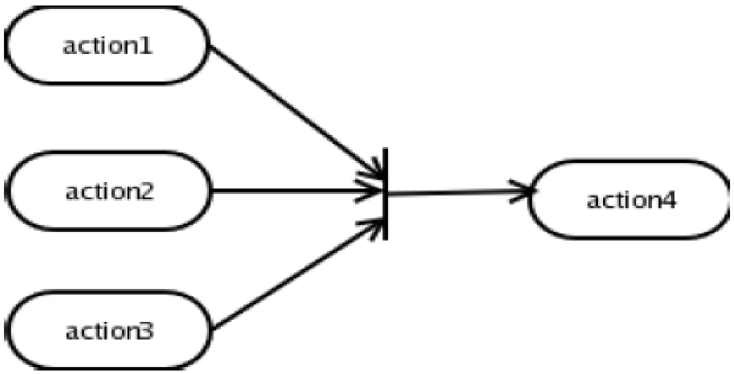
Basic Notational Elements of Activity Diagrams

- Describe the dynamic behavior of a system as a flow of activities (workflow)
- Flow
  - Sequence
  - Alternative
  - Parallel
- Note: in this diagram, the data flow objects are not shown. They may be shown as boxes on the control flow lines.

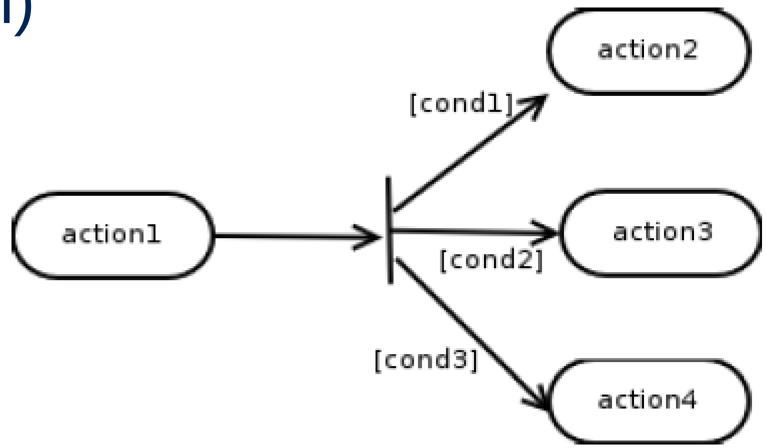


Action Flow – Join and Fork

- Join: action4 starts after action1, action2, and action3 have completed (synchronization)

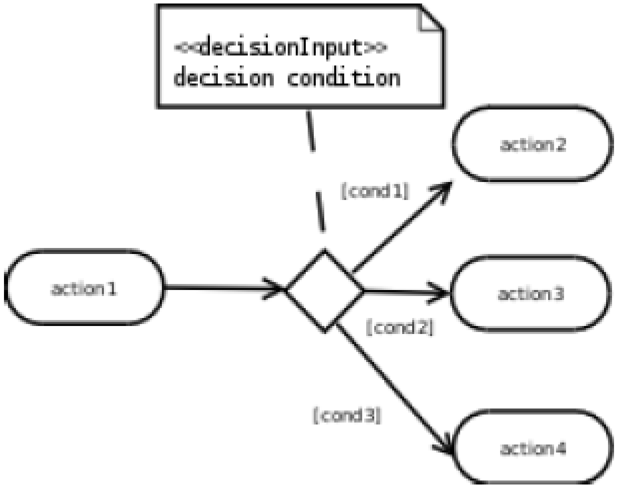


- Fork: flow continues to action2, action3, and action4 after action1 (concurrent execution)

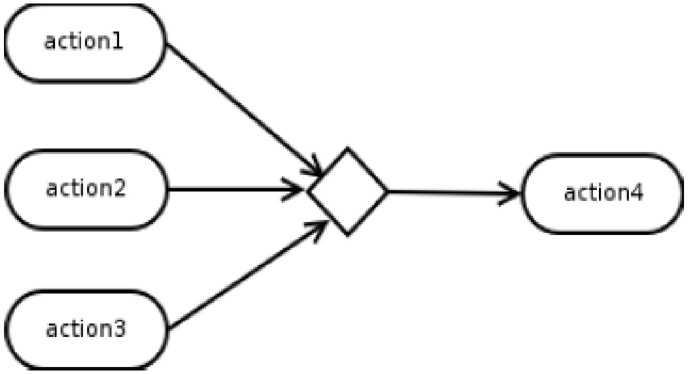


Action Flow – Decision and Merge

- Decision: action2 or action3 or action4 occurs after action1 depending on condition

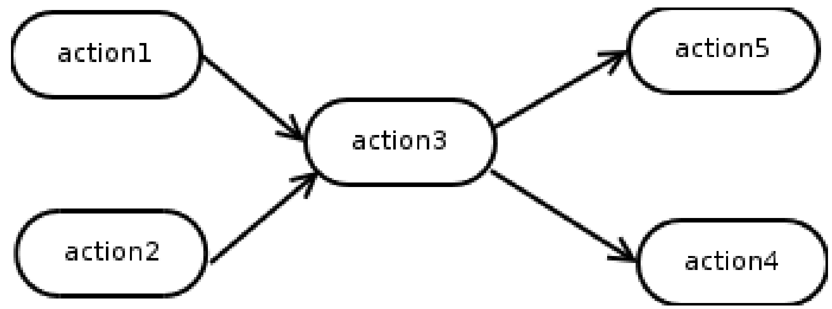


- Merge: flow continues to action4 after either action1 or action2 or action3 (no synchronization)

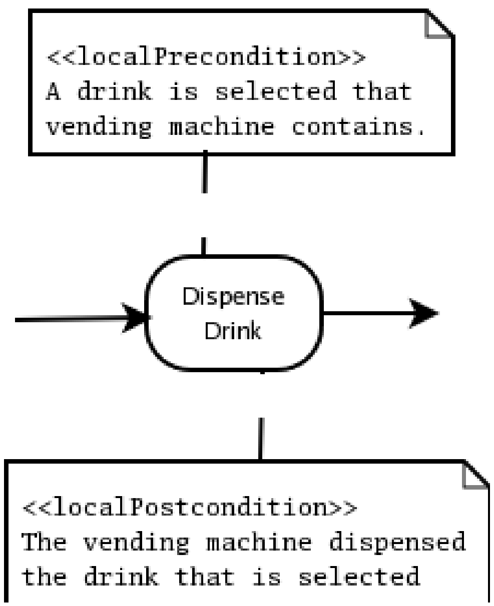


Action Flow – Implicit Join and Implicit Fork

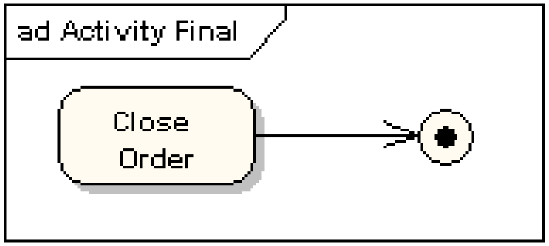
- Action3 starts after action1 and action2 (implicit join) and then action4 and action5 can start (implicit fork)

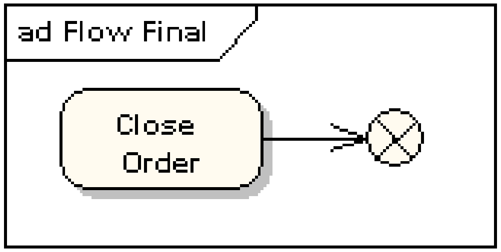


- Pre- and postconditions may also be assigned

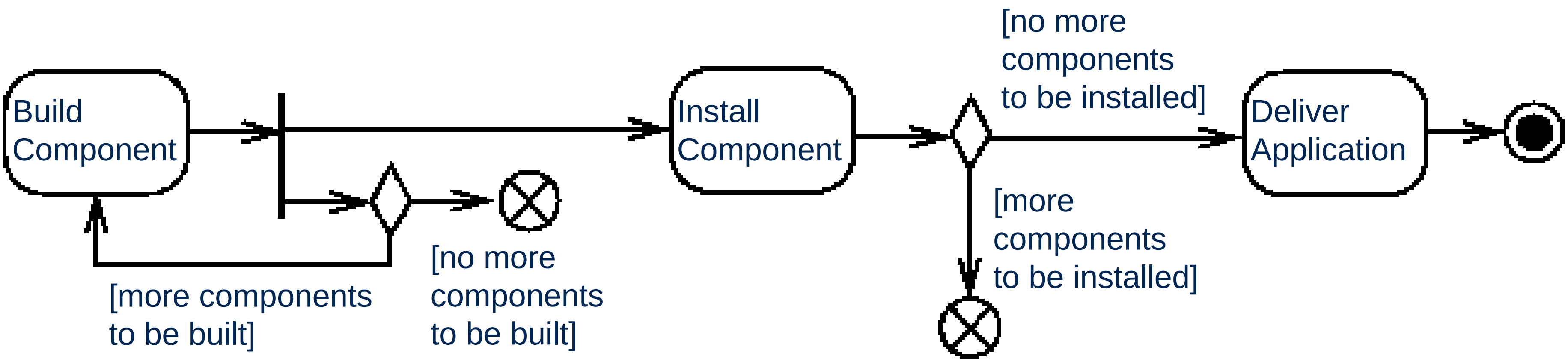


Two Terminal Nodes

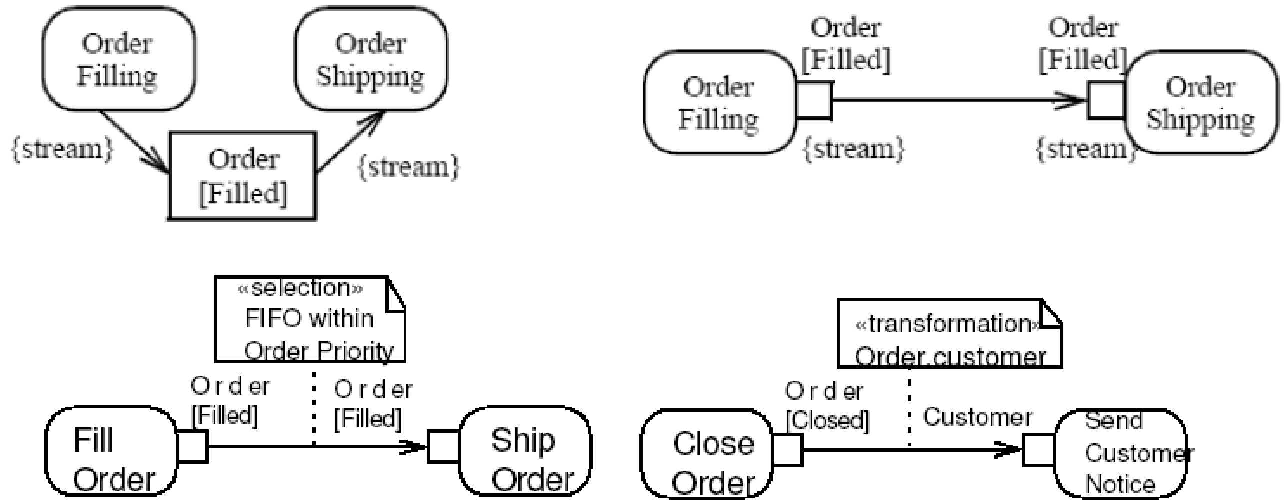




- Final activity node (left)
  - Terminates the entire activity (and returns to the parent one, if any)
- Final flow node (right)
  - Only terminates the flow (the activity continues if there are unfinished parallel flows)

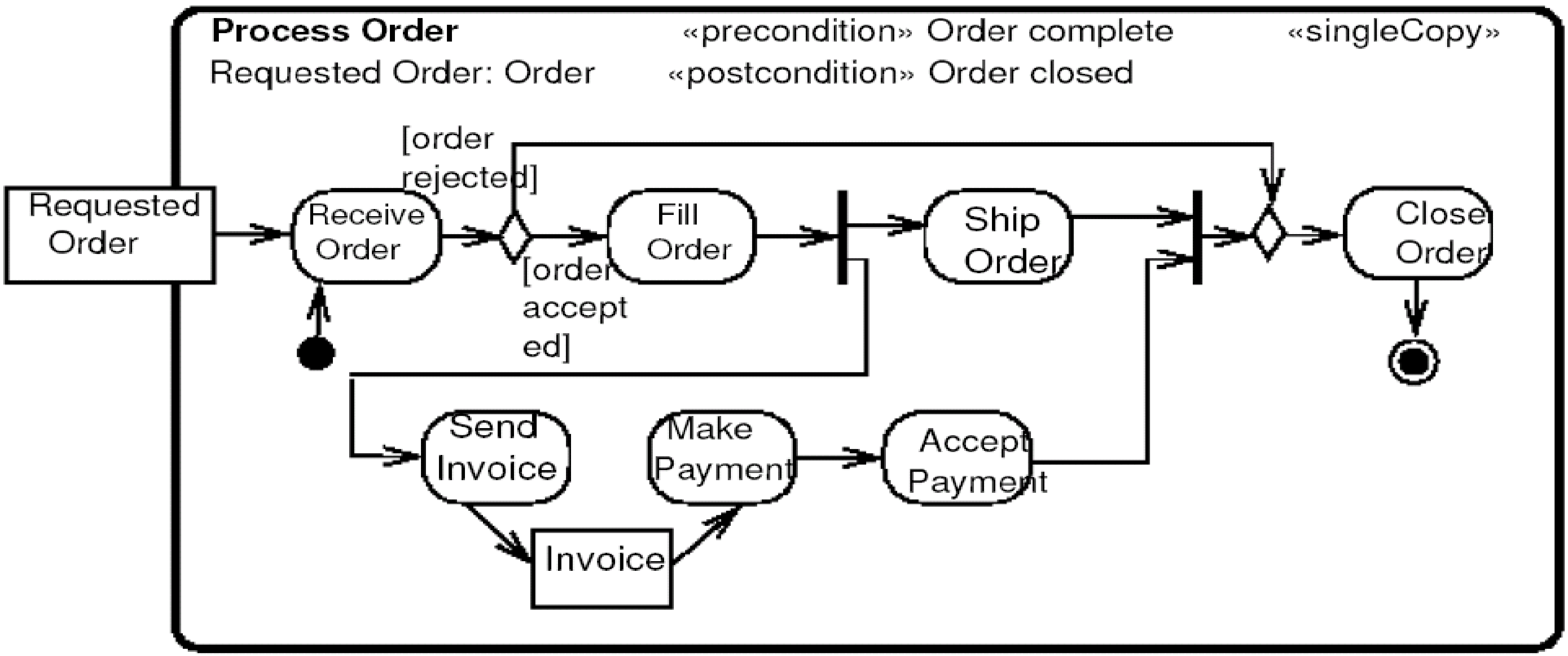


Object Flow with and without Pins



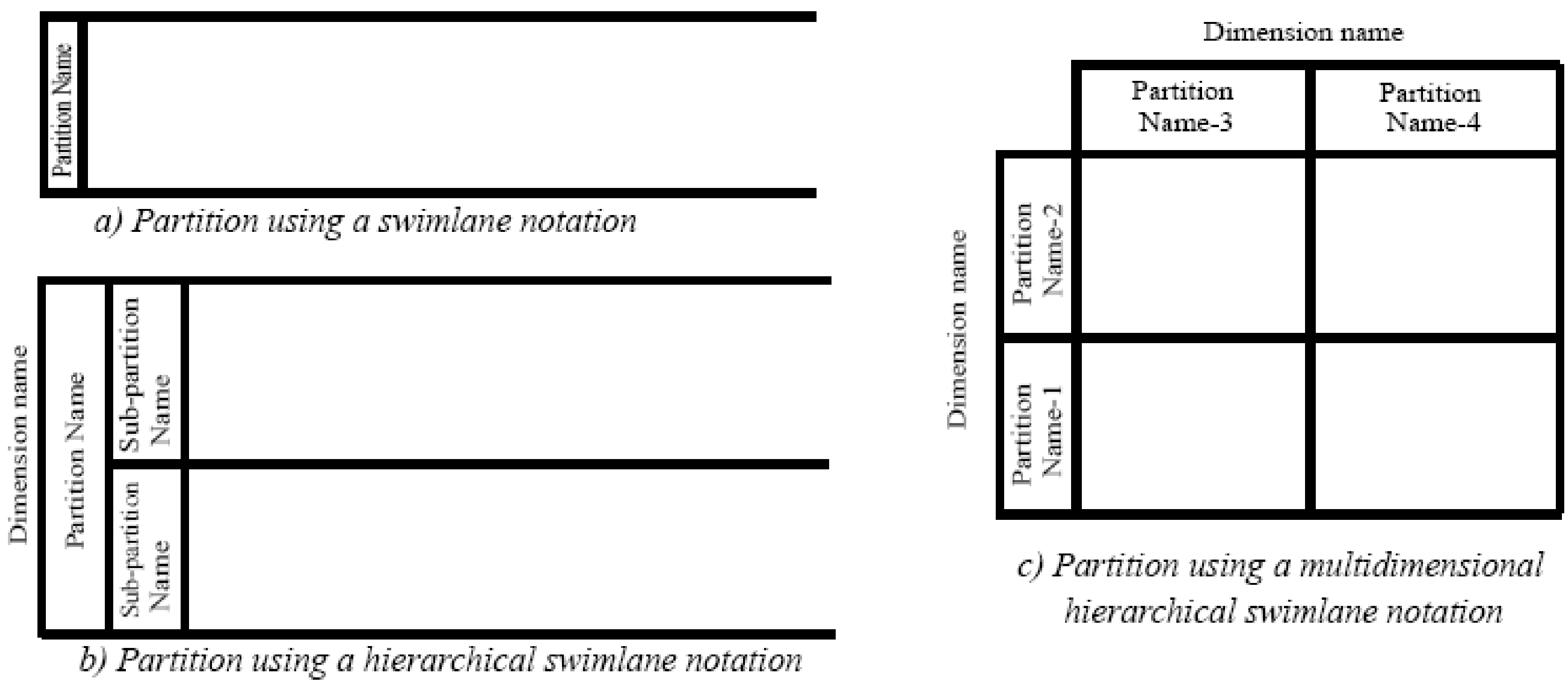
- Data (objects) passing along activity edges (can be specified as action pins – see right and bottom)
- Activities may have multiple input and output pins
- Possibility to characterize properties of a data flow link:
  - “stream” means that several tokens may be generated and waiting to be processed. Different selection behaviors (e.g., FIFO, LIFO)
  - Some transformation behavior may be specified
- Possibility to constrain the nature (e.g. state) of the object

Activity Diagram – Example

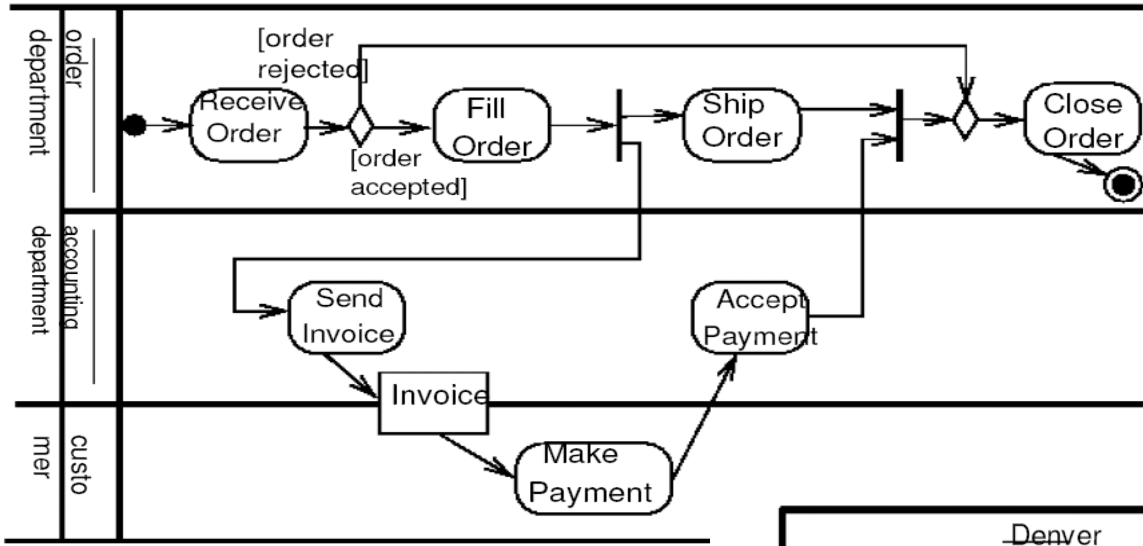


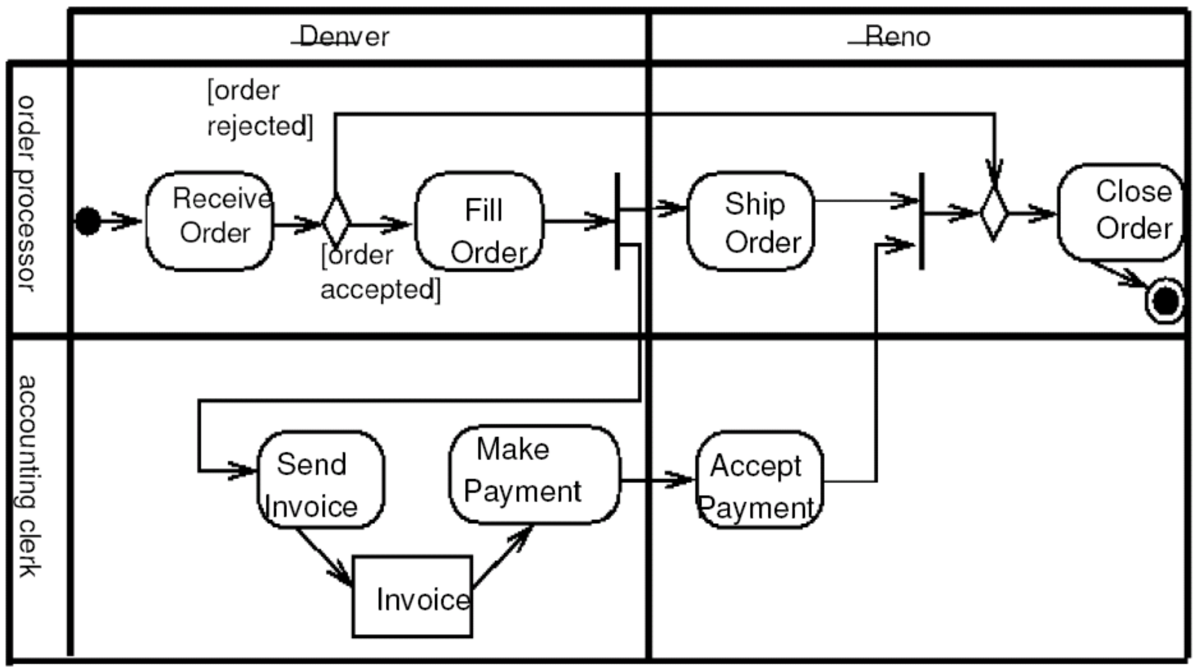
Partitions

- Partitions replace "swimlanes" in earlier UML versions
- Can have multiple dimensions and be hierarchical
- Getting closer to UCM components, but not quite there yet



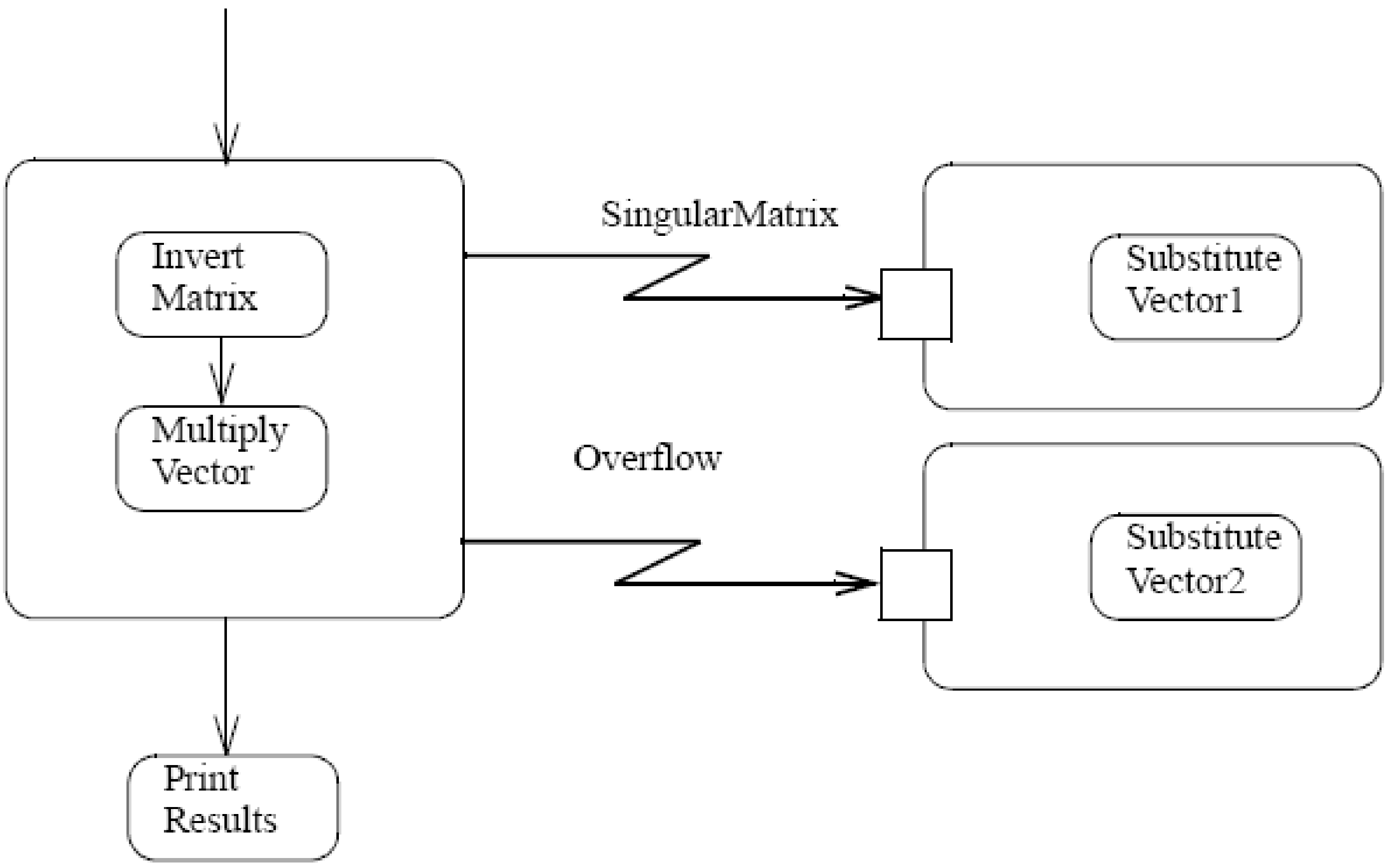
Partitions – Examples



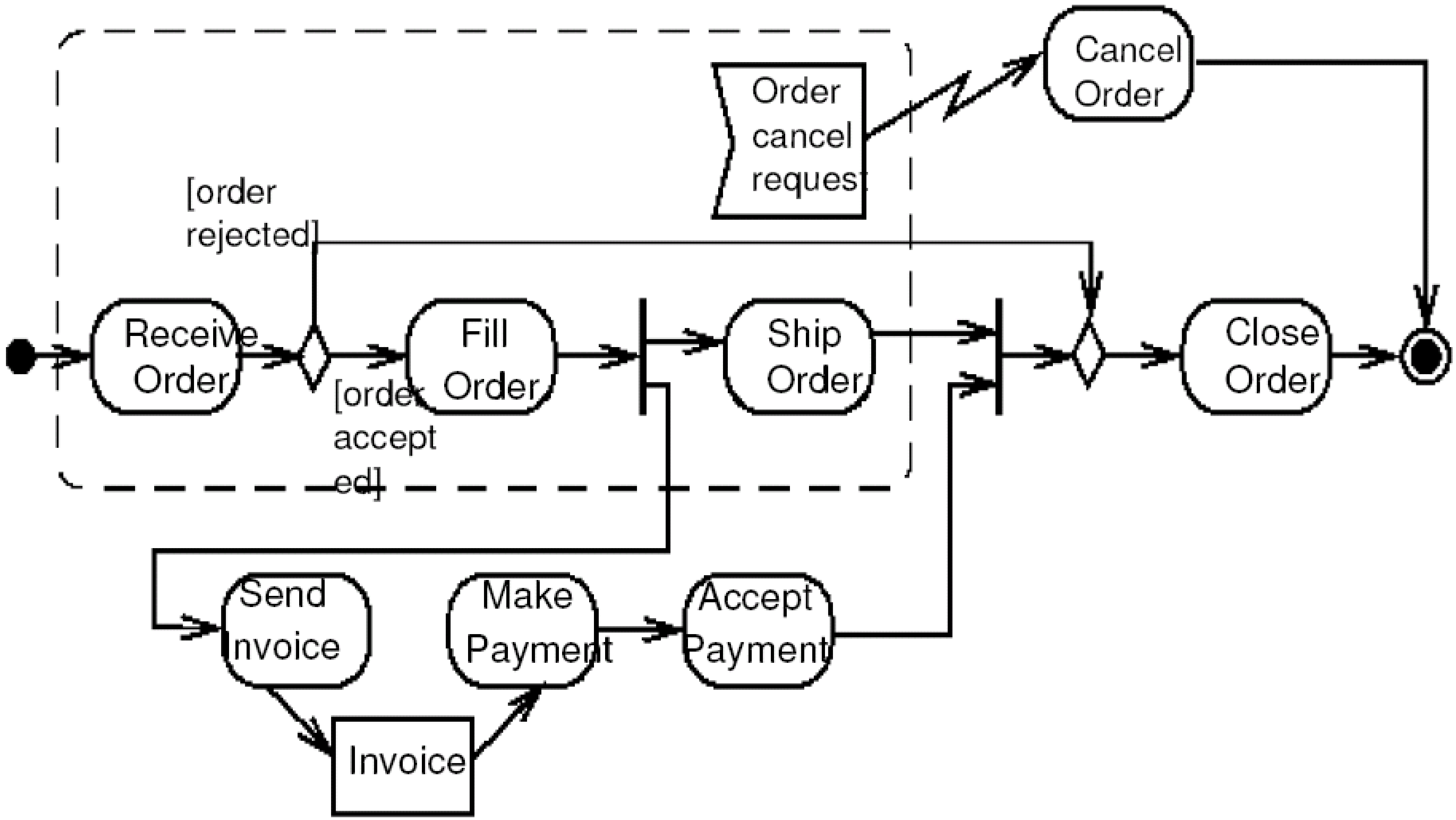


Exceptions

- An activity zone (left) can have exceptions (zigzag lines) handled by other activities (right)



Region Interruption



UML and URN

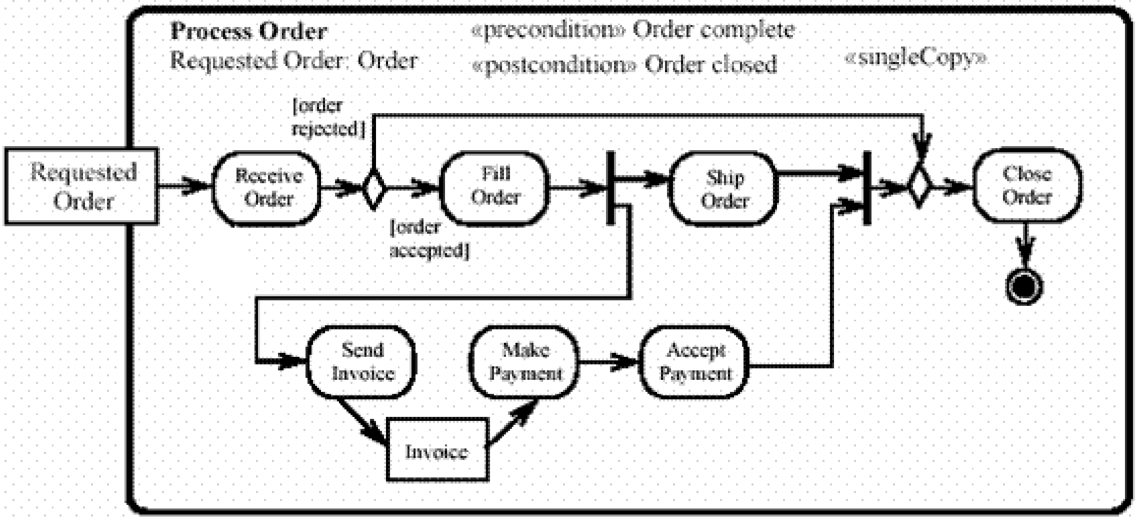
When to Use UML

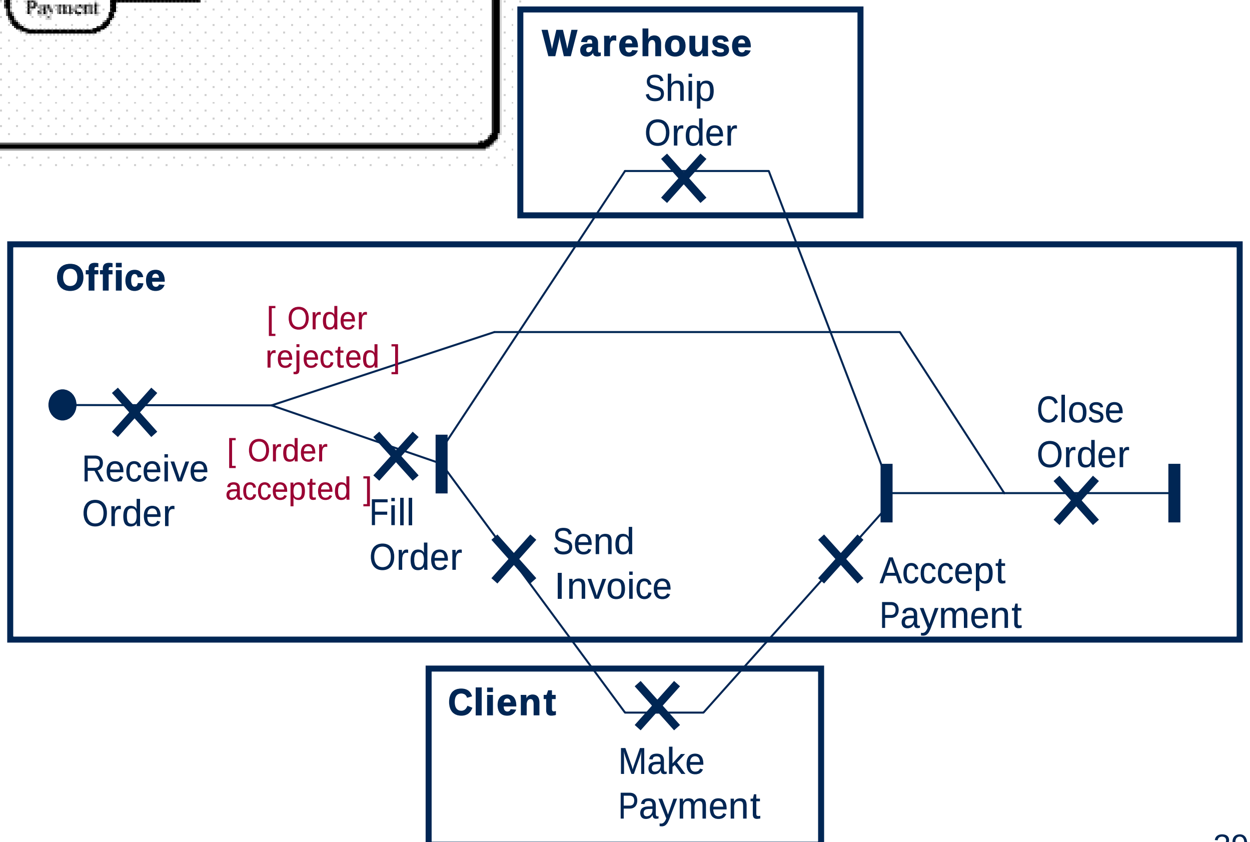
- Use Case Diagrams
  - Actors, system boundary, and structure of use cases
  - Applicable to system, subsystem, component...
- Class Diagrams
  - Domain modeling
- Activity Diagrams – here one can also use Use Case Maps
  - Process modeling (business or other)
  - Modeling of data and control flow
- Sequence Diagrams
  - Modeling interactions between actors and system or components
- State Machine Diagrams
  - Modeling detailed behavior (objects, protocols, ports)
  - Modeling the behavior of the system (black box)

UCM or UML Activity Diagrams?

- UCM and activity diagrams have many concepts in common
  - Responsibility ↔ action
  - Start/end points
  - Alternatives (fork / join)
  - Concurrency (fork / join)
  - Stub / plug-in ↔ action / sub-activity diagram
  - Association between elements and components / partition
  - Both may represent operational scenarios and business processes

Example comparison





Unique to UCM

- Dynamic stubs with several plug-ins
  - Activity diagrams have a single sub-activity diagram per action
- Plug-ins can continue in parallel with their parent model
  - Sub-activity diagrams must complete before returning to the parent activity diagram
- 2D graphical layout of components
- Definitions of scenarios (integrated testing capabilities!)
- Integration with GRL in URN

Unique to Activity Diagrams

- Data flow modeling
- Interruptible regions
- Conditions on parallelism (branches of an AND-fork)
- Constraints on action pins
- Integration with UML

Model-Based Analysis
(for Workflow models)

Sequence Diagram

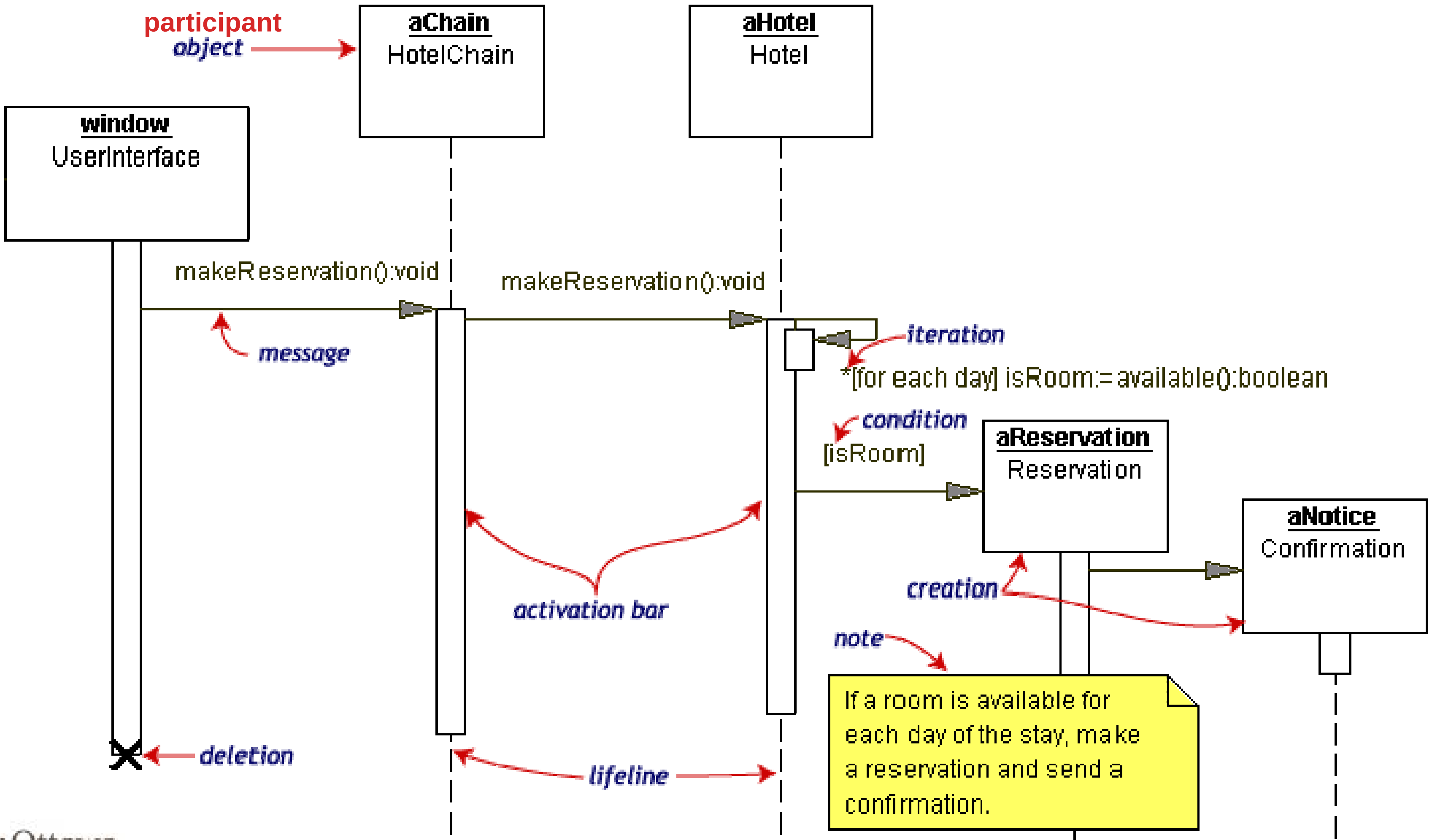
UML 2.x Sequence Diagrams

- Major improvements in UML version 2, based on ITU-T's Message Sequence Charts (MSC)

- The most important one: combined fragments
- Other improvements
  - (A)synchronous interactions
  - References
  - Hierarchical decomposition
  - Temporal aspects
  - …

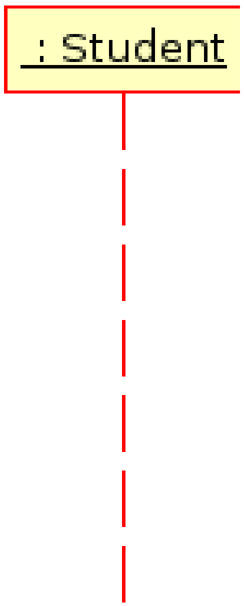
Basic Notational Elements of Sequence Diagrams

- Describe the dynamic behavior as interactions between so-called "participants" (e.g. agents, actors, the system, system components). For each participant, there is a "lifeline"

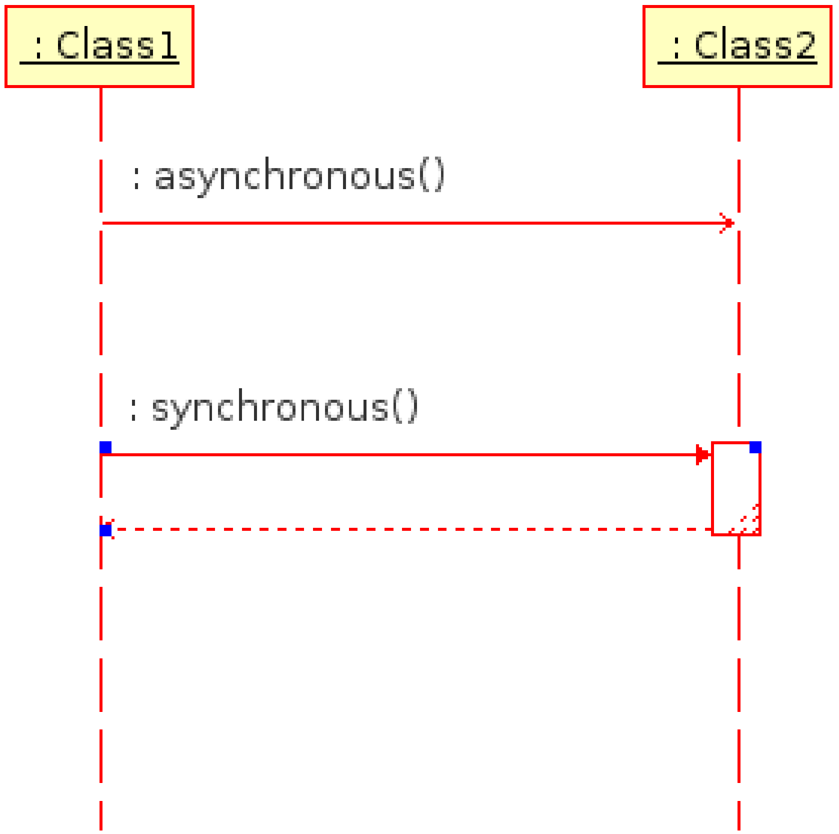


Lifelines and (A)synchronous Interactions

- Participants, shown using lifelines, participate in the interaction sequence by sending / receiving messages



- Messages can be synchronous or asynchronous

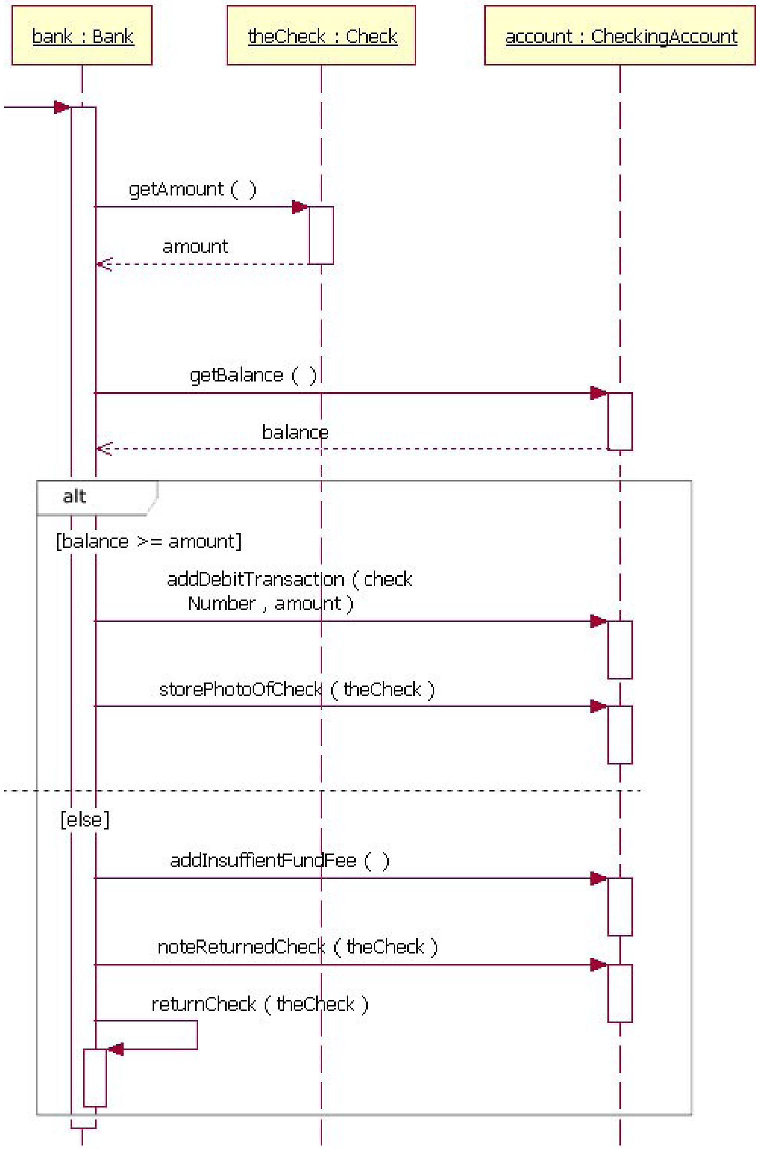


Combined Fragments

- Allow multiple sequences to be represented in compact form (may involve all participants or just a subset)
- Combined fragment operators
  - alt, for alternatives with conditions
  - opt, for optional behavior
  - loop(lower bound, upper bound), for loops
  - par, for concurrent behavior
  - critical, for critical sections
  - break, to show a scenario will not be covered
  - assert, required condition
  - ignore/consider(list of messages), for filtering messages
  - neg, for invalid or mis-use scenarios that must not occur
  - strict or seq, for strict/weak sequencing (WHAT IS THIS ?)
  - ref, for referencing other sequence diagrams

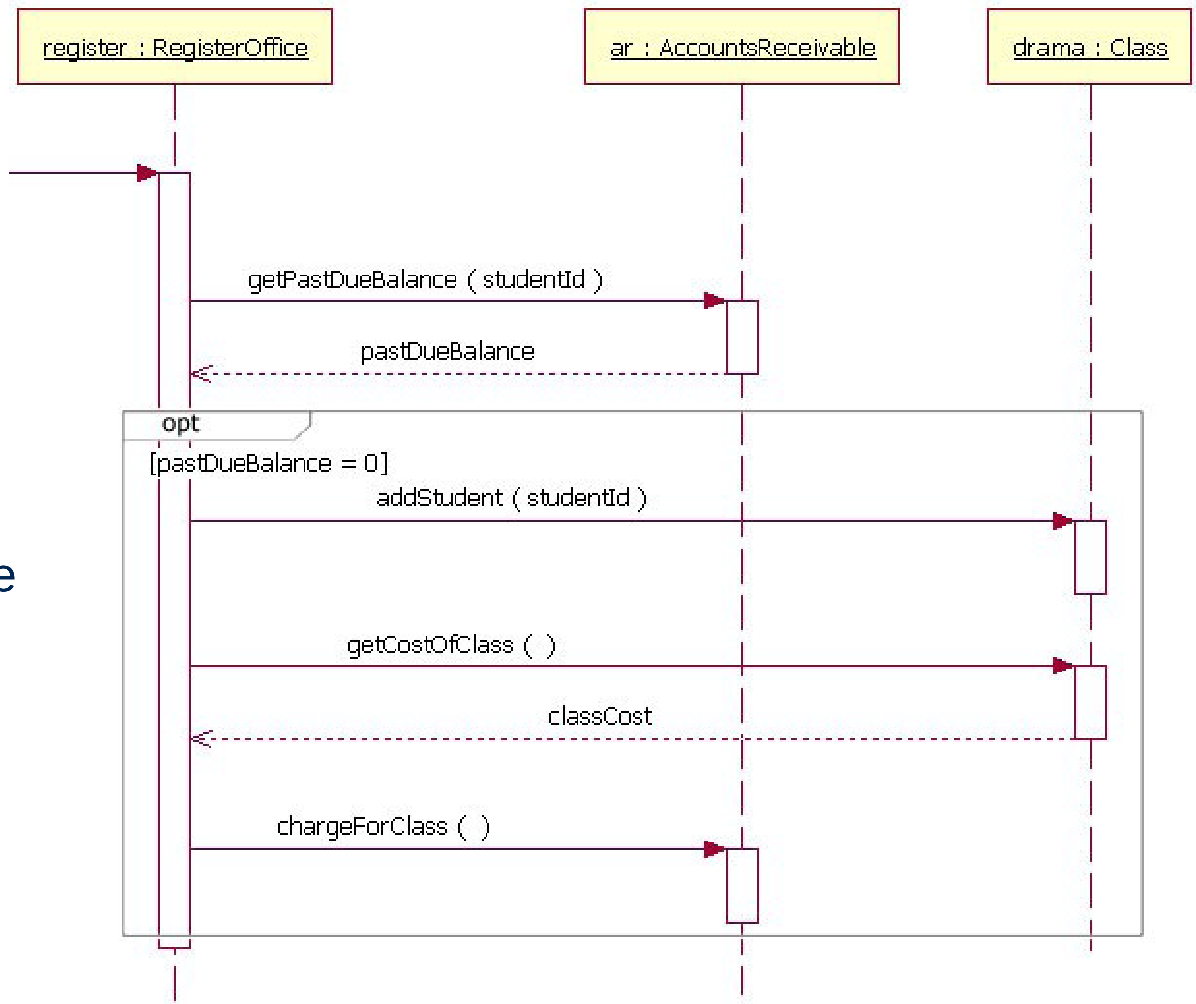
Combined Fragments – Alternative

- Alternative (operator alt)
  - Multiple operands (separated by dashed lines)
  - Each operand has guard condition (no condition implies true)
  - One will be chosen exclusively – nondeterministically if more than one evaluates to true
  - Special guard: else
    - True if no other guard condition is true



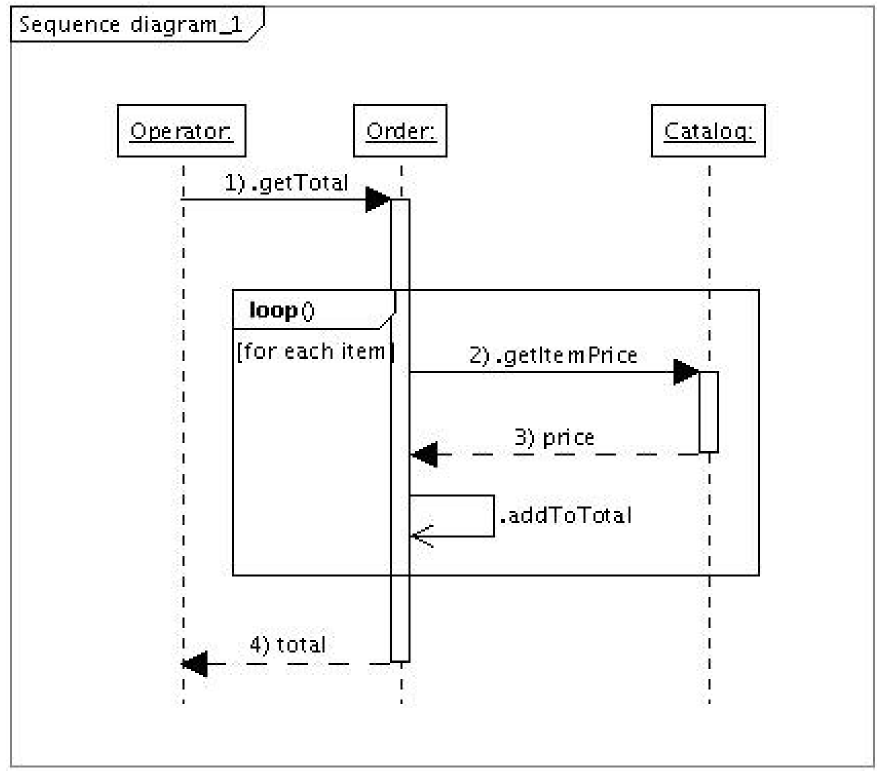
Combined Fragments – Optional

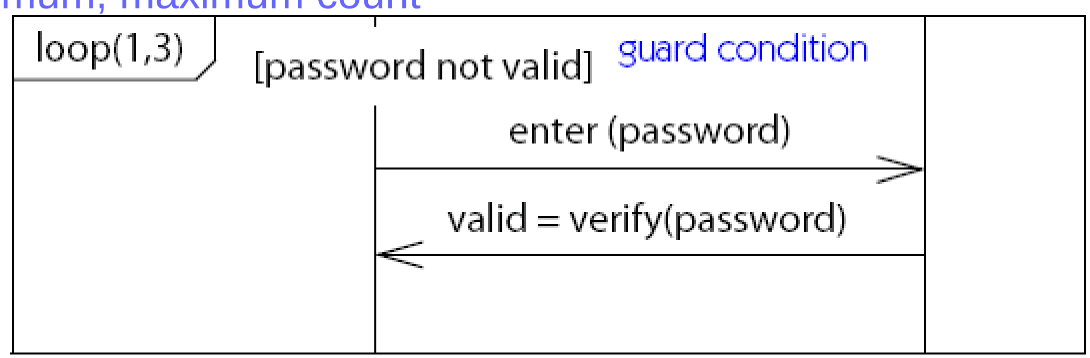
- Optional (operator opt)
  - To specify a guarded behavior fragment with no alternative
  - Special case of alt
  - Equivalent to an alt with two operands
    - The first is the same as the operand for the opt
    - The second is an empty operand with an else guard



Combined Fragments – Loop

- Loop (operator loop)
  - Loop fragment may execute multiple times
  - At least executed the minimum count
  - Up to a maximum count as long as the guard condition is true (no condition implies true)

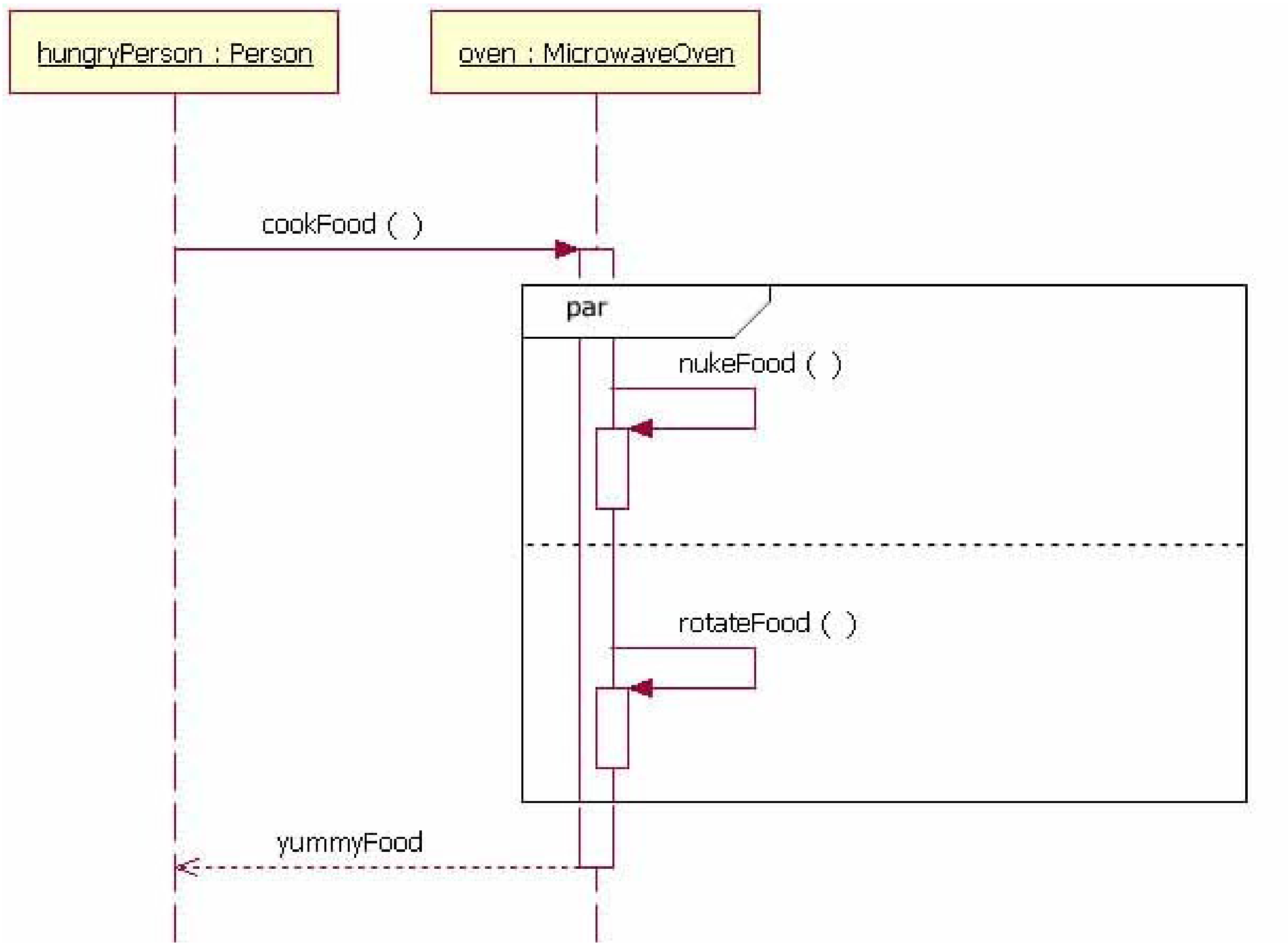




Source for Password Example: UML Reference Manual

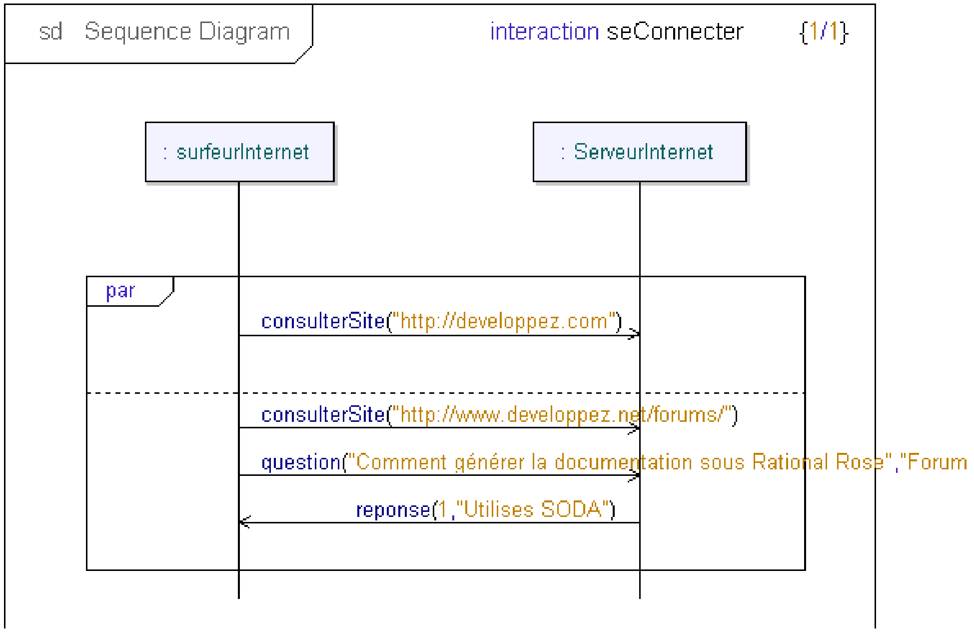
Combined Fragments – Concurrency

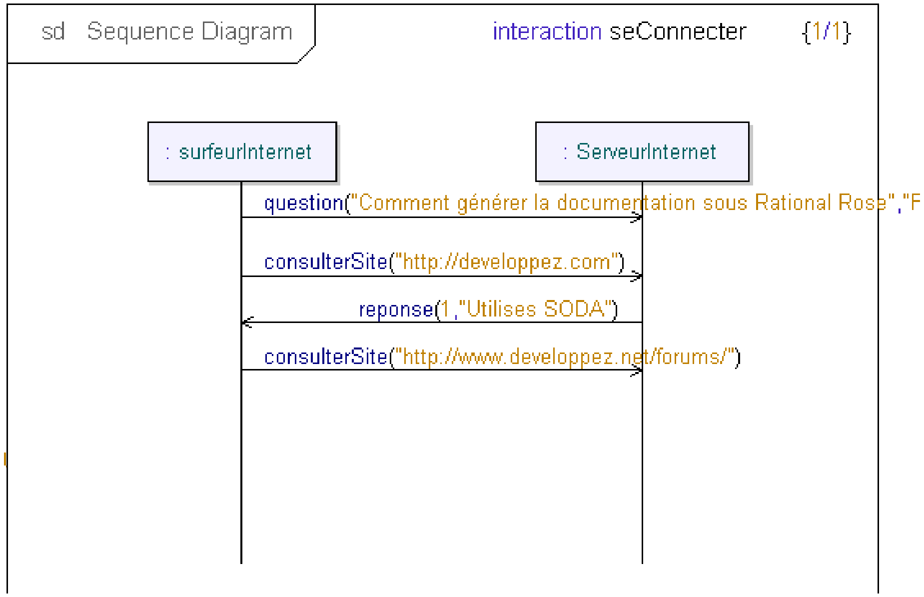
- Concurrency (operator par)
  - Two or more operands that execute in parallel



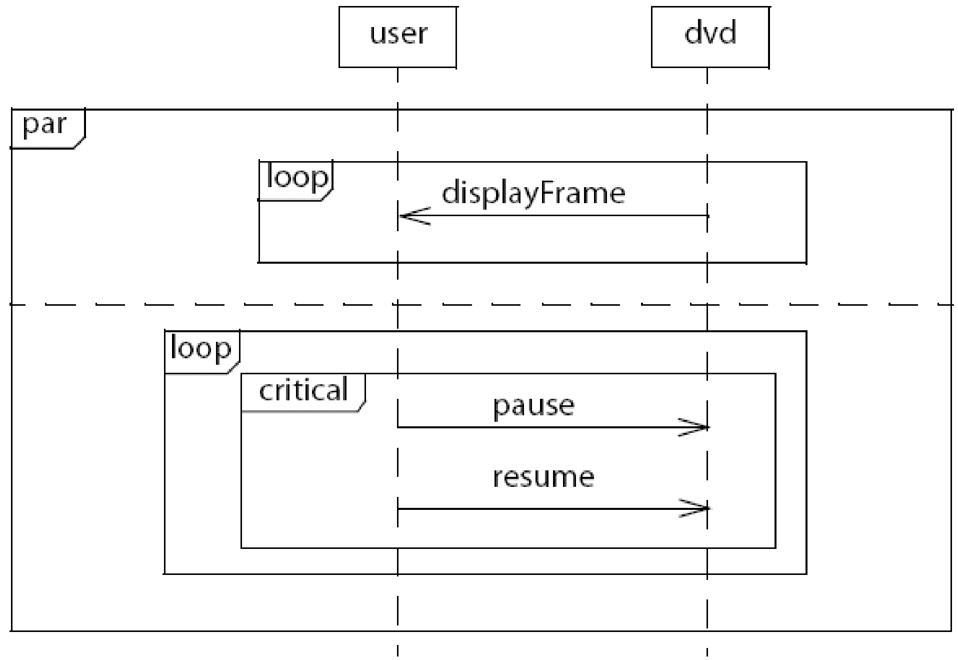
Concurrency Quiz – Part One!

- Is the interaction on the right a valid sequential trace that can be generated from the interaction with the par combined fragment on the left?
- No! The sequences of the two operands may be interleaved but the ordering defined for each operand must be maintained.





Concurrency Quiz – Part Two!



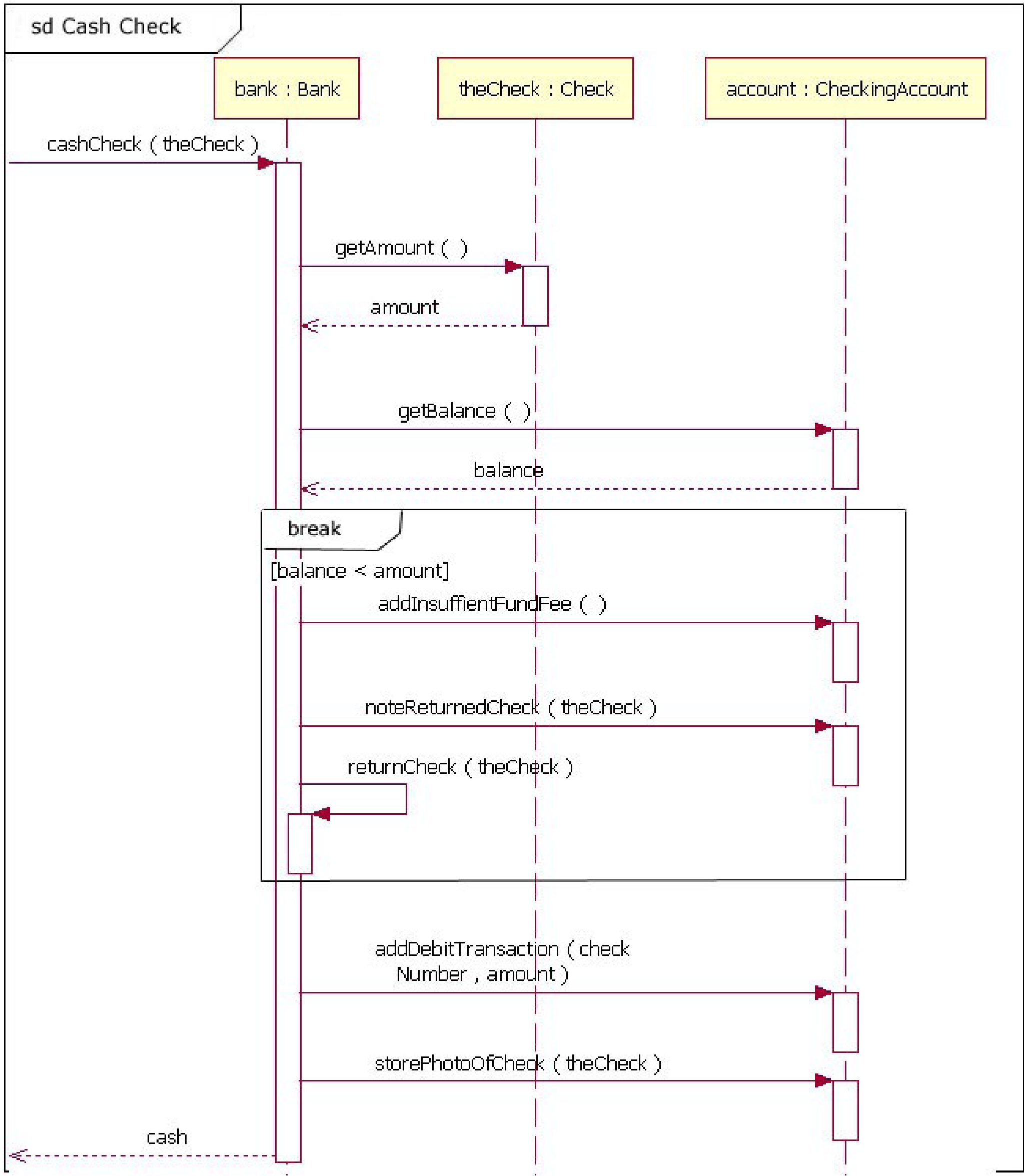
- What are valid sequential traces for this interaction with the critical operator?

- In the main loop, the player repeatedly displays frames. At any time (because it is within a par combined fragment), the user can send a pause message to the player. Afterwards the user sends a resume message. Because these two messages are in a critical region, no displayFrame message may be interleaved. Therefore, the player stops displaying frames until the resume message occurs!

Source: UML Reference Manual

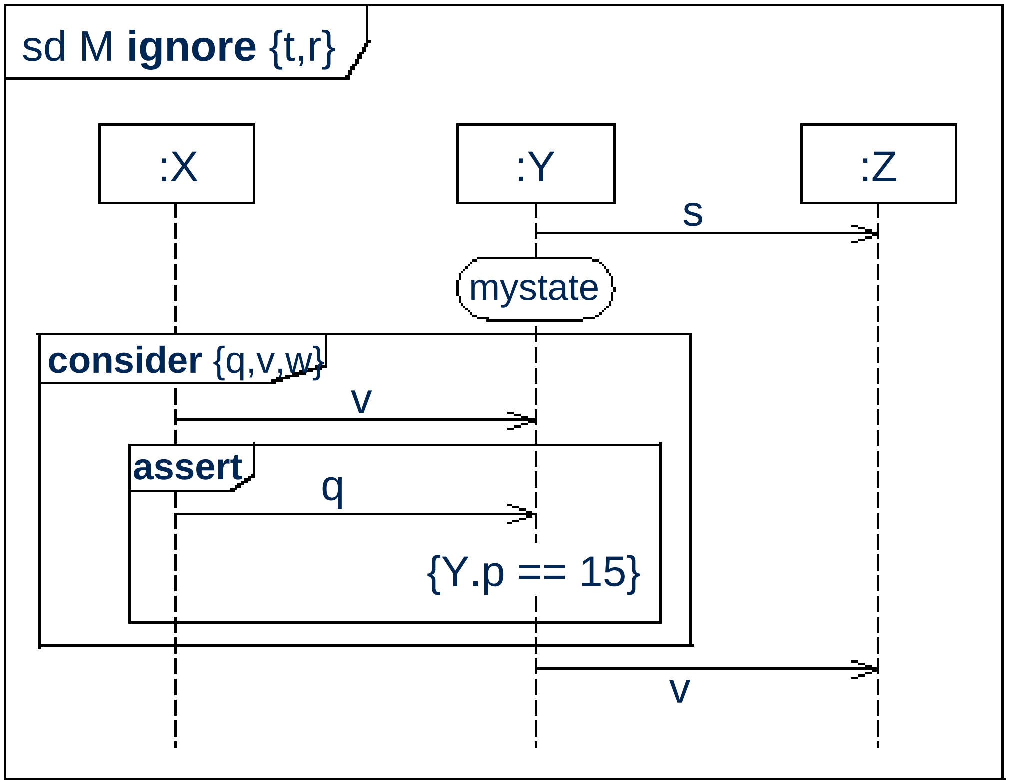
Combined Fragments – Break

- Concurrency (operator break)
  - Execute the break combined fragment if the guard condition is true and then jump to the end of the interaction
  - If the guard condition of the break combined fragment is not true, do not execute the break combined fragment and continue with the interaction below the break combined fragment



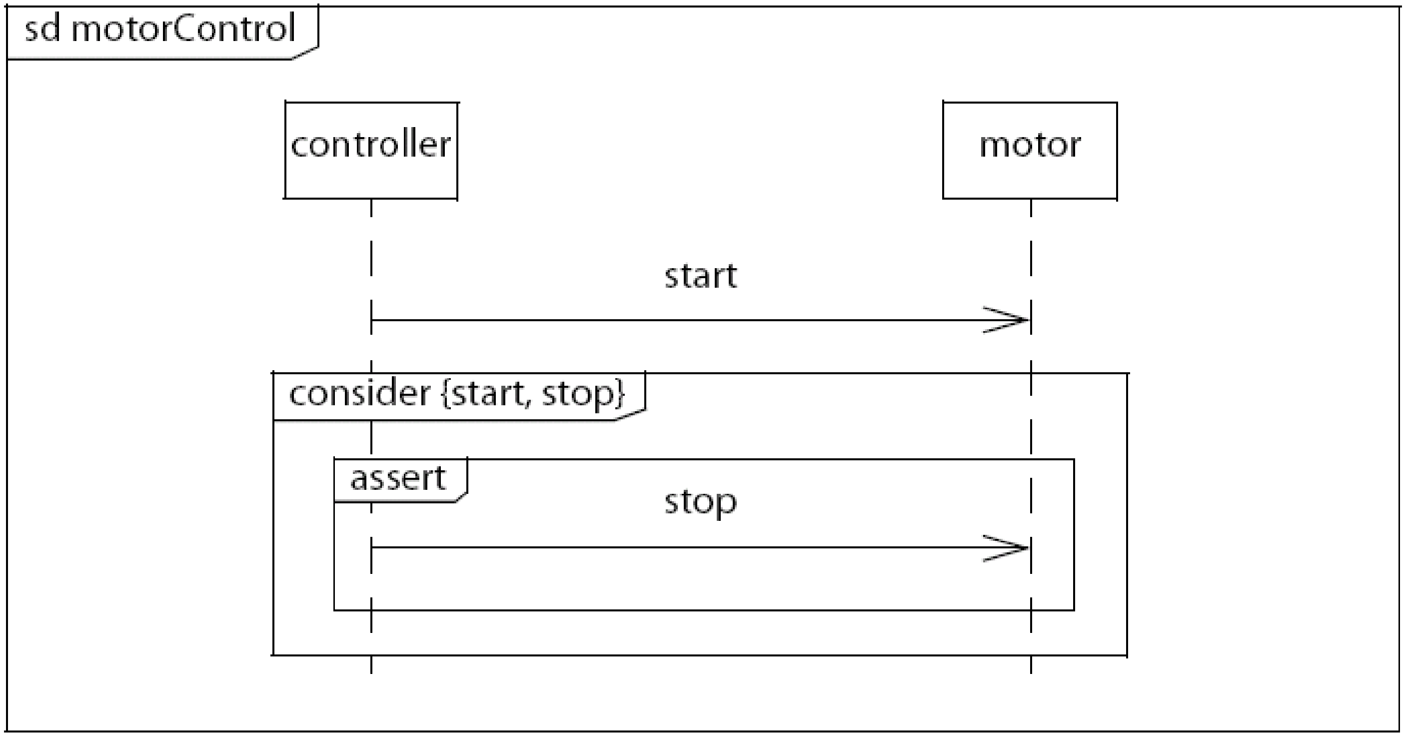
Combined Fragments – Assertions and State Invariants

- Assertions (operator assert)
  - Behavior of assert combined fragment must occur
  - Often combined with consider and ignore
  - Consider: other messages may occur but we do not care about them
  - Ignore: listed messages may occur but we do not care about them
- State invariant
  - Evaluated when the next event occurs on lifeline
  - Small rectangle with rounded corners or curly brackets
- Useful for testing



Assertion Quiz!

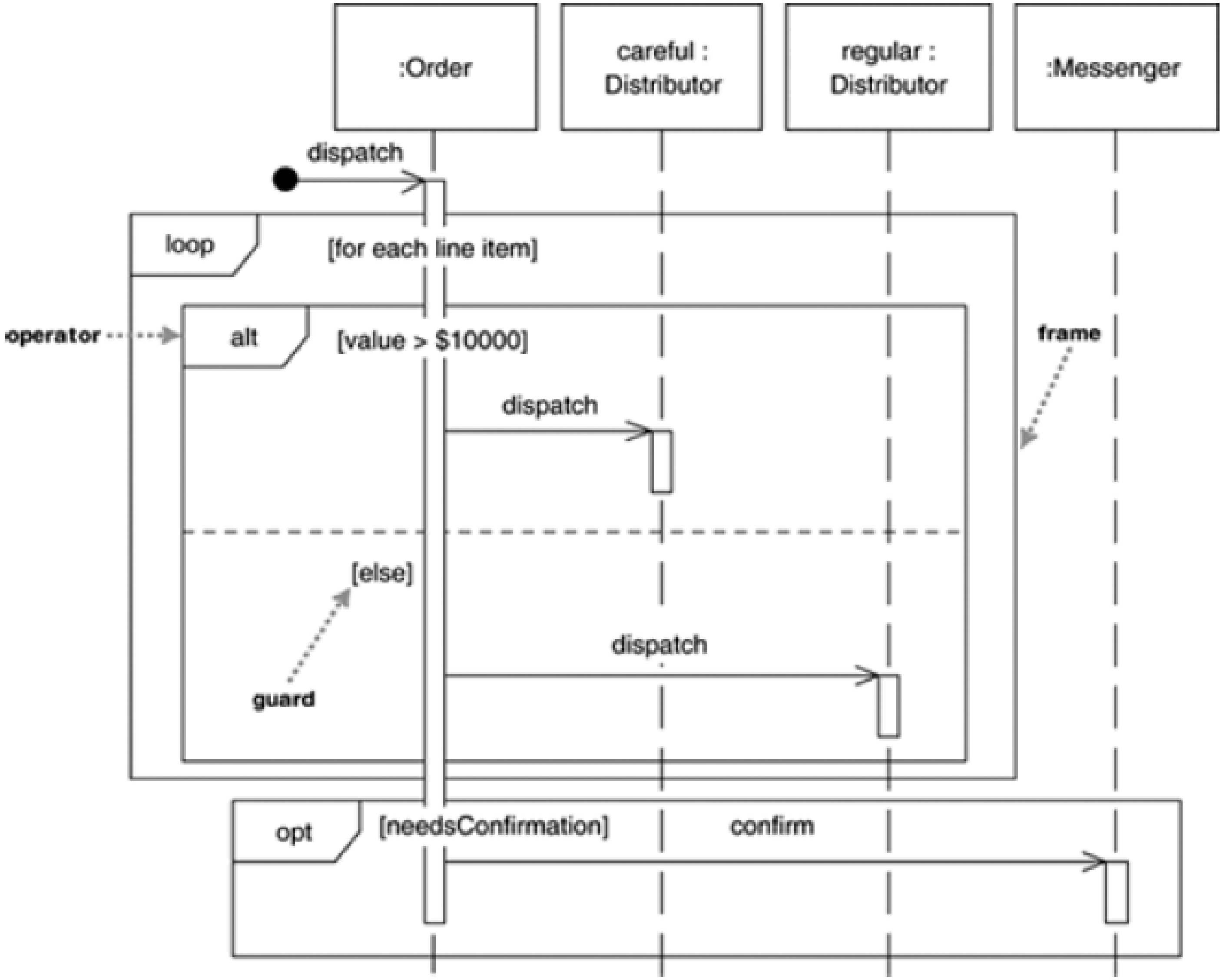
- What are valid sequential traces for this interaction with the assert and consider operators?



- Start; any other messages except start may occur; stop must occur

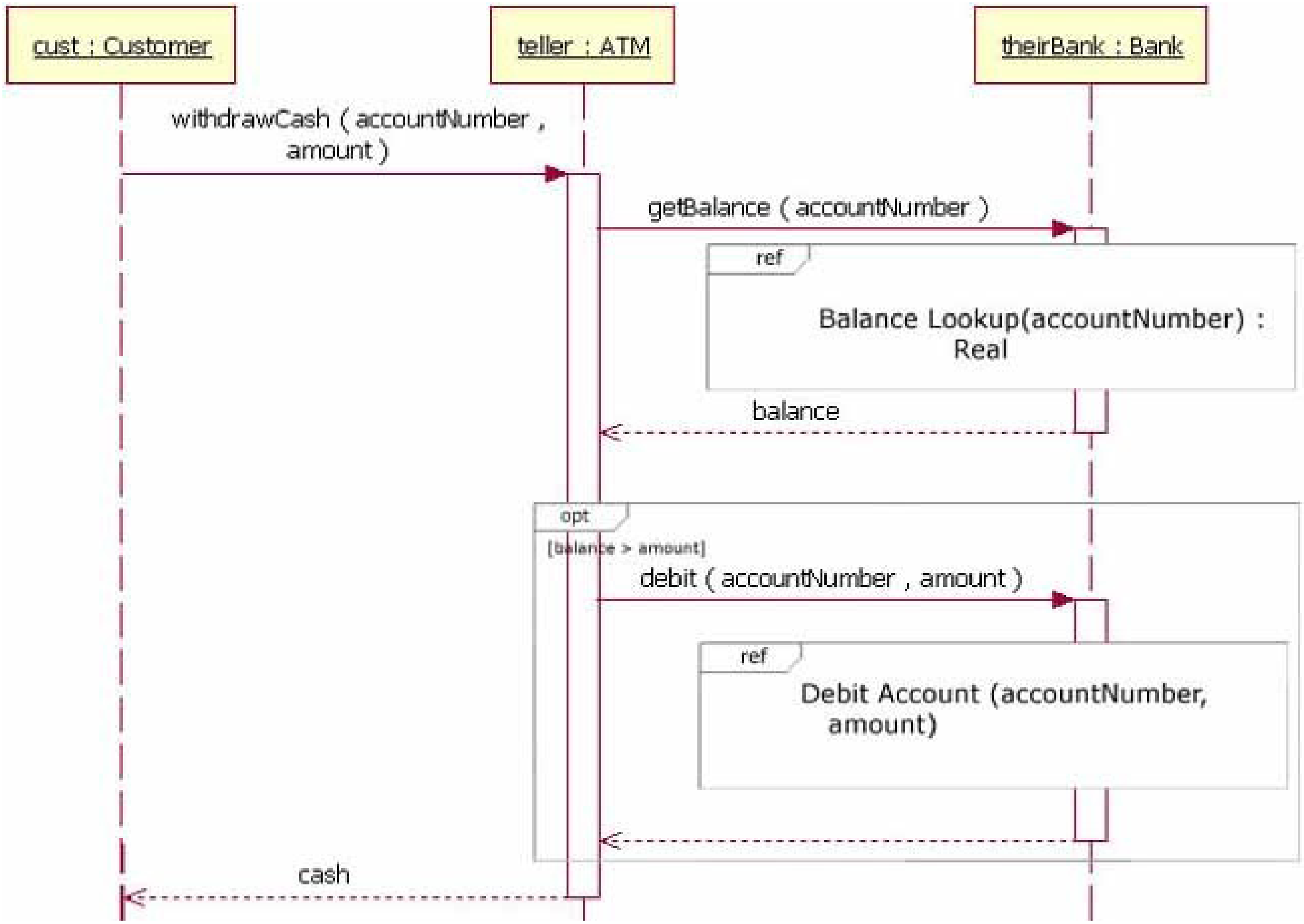
Source: UML Reference Manual

Nested Combined Fragments



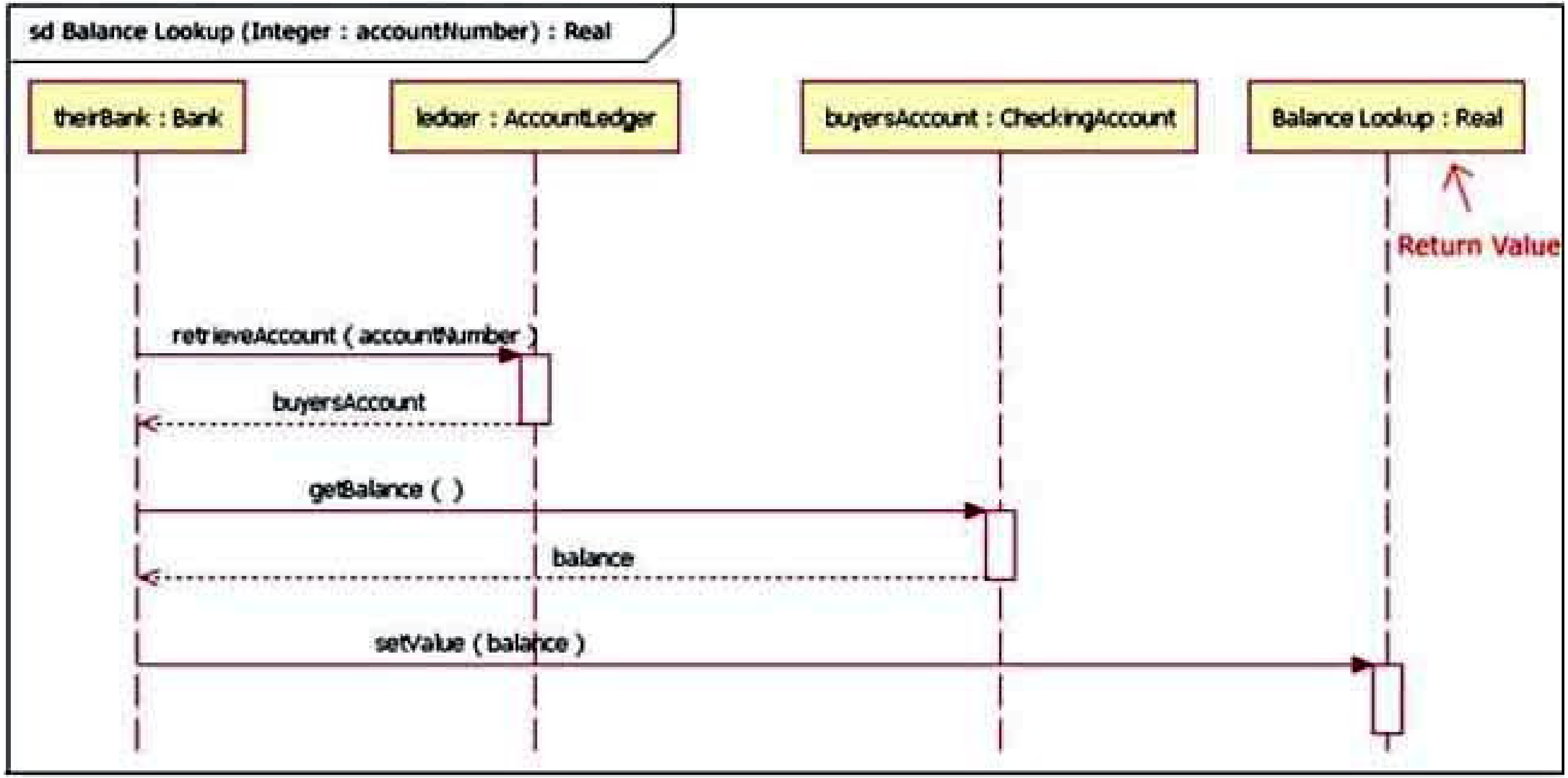
Combined Fragments – References (1)

- References (operator ref)
  - Called interaction use
  - Includes another sequence diagram
  - Parameters may be passed and a result returned



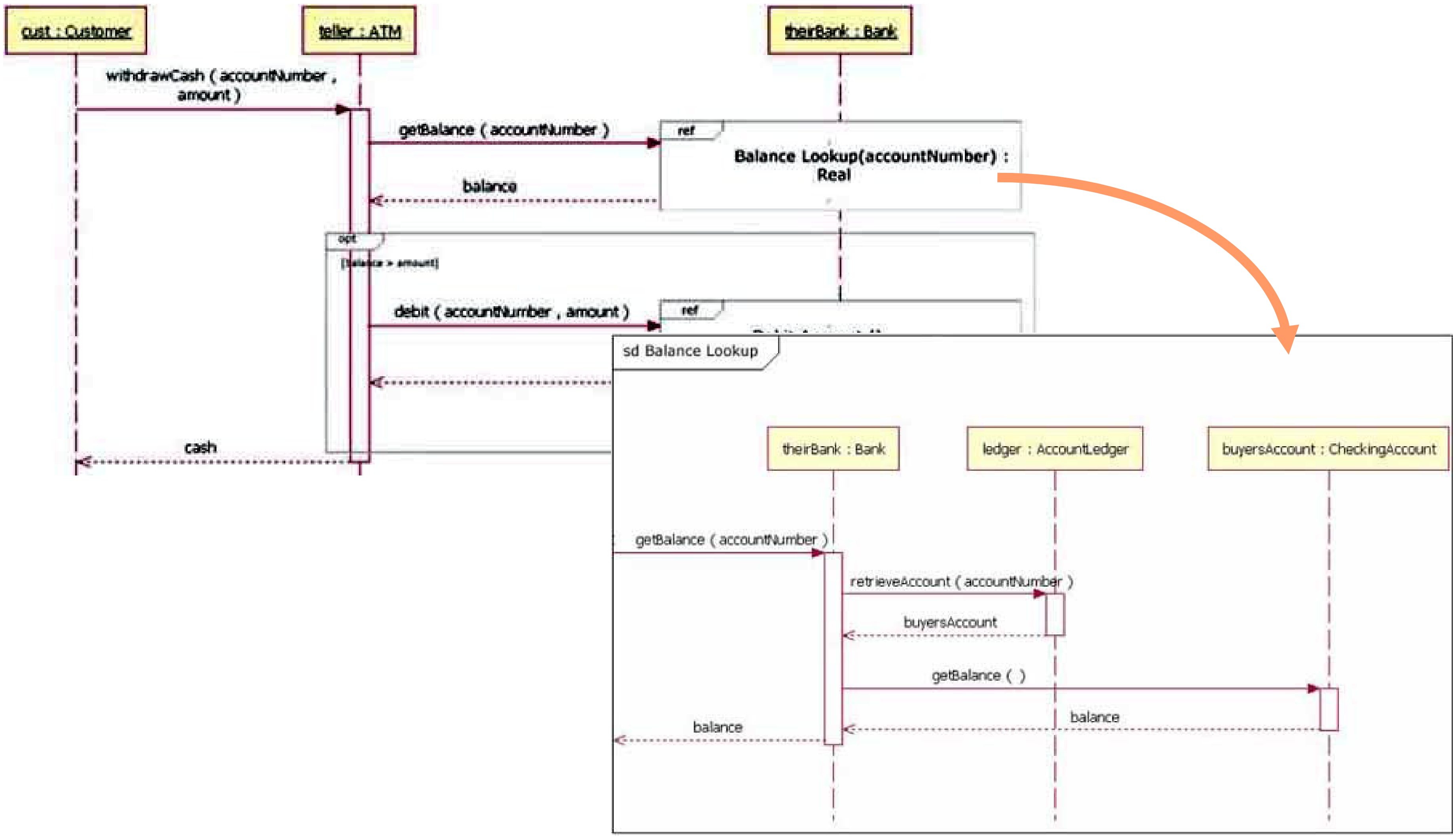
Combined Fragments – References (2)

- Referenced sequence

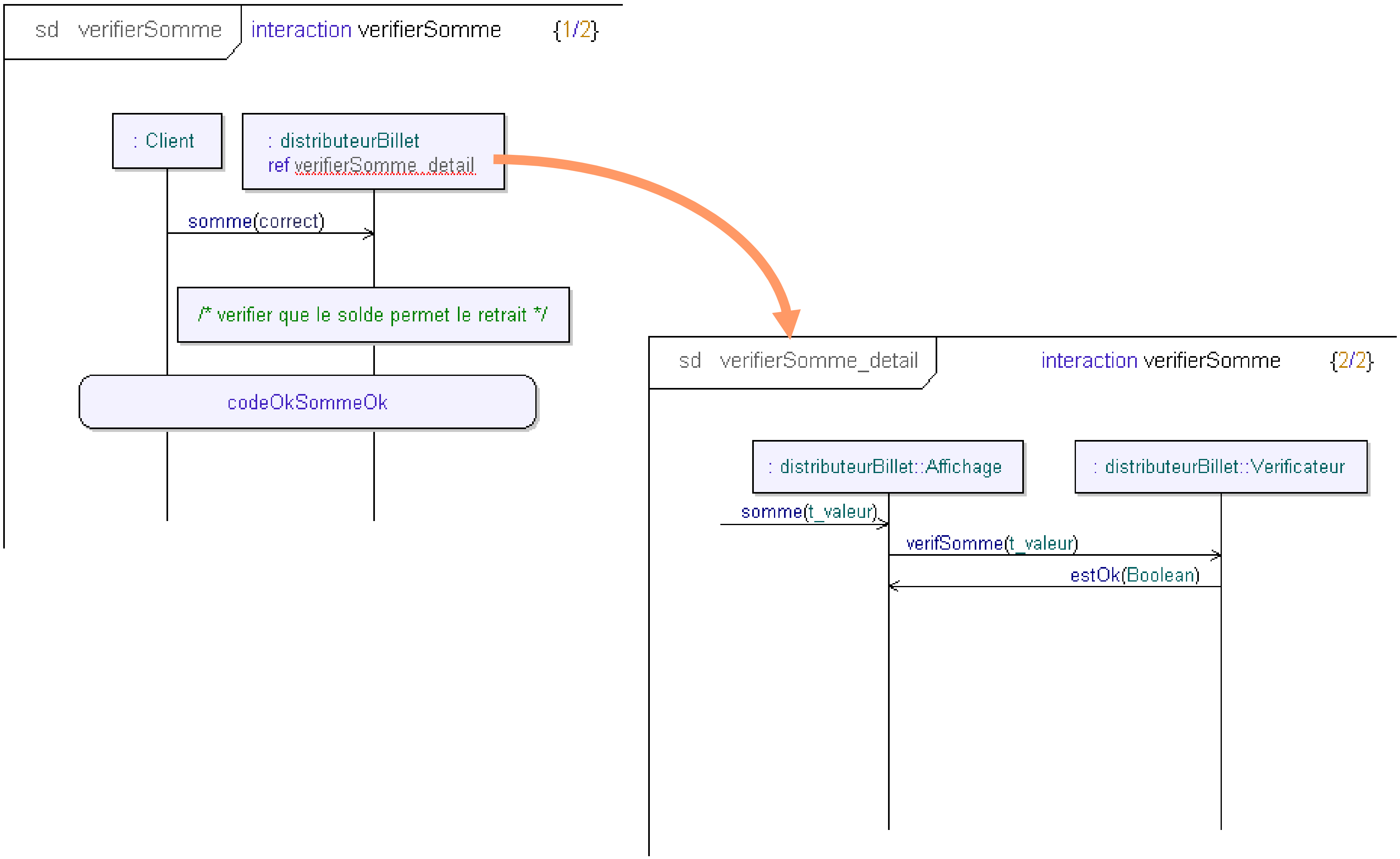


Combined Fragments – References (3)

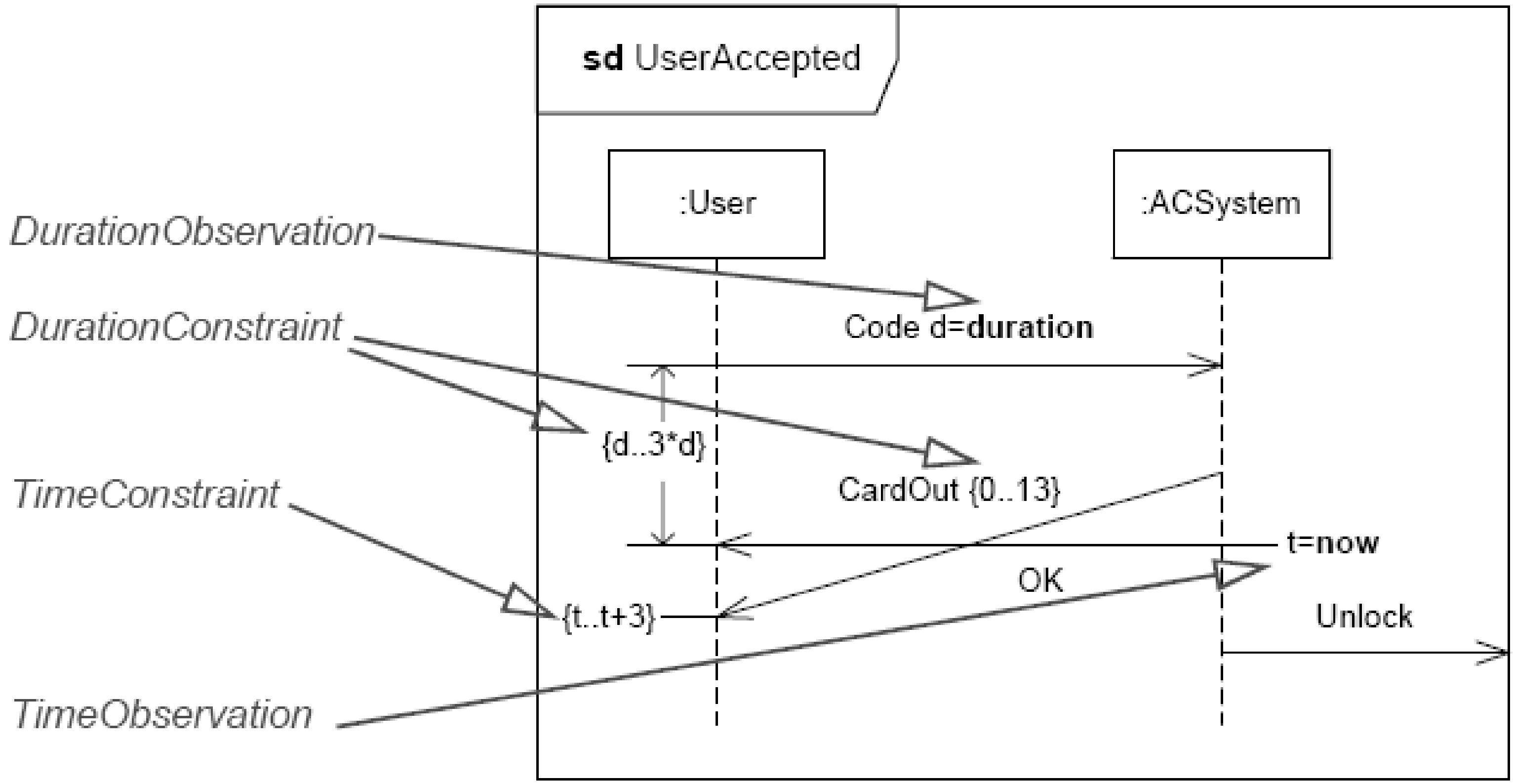
- Reference can connect to a gate (border of diagram frame)



Hierarchical Decomposition of Participants

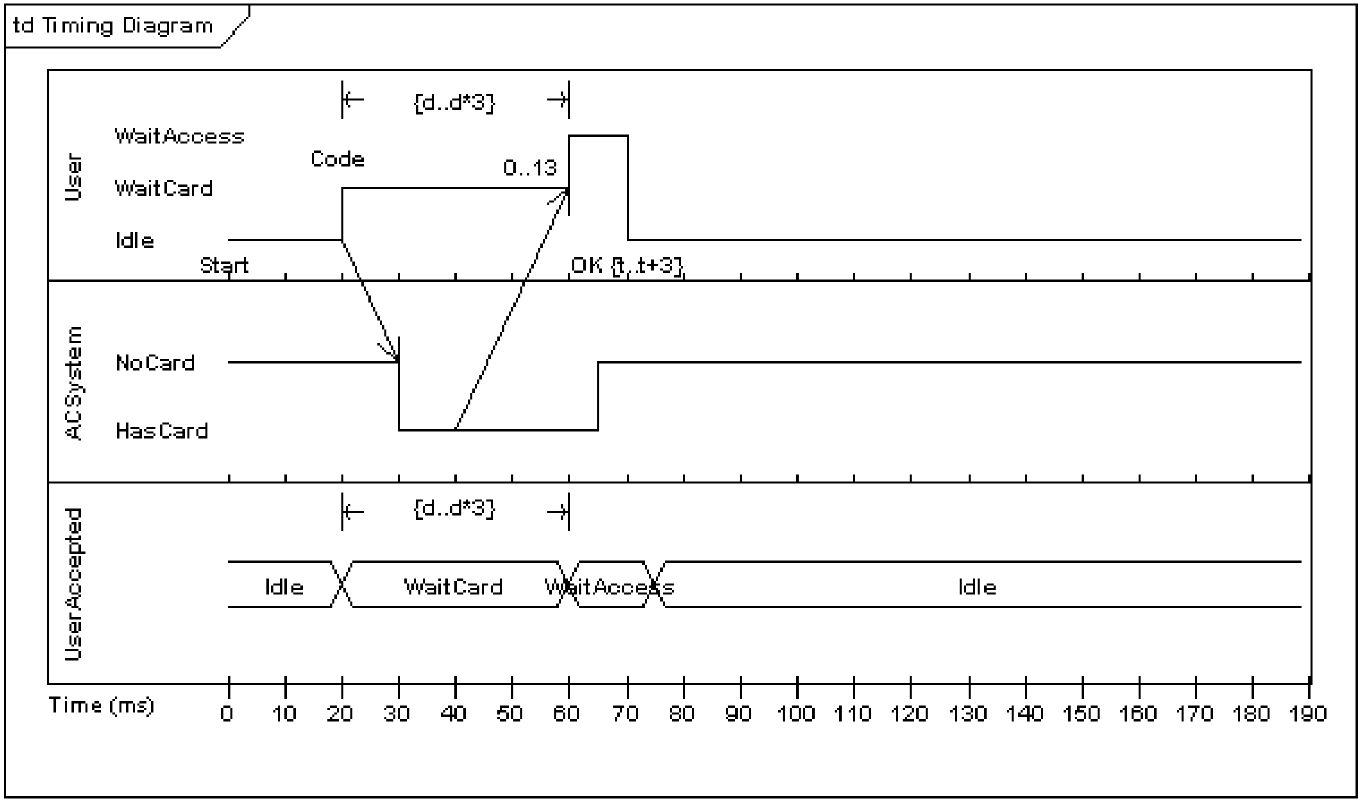


Temporal Aspects



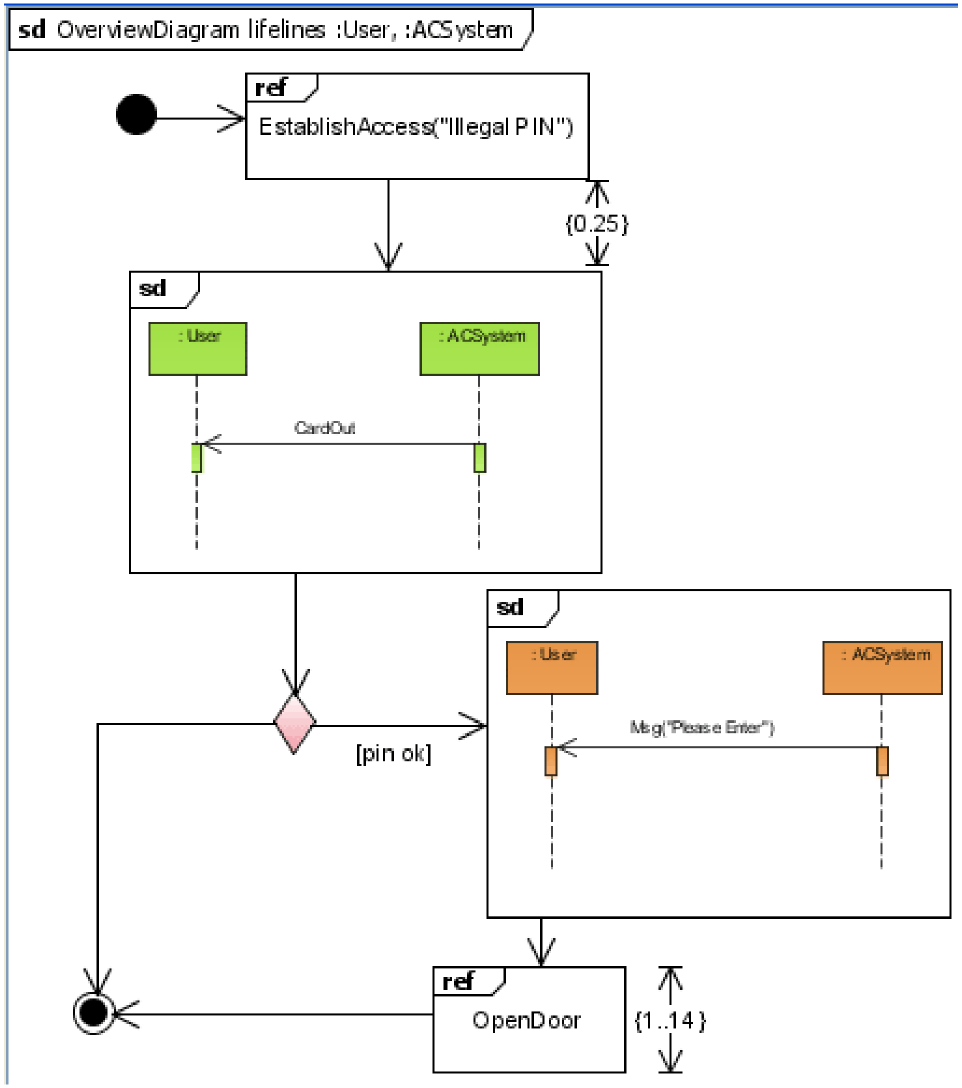
UML 2.x Timing Diagrams

- Behavioural view similar to sequence diagrams but presented with a graphical syntax inspired from signals in logic circuits
- Can be used to specify time-dependent interactions
  - Primary purpose of the diagram is to reason about time
  - Focus on conditions changing within and among lifelines along a linear time axis



UML 2.x Interaction Overview Diagrams

- Similar to an activity diagram that references or includes sequence diagrams
- Give an overview of the flow of control
- Nodes are interaction diagrams



Source: http://www.visual-paradigm.com/VPGallery/diagrams/InteractionOverviewDiagram.html

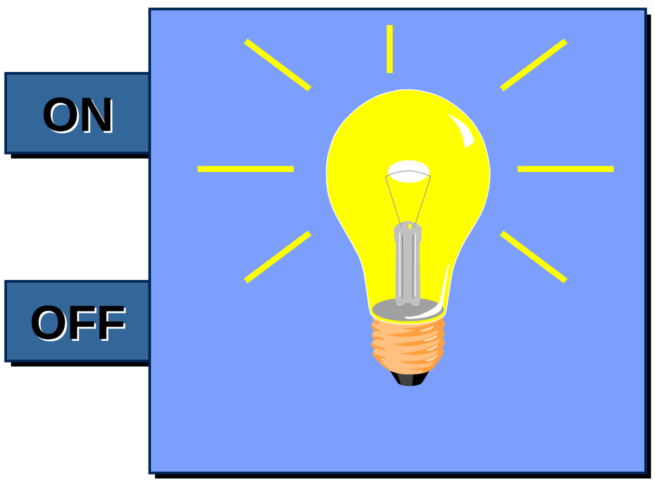
State Machine Diagram

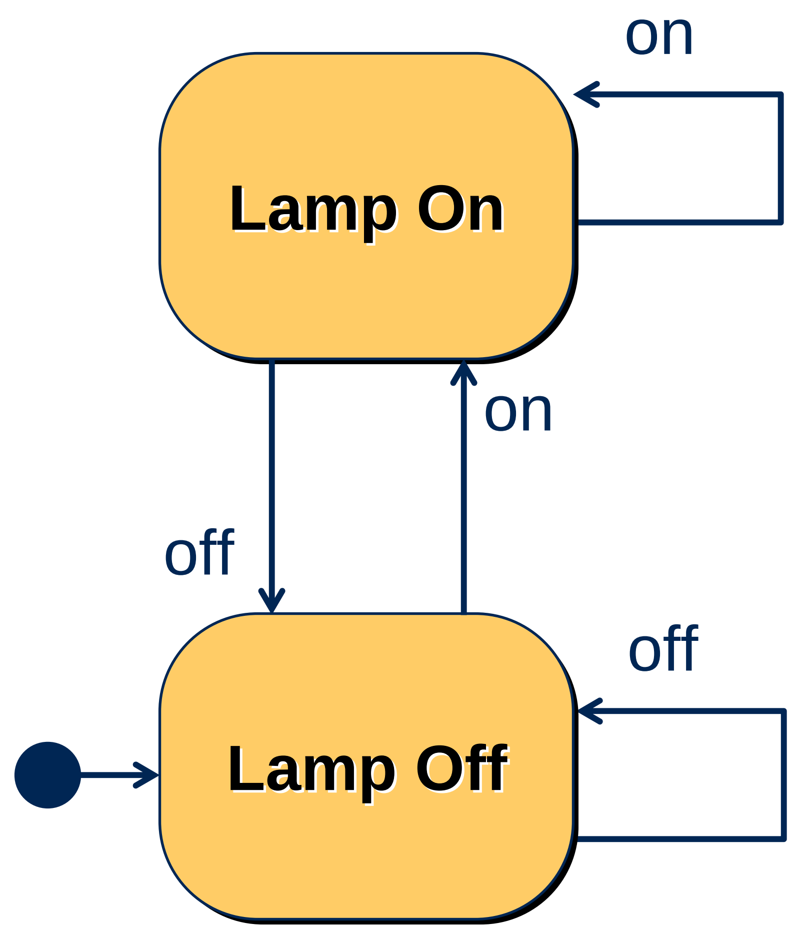
UML 2.x State Machine Diagrams

- Model discrete behavior (finite state-transition systems)
  - System
  - Component
  - Class
  - Protocol
- Several formal definitions as well as textual and graphical syntax of state machines exist
  - We focus on the state machines of UML 2.x
- Several techniques and tools exist for defining, analyzing, combining, and transforming (e.g., to code) state machines

Automaton

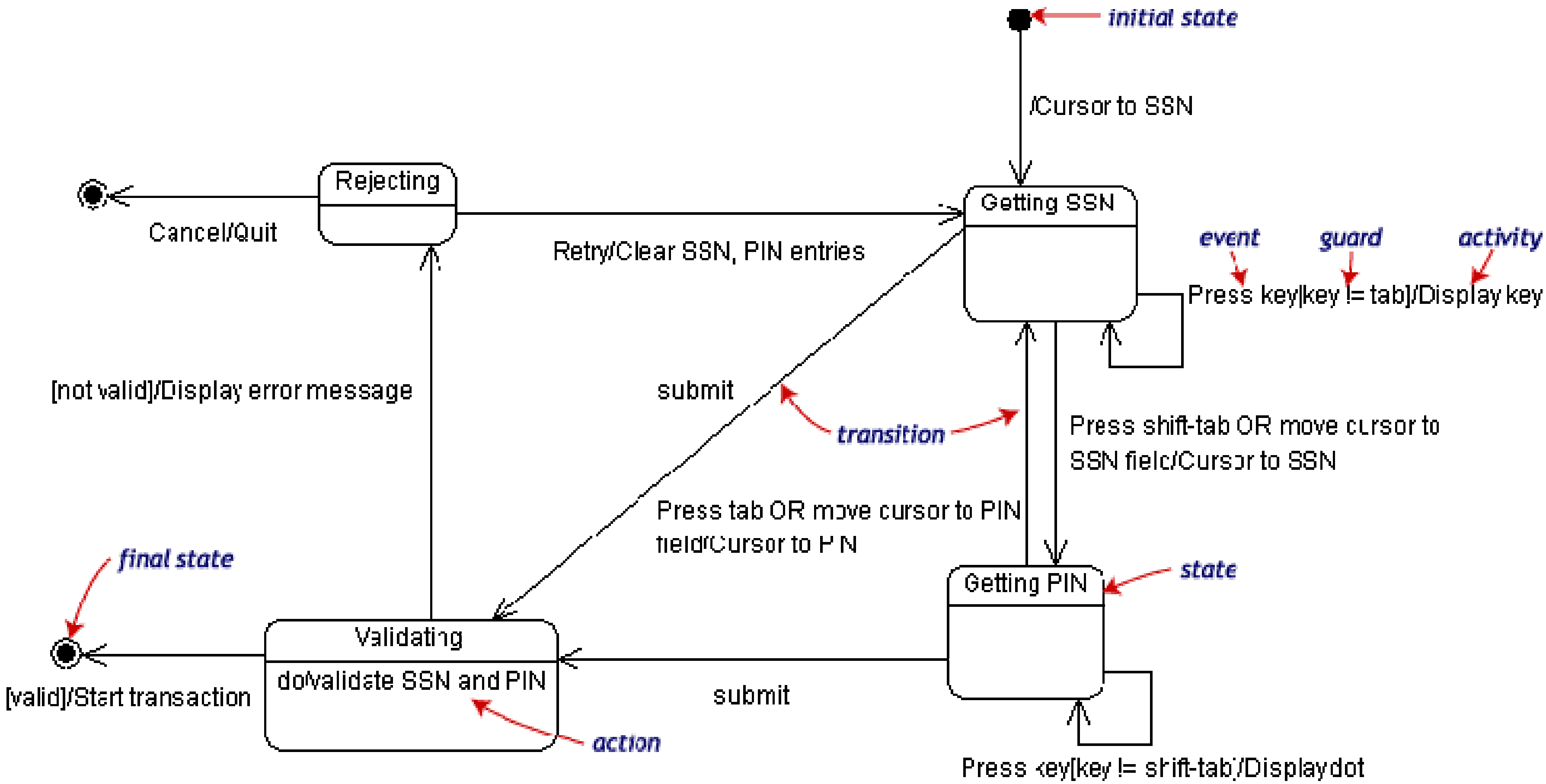
- A machine whose output depends not only on the input but also on the history of past events
- Its internal state characterizes this history



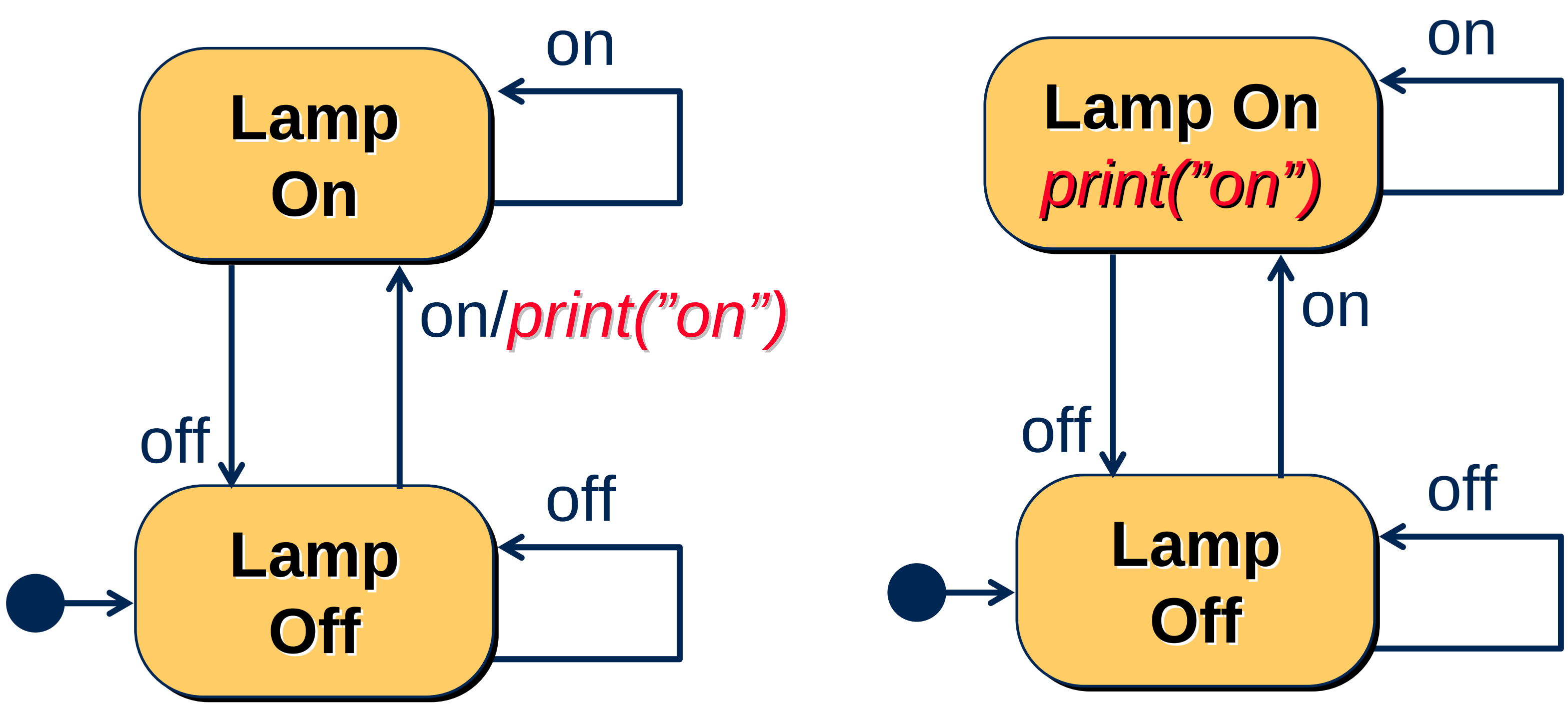


Basic Notational Elements of State Machine Diagrams

- Describe the dynamic behavior of an individual object (with states and transitions)



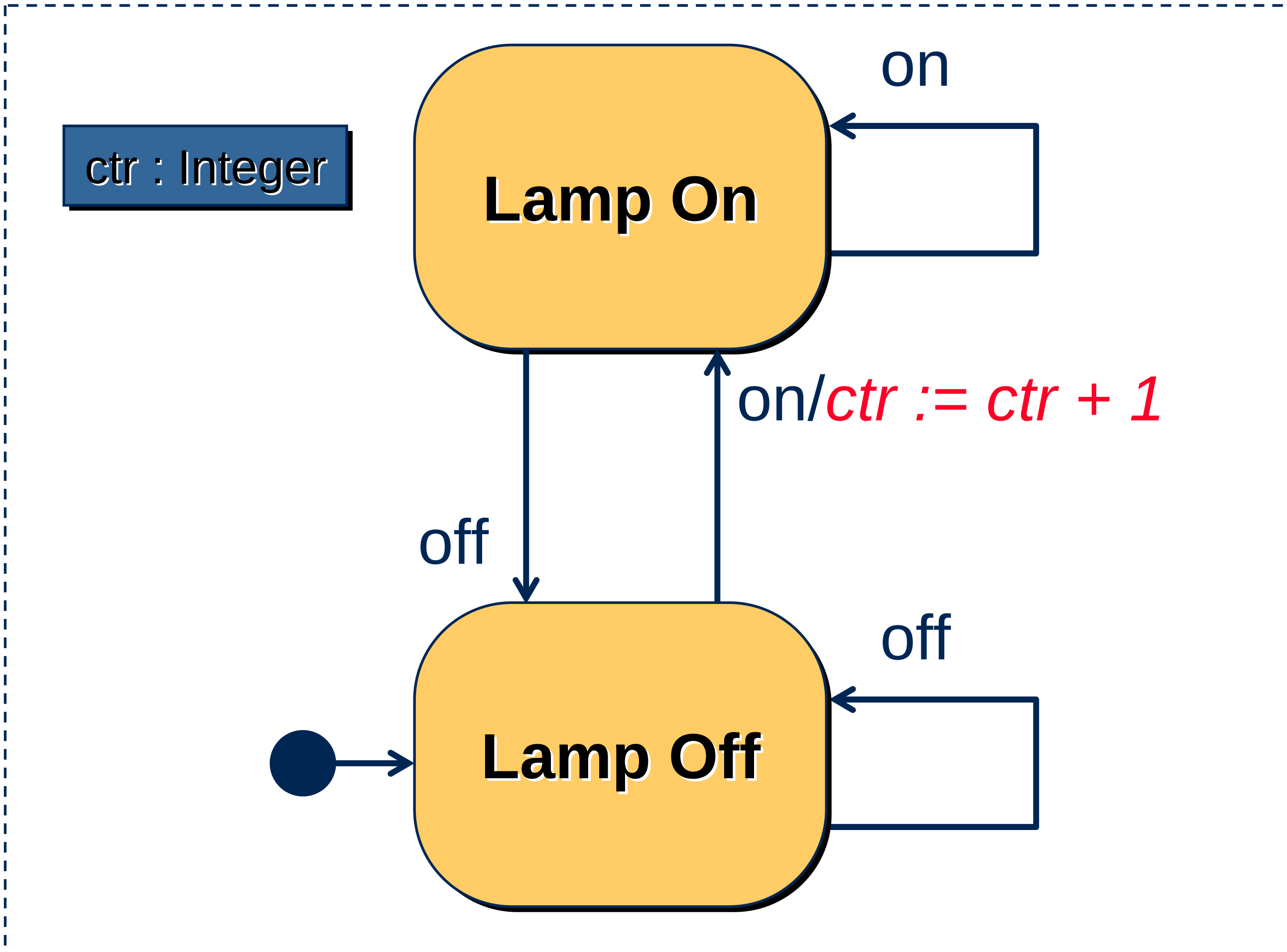
Types of State Machines



Mealy Automaton **Moore** Automaton

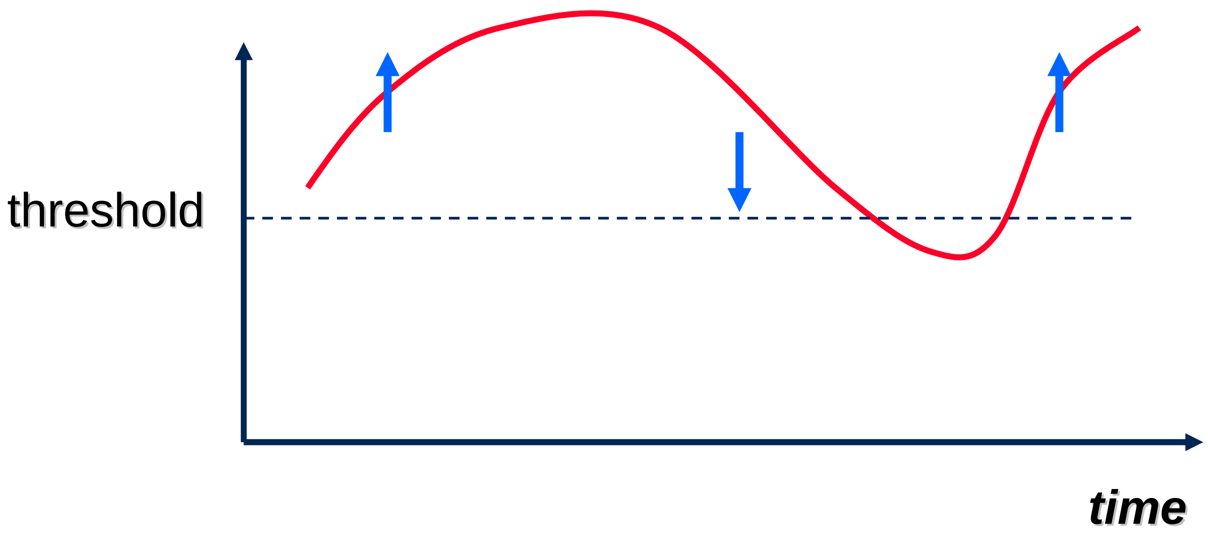
- UML allows both types to be mixed

Variables ("Extended" States)

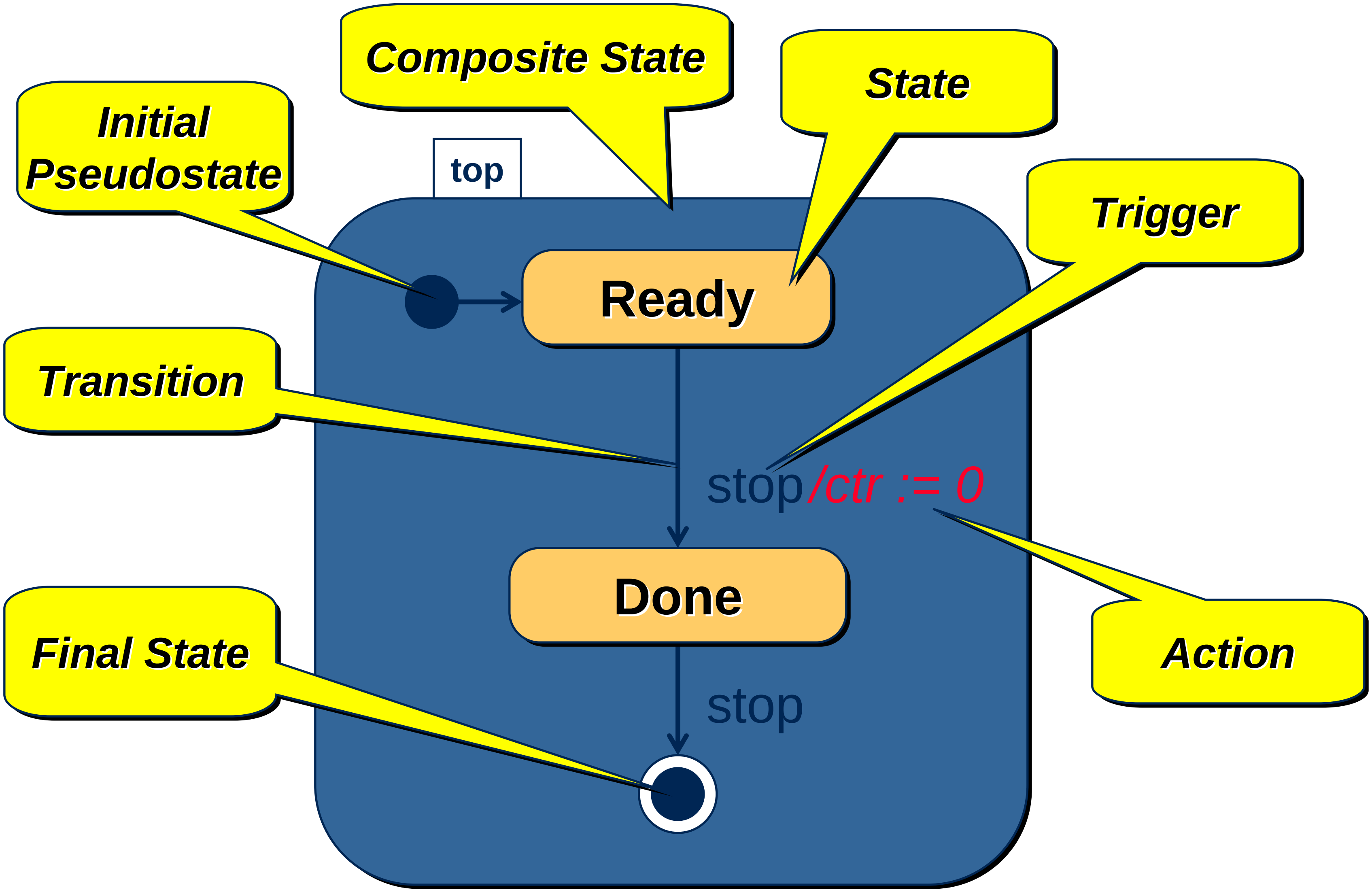


Modeling Behavior

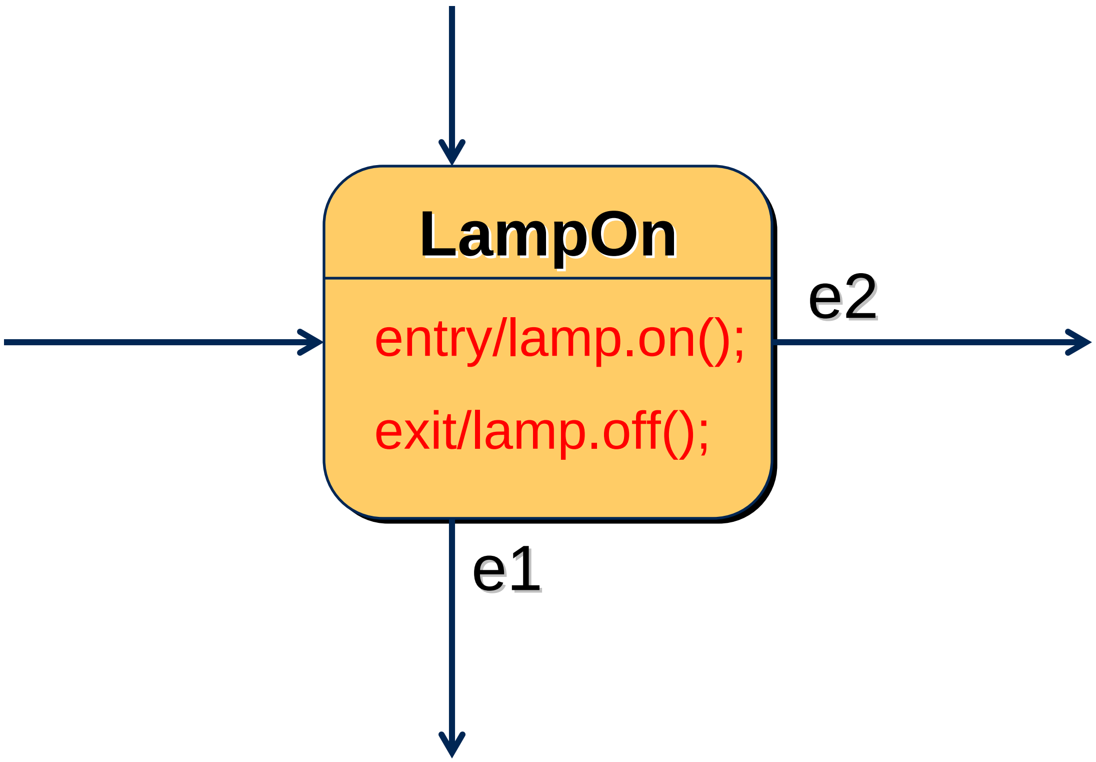
- In general, state machines are suitable for describing reactive systems based or events
- Not appropriate to describe continuous systems (e.g., spacecraft trajectory control, stock market predictions)



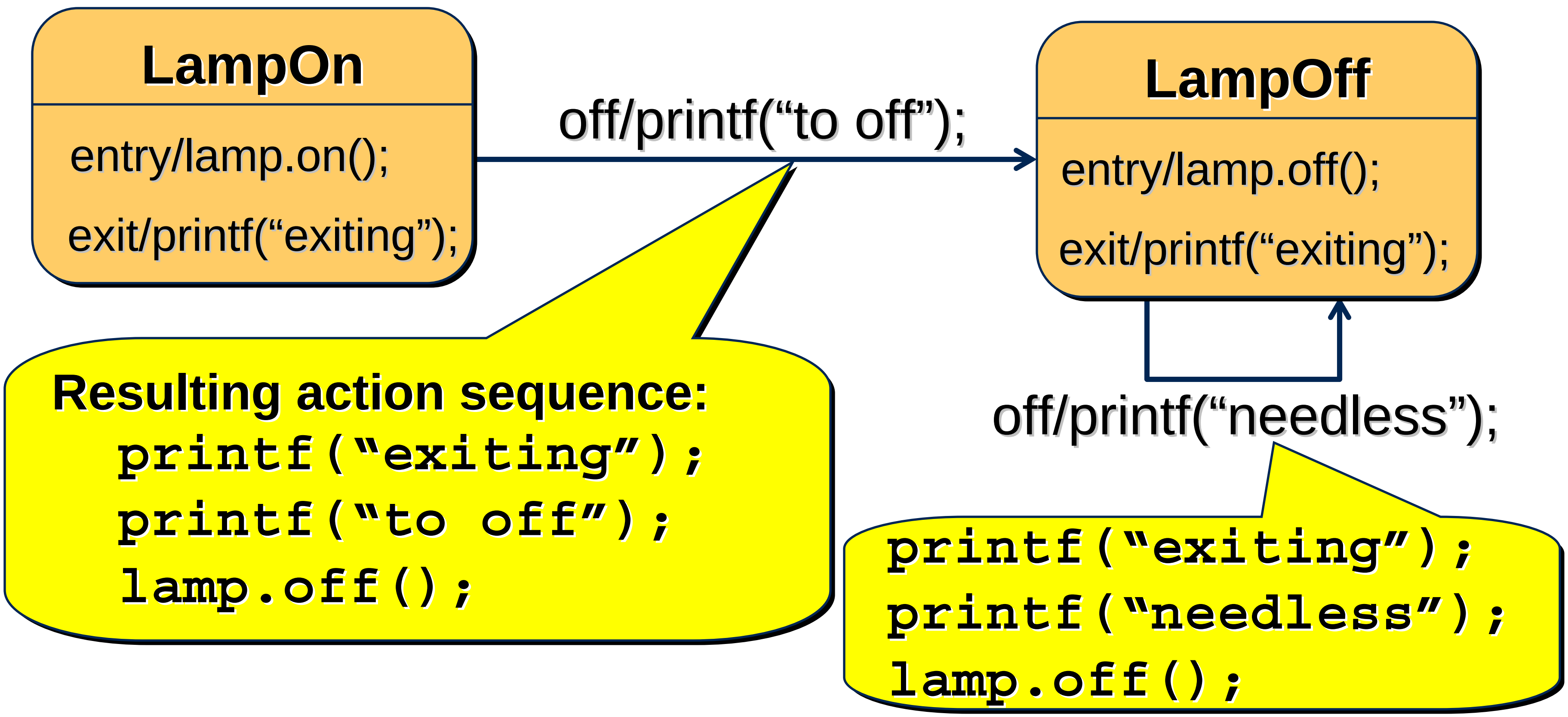
UML State Machine Diagrams – Summary



Entry and Exit Actions



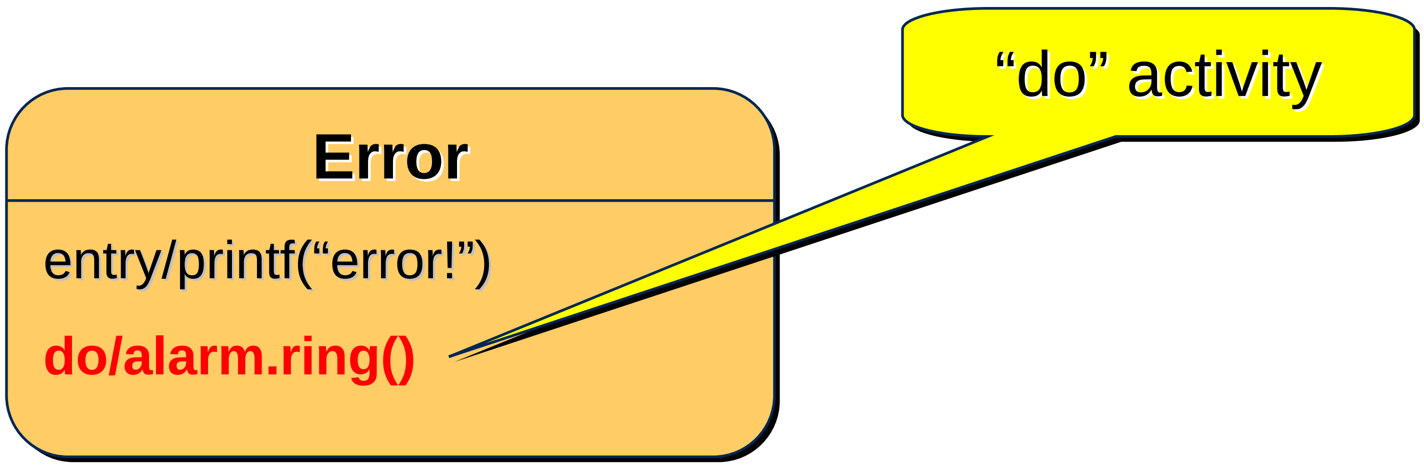
Action Ordering



- Output actions: transition prefix
- Input actions: transition postfix

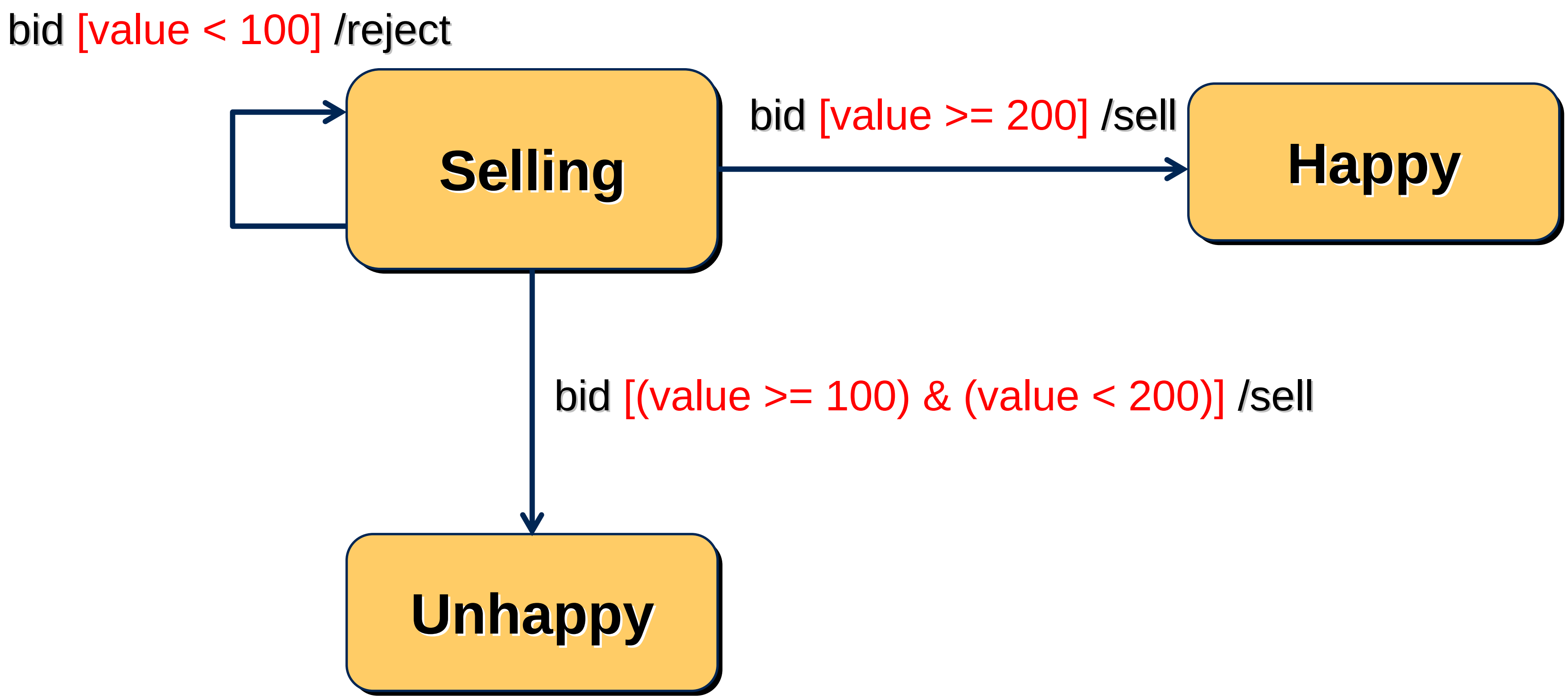
State Activity (Do)

- Creates a concurrent process that will execute until
  - The action terminates, or
  - We leave the state via an exit transition



Guards (Conditions)

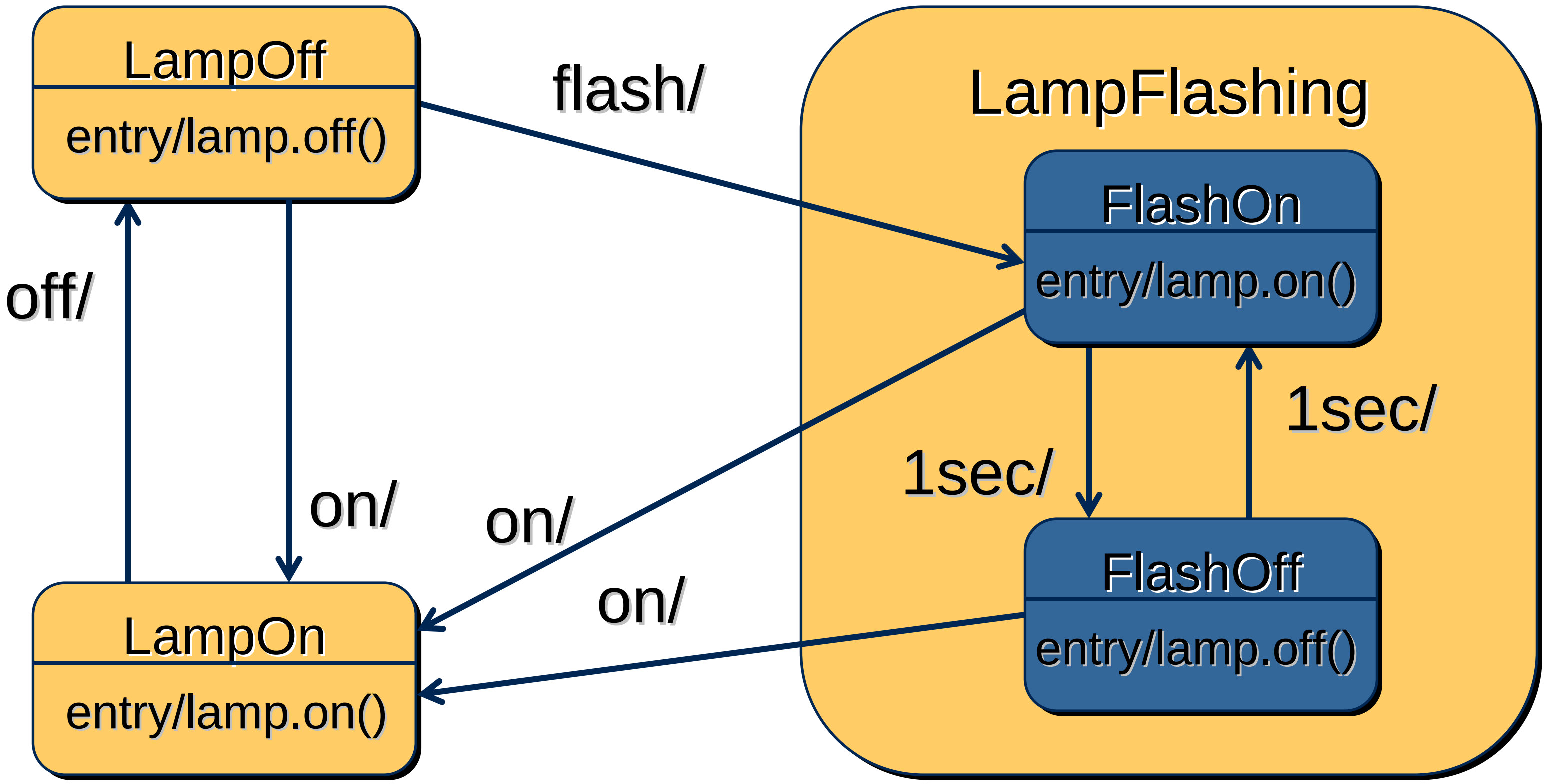
- Conditional execution of transitions



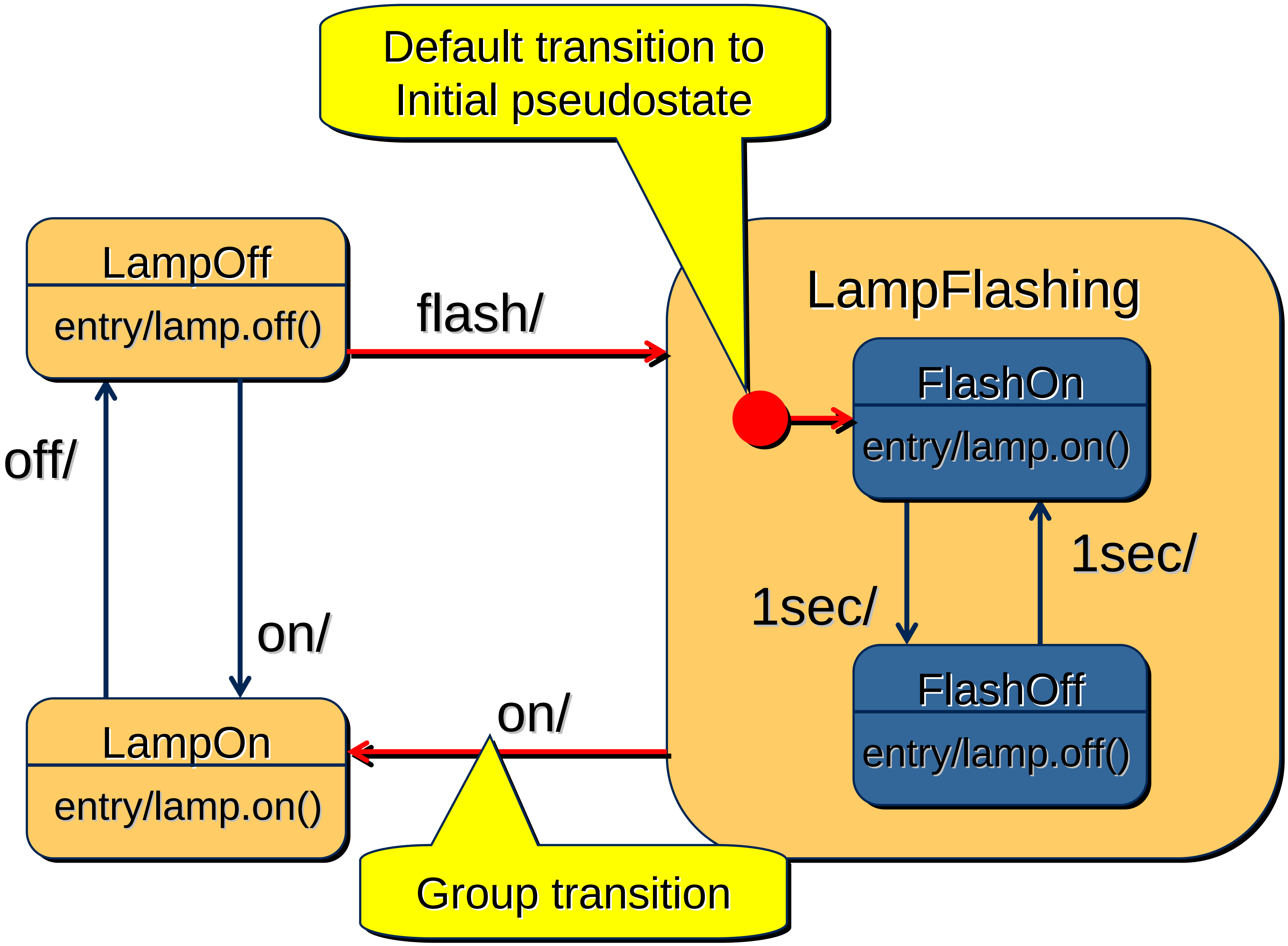
- Guards must not have side effects

Hierarchical State Diagrams

- Composed states, to manage complexity

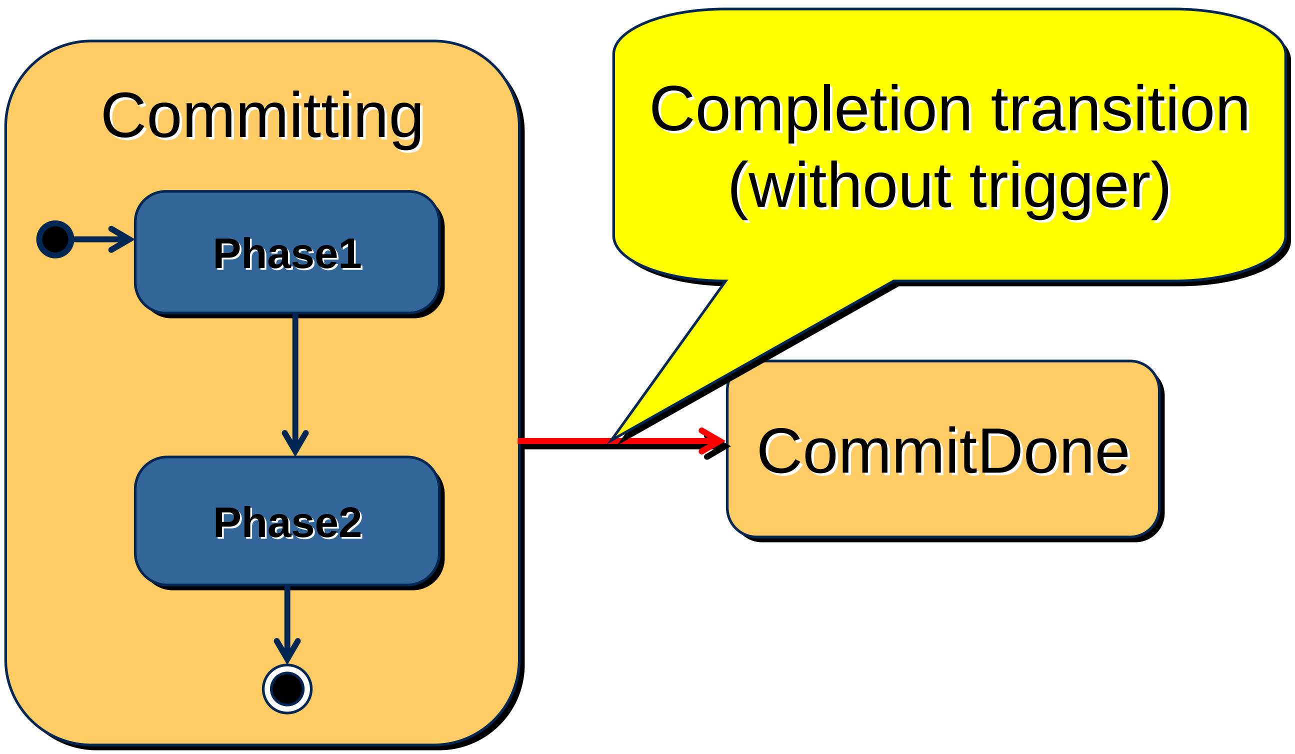


Group Transitions



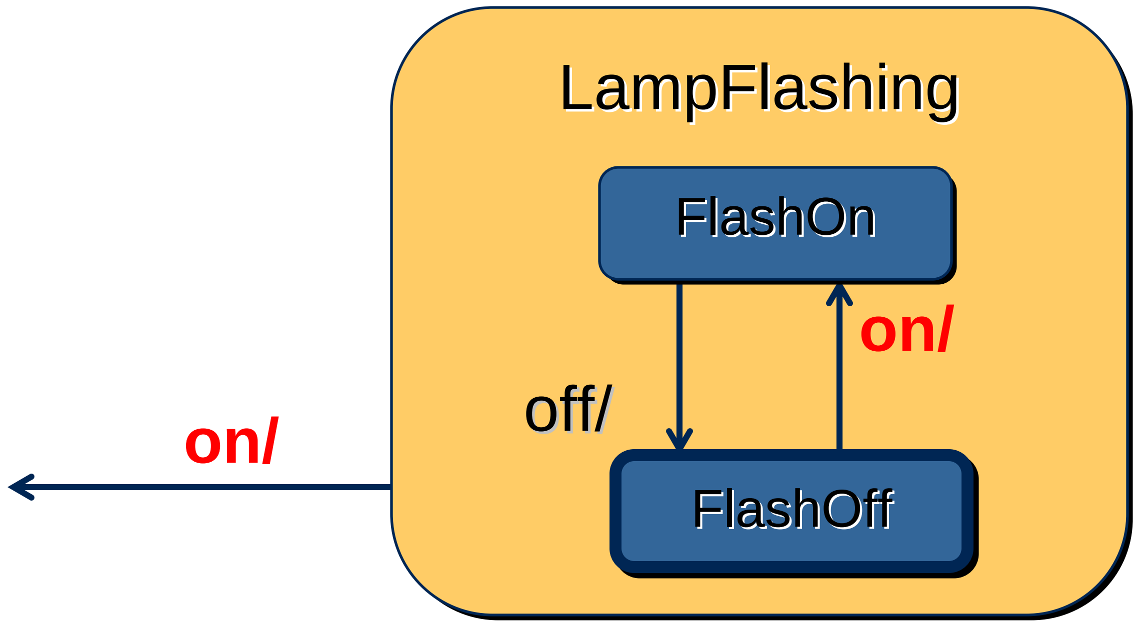
Completion Transition

- Triggered by a completion event
  - Automatically generated when an embedded state machine terminates

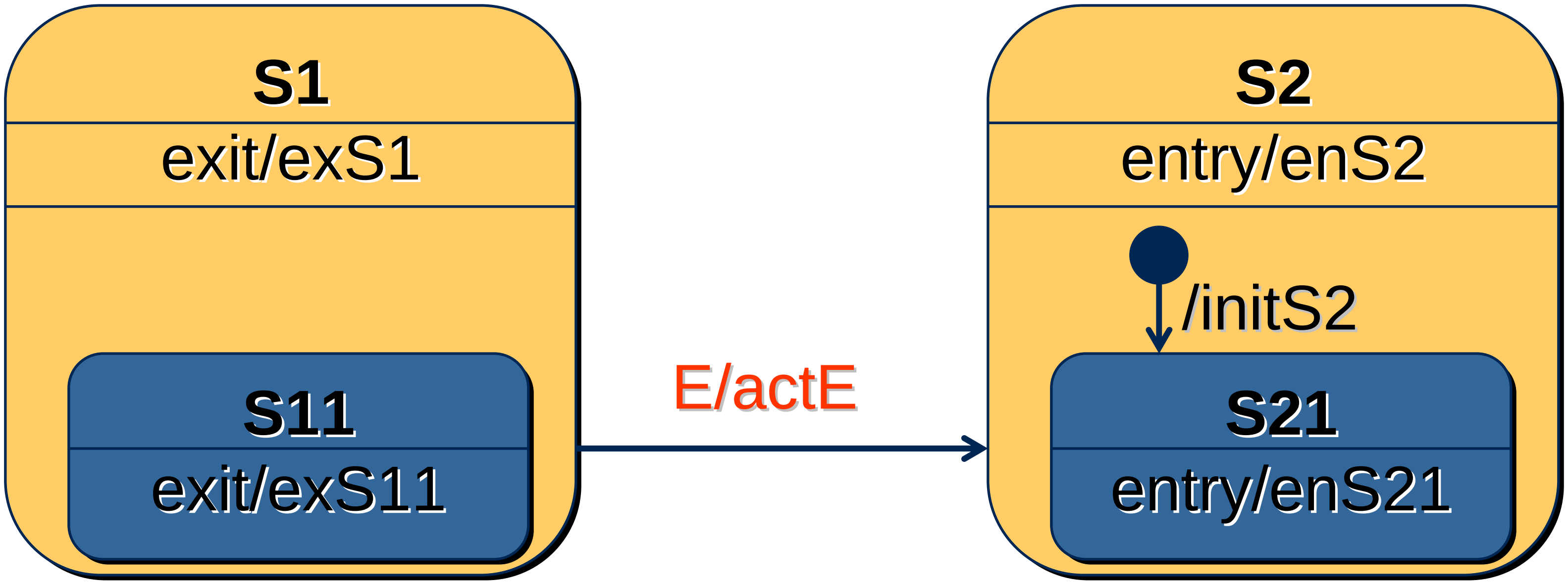


Triggering Rules

- Many transitions can share the same triggering event
  - When leaving, the most deeply embedded one takes precedence
  - The event disappears whether it triggers a transition or not



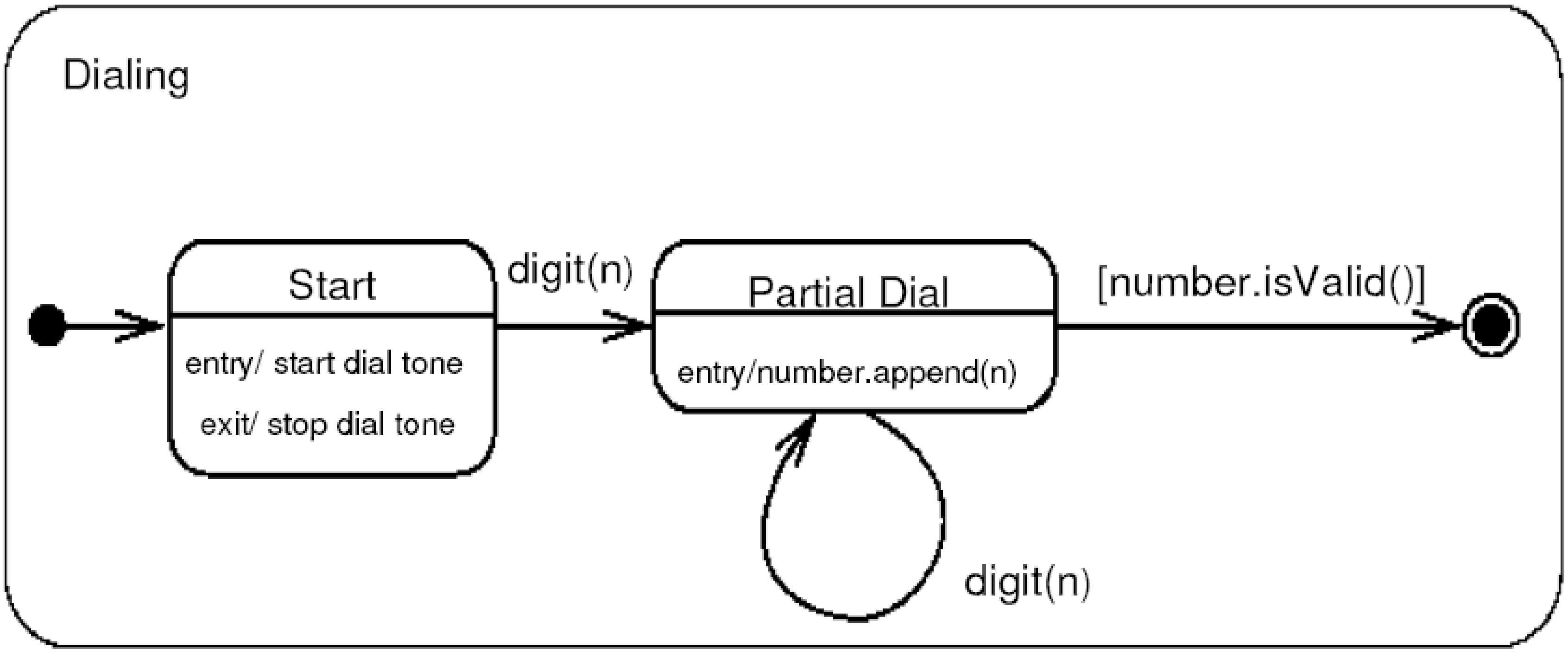
Action Ordering – Composite States



Action sequence on transition E:

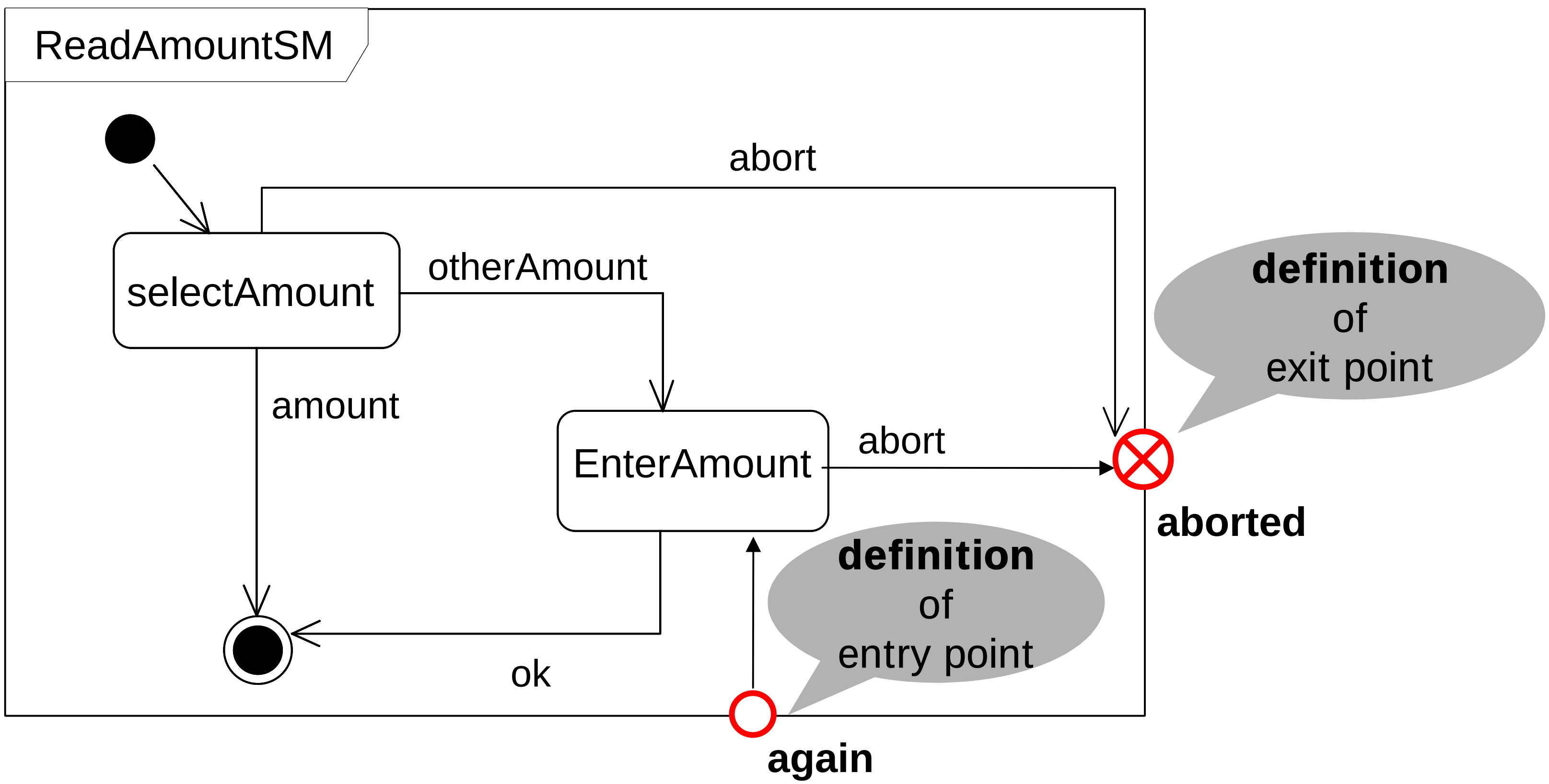
exS11 ⇨ exS1 ⇨ actE ⇨ enS2 ⇨ initS2 ⇨ enS21

Exercise I – Describe this Behavior



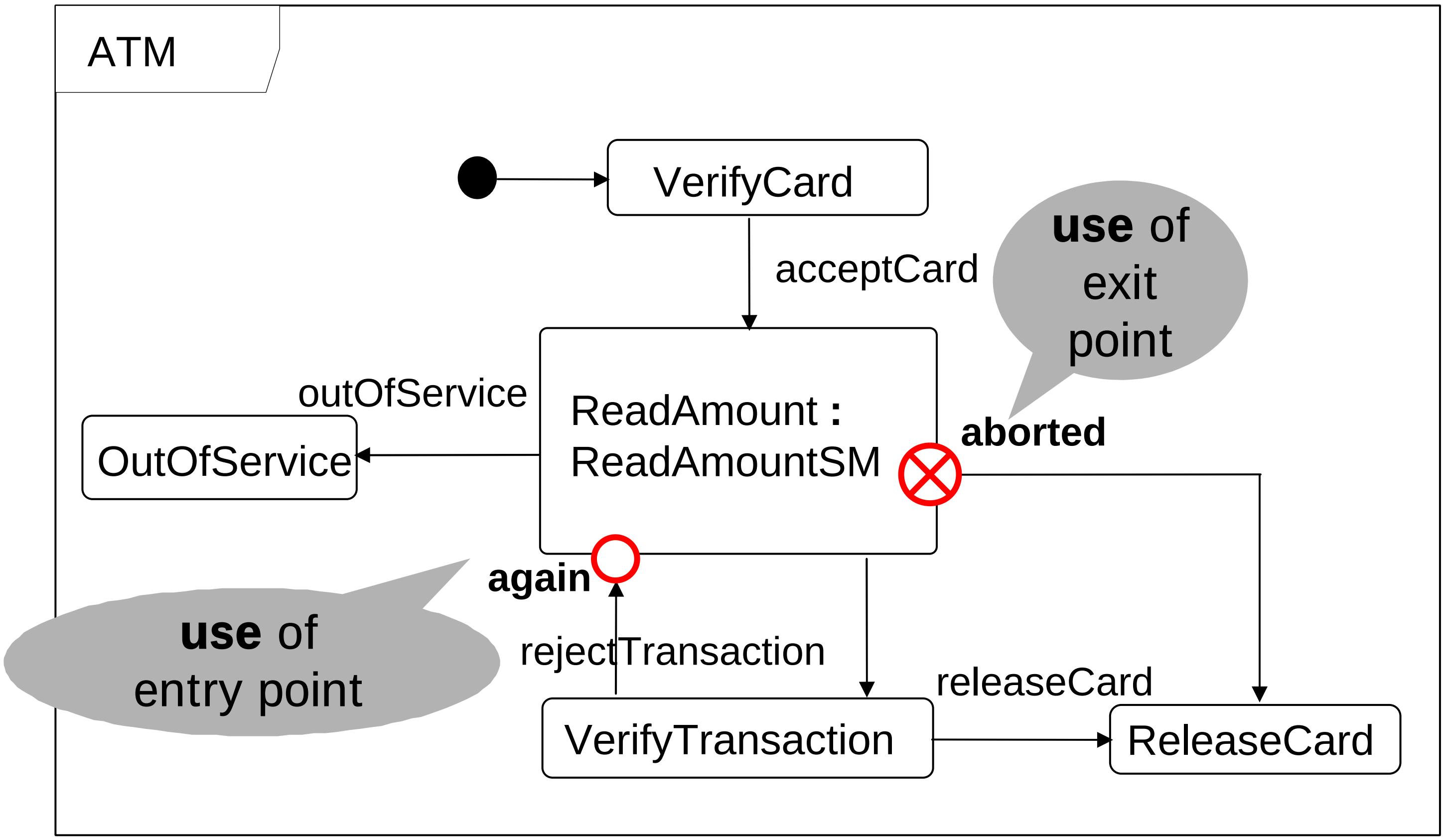
- What should be added to this state machine to more fully describe the dialing behavior?

"Reusable" State Machines (1)



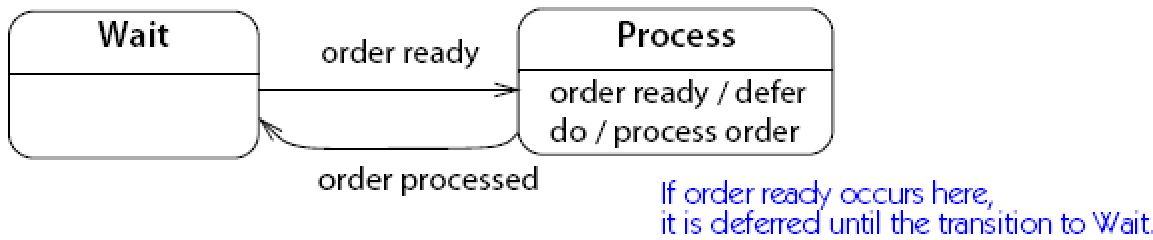
"Reusable" State Machines (2)

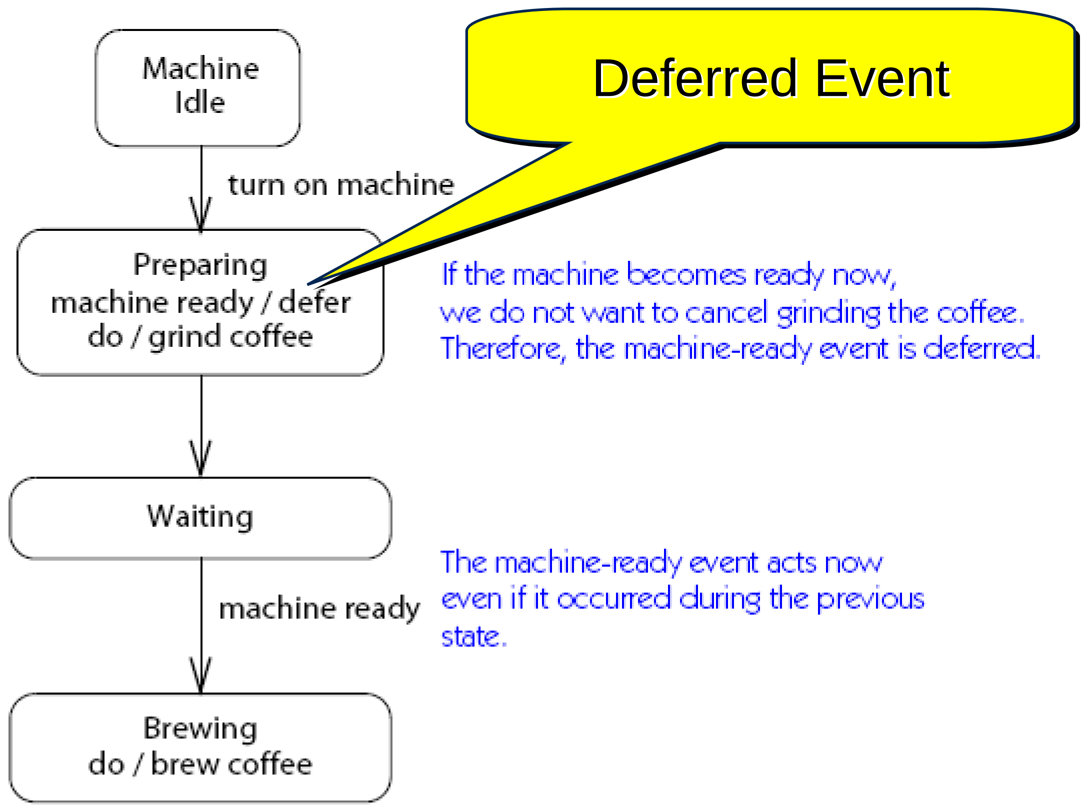
- Encapsulation of a sub-state machines



Deferred Events (this is called SAVE in SDL)

- Explicitly select event for later treatment
- An event may be deferred if it does not trigger a transition in the state which defers the event
- Saved until the system is in a state that does not defer the event
- Possible to model a required event that may occur before or after another required event

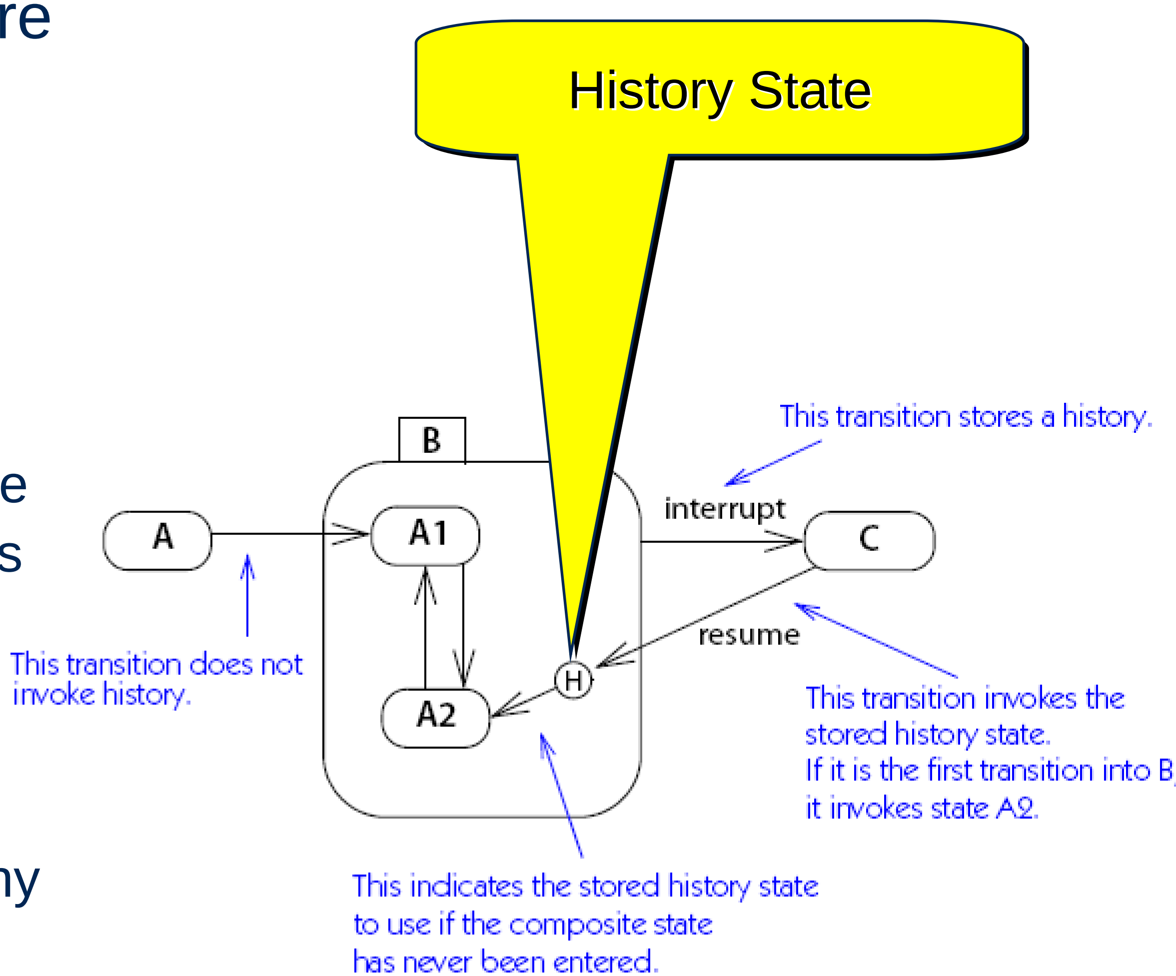




Source: UML Reference Manual

History State

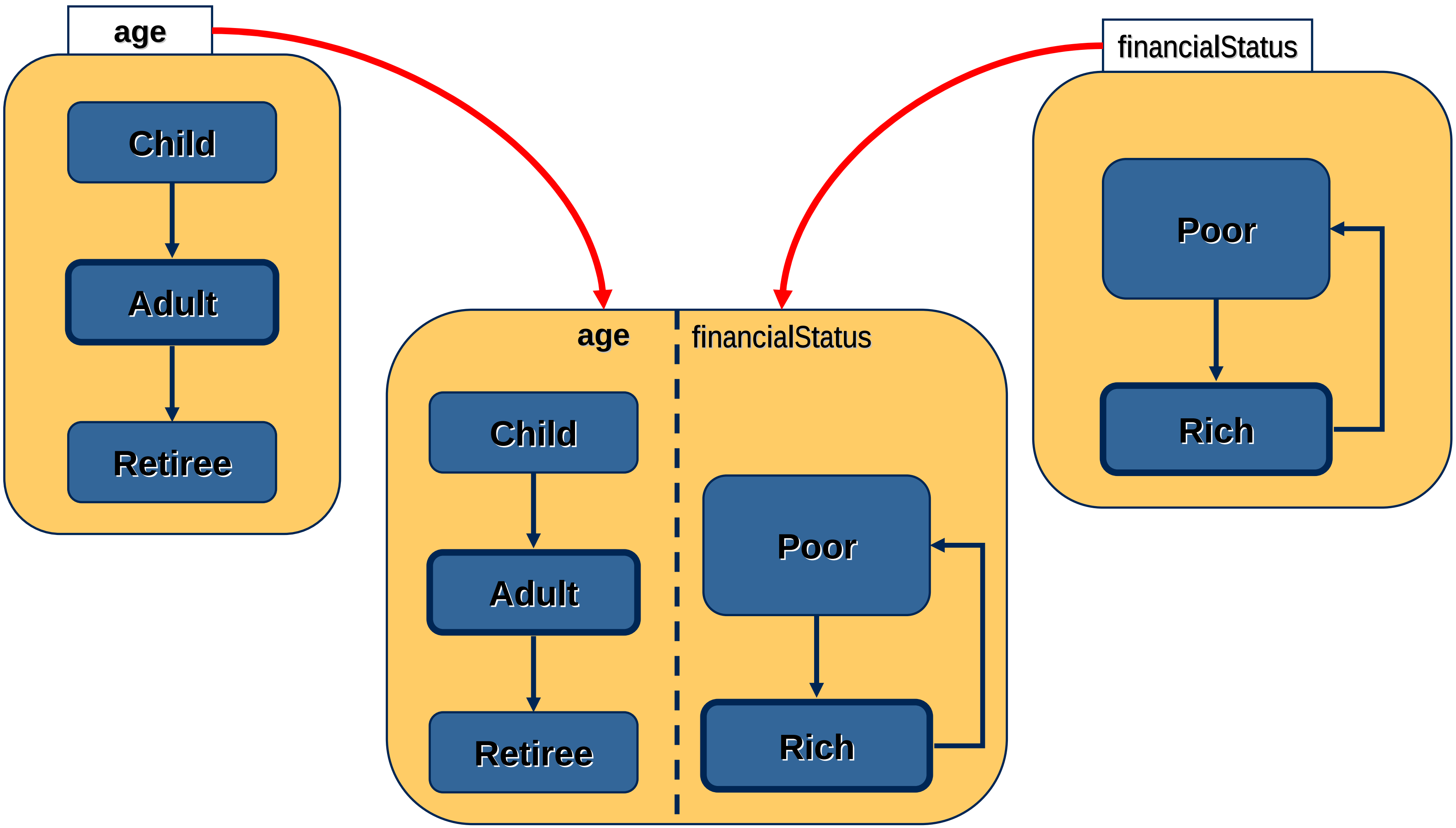
- Remember the last active sub-state before the most recent exit from the composite state
- Shallow history (H)
  - Remember state at the same nesting depth as history state
- Deep history (H*)
  - Remember state at any nesting depth



Source: UML Reference Manual

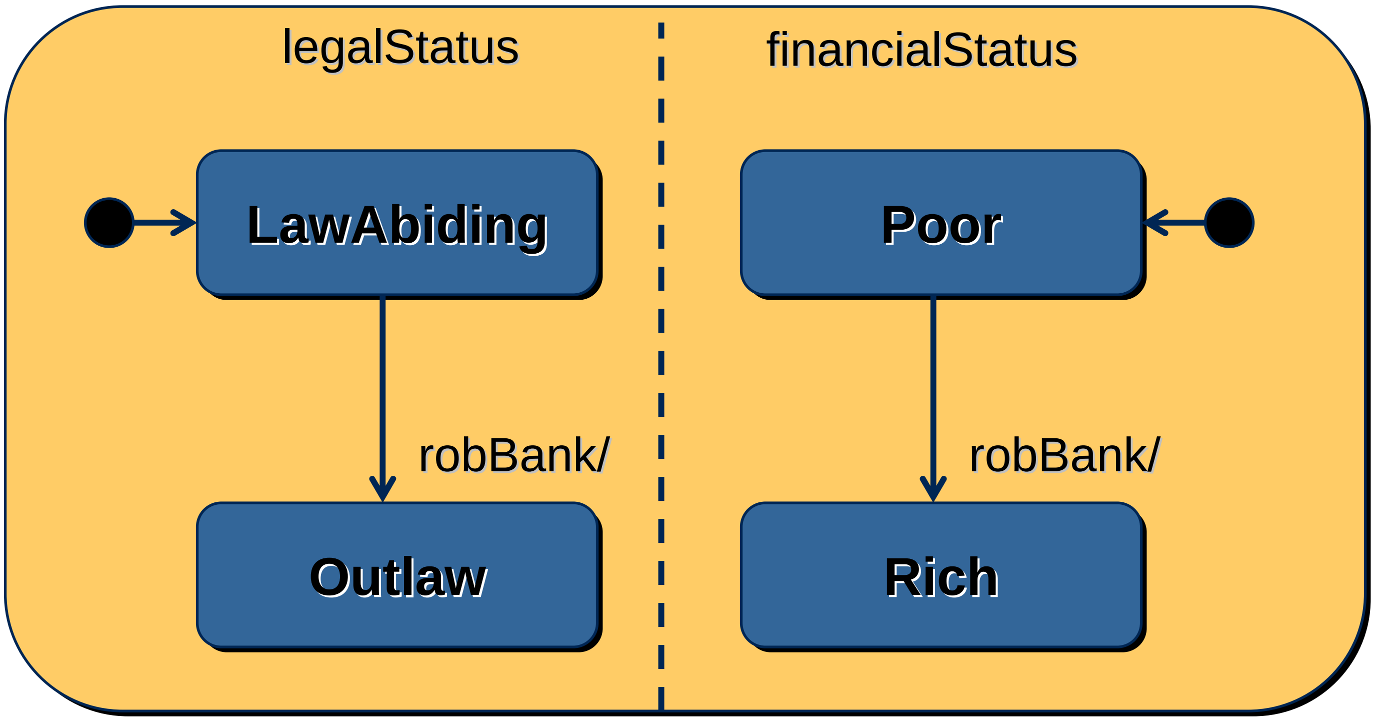
Orthogonal Regions

- Combine many concurrent perspectives – interactions across regions typically done via shared variables



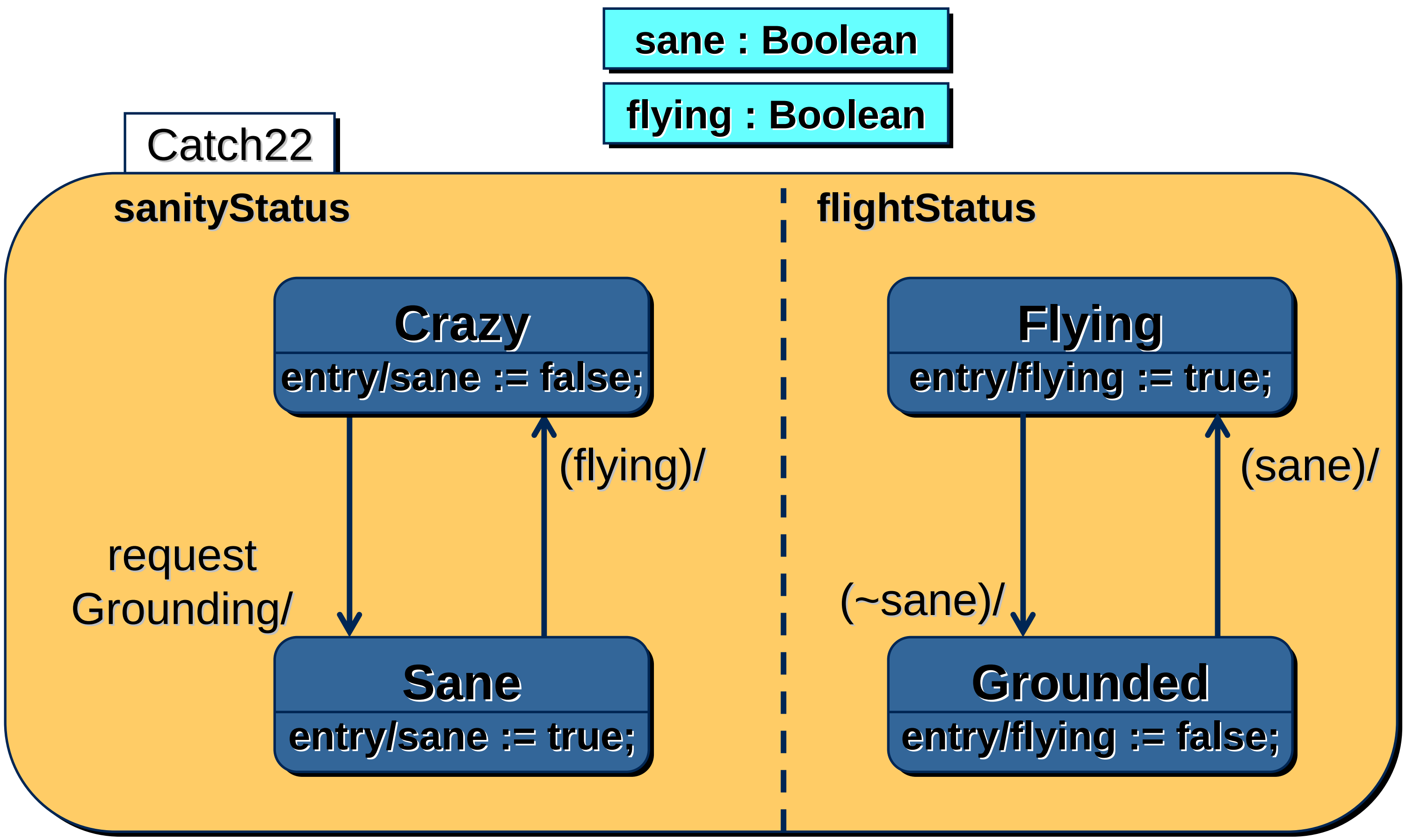
Semantics of Orthogonal Regions

- All mutually orthogonal regions detect the same events and respond simultaneously (possibly interleaved)

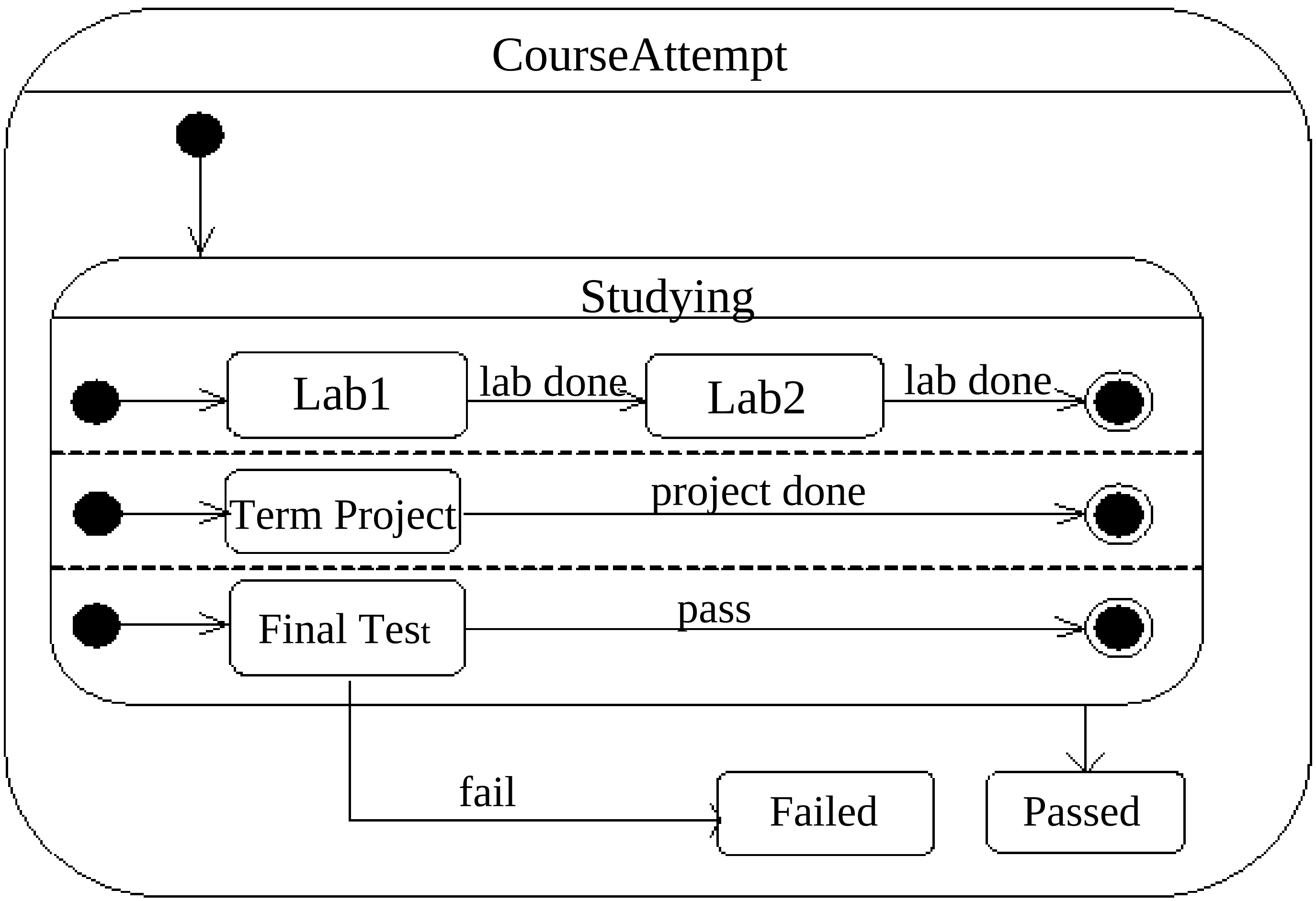


Interactions Between Regions

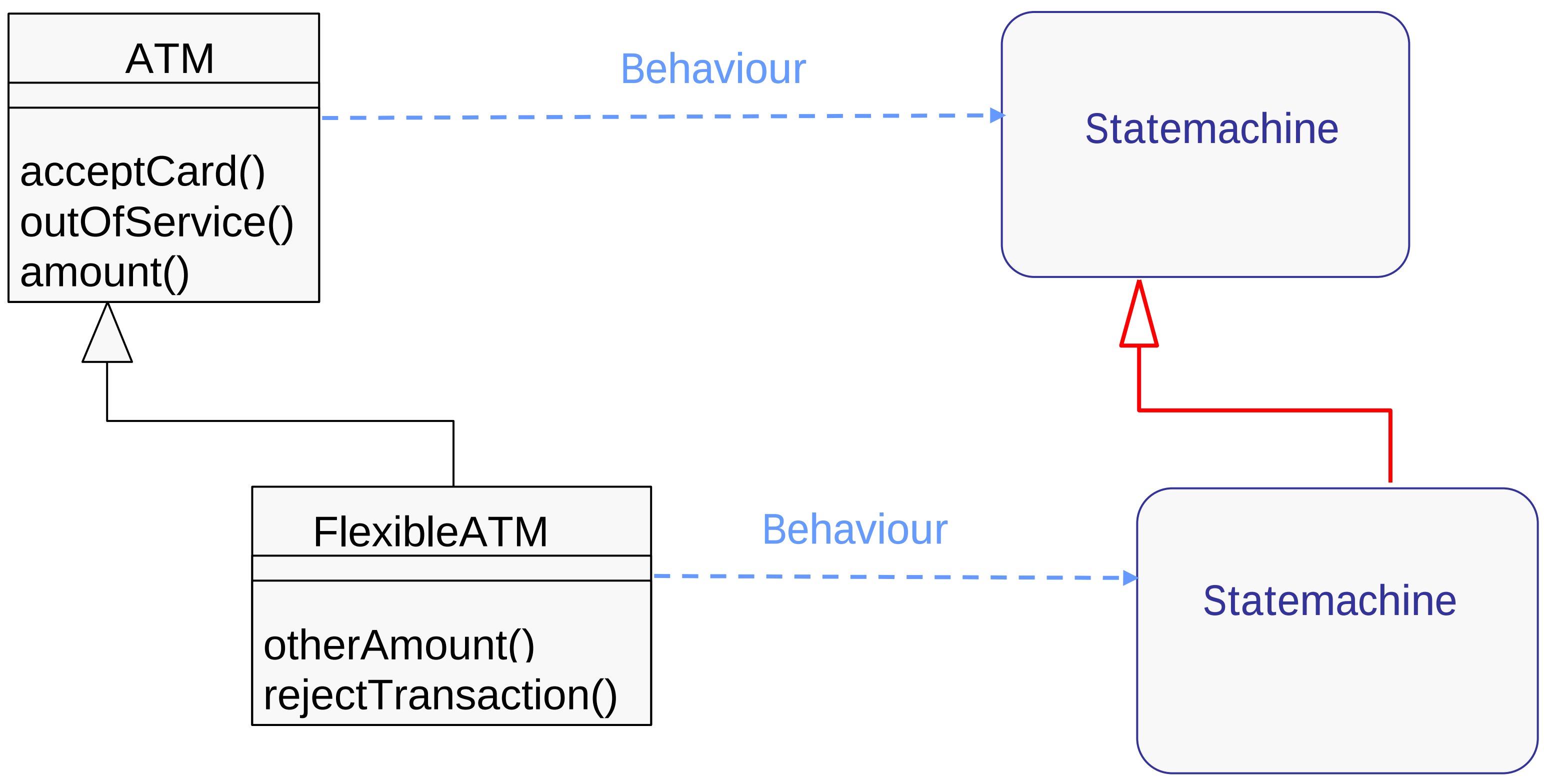
- Typically through shared variables



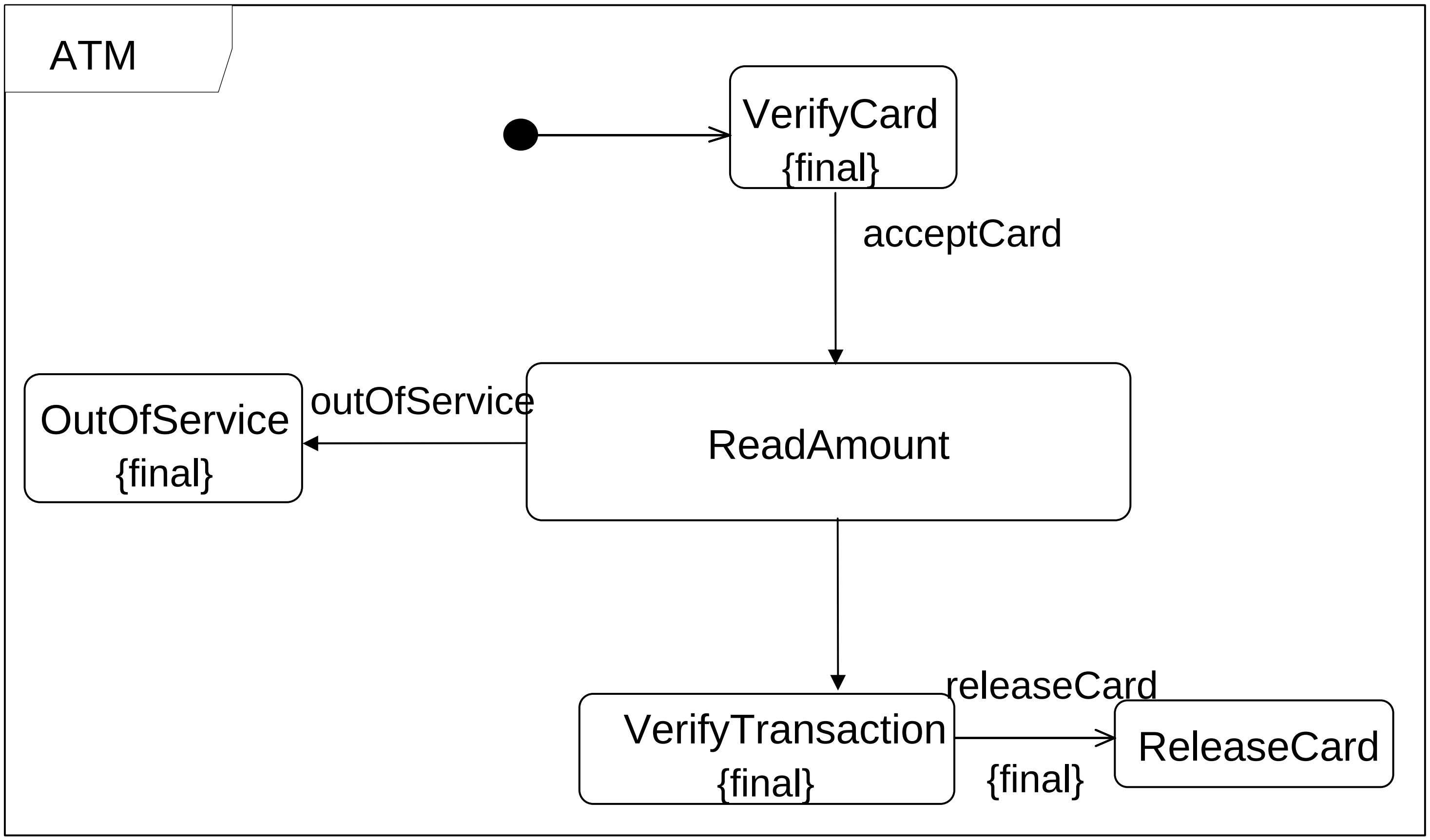
Exercise II – Describe this Behaviour



Advanced Notation: State Machine Inheritance

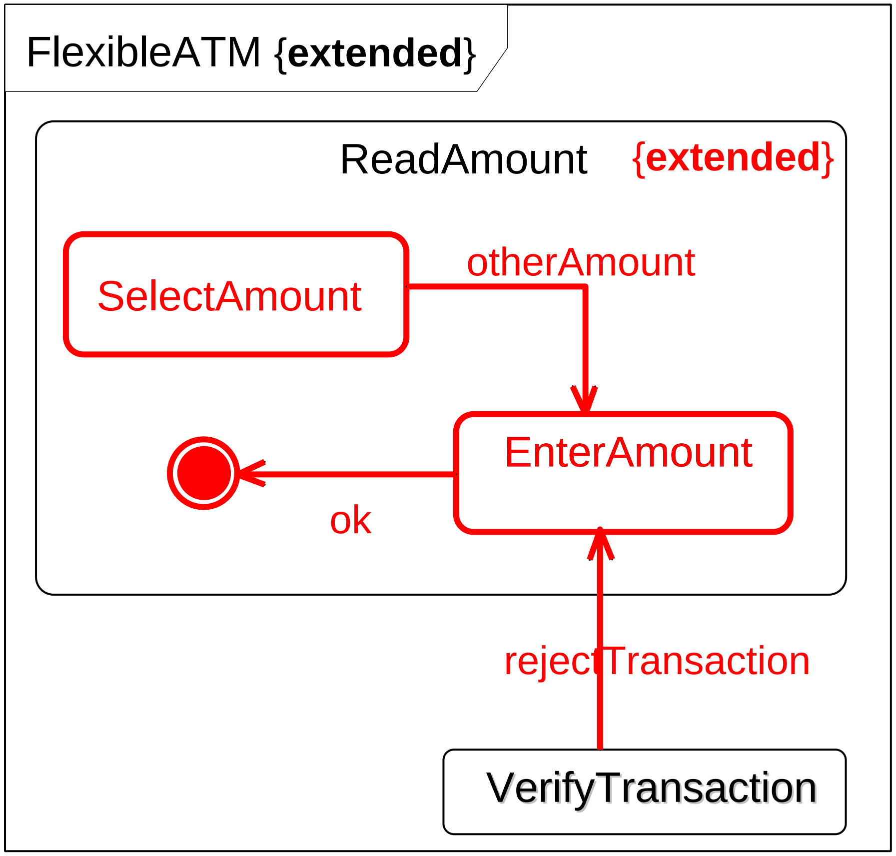


State Machine Inheritance – ATM



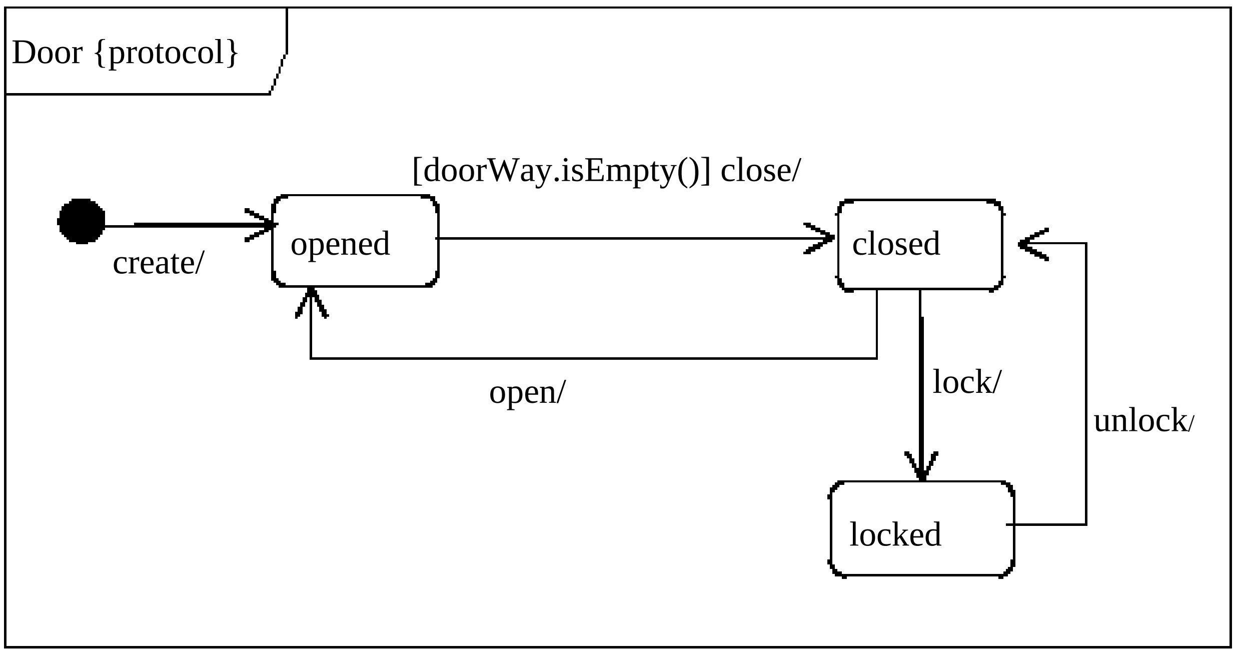
State Machine Inheritance – FlexibleATM

- States can be added and extended
- Regions can be added and extended
- Transitions can be added or extended
  - Actions may be replaced
  - Guards may be replaced
  - Targets may be replaced
- **Be very careful.** One would like that all properties that can be proved for the abstract model, also hold for the detailed model (and possibly more properties). But this is not true in general – it depends on what extensions have been made.



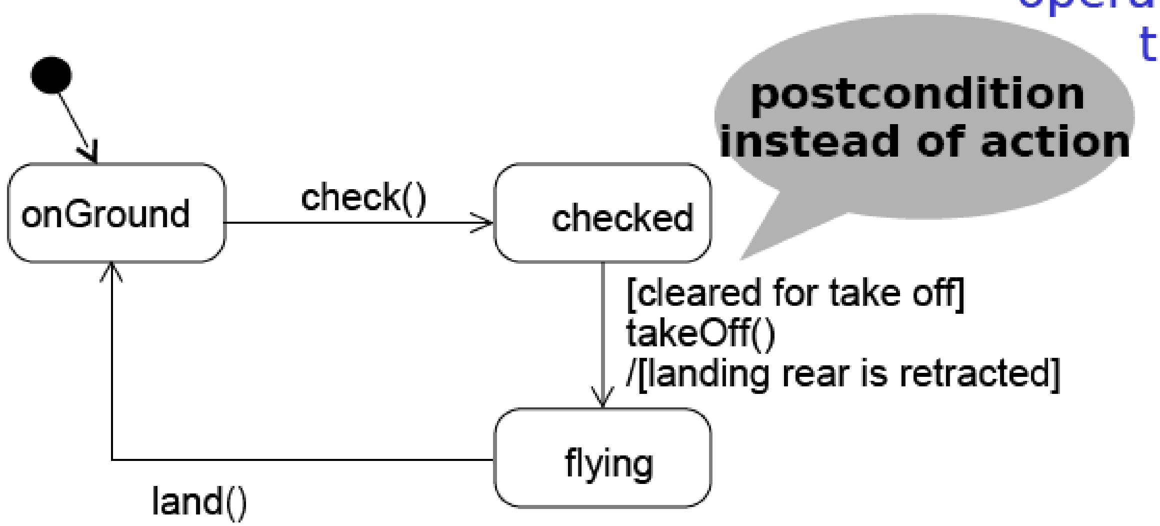
Protocol State Machine

- Specifies which operations can be called in which state and under which condition
  - Allowed call sequences – legal transitions, order of invocation of operations
  - Transitions do not include *actions*
- May be associated with ports



Protocol State Machine – Pre/Postconditions

- Transitions specification may include pre- and postconditions



Equivalent to pre and post conditions added to the related operations:

takeOff()

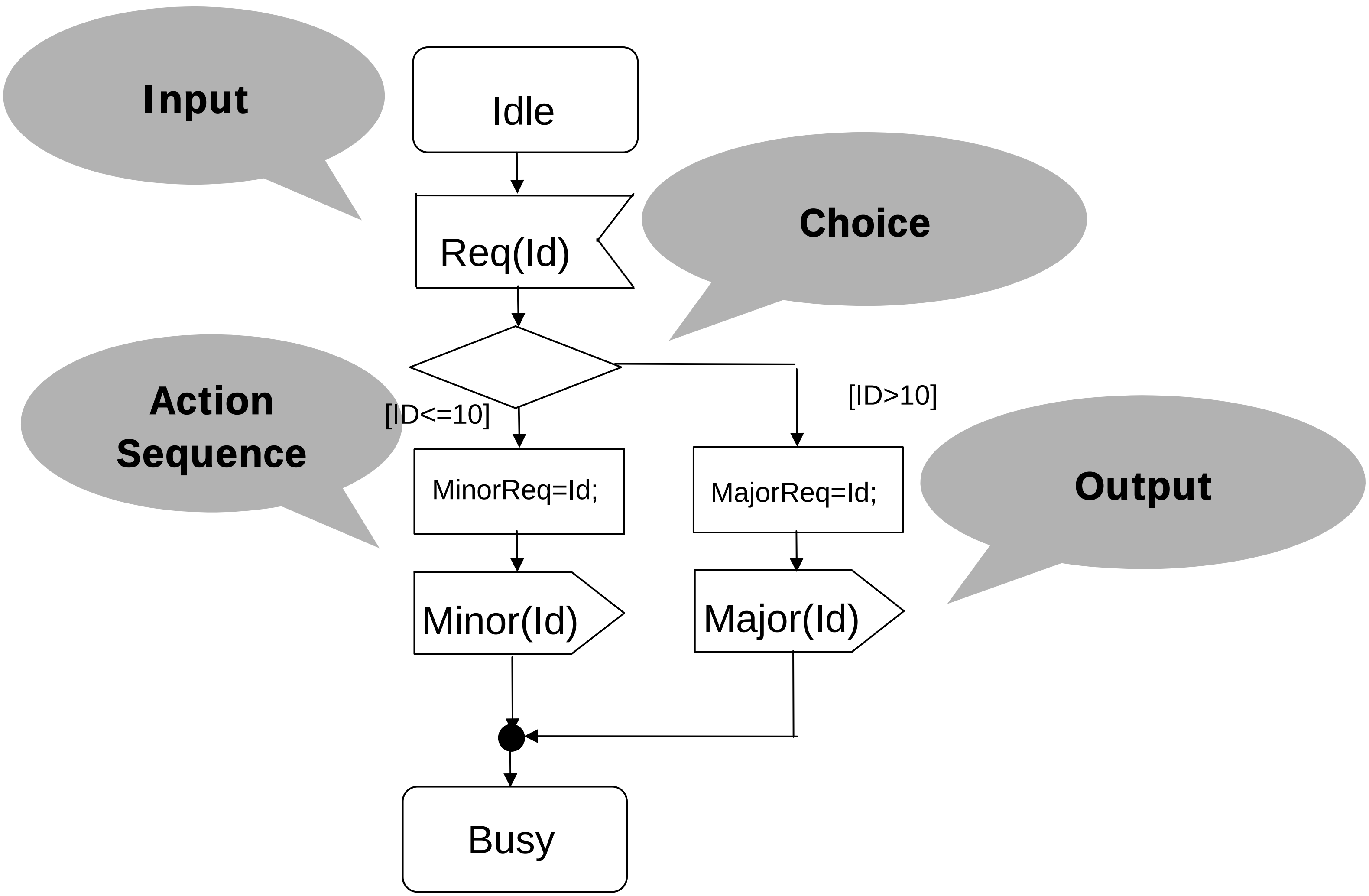
Pre

-in state "checked"
-cleared for take off

Post

-landing rear is retracted
-in state "flying"

Alternative Notation (à la SDL)



State Machine-Based Analysis

State Machine-Based Analysis (1)

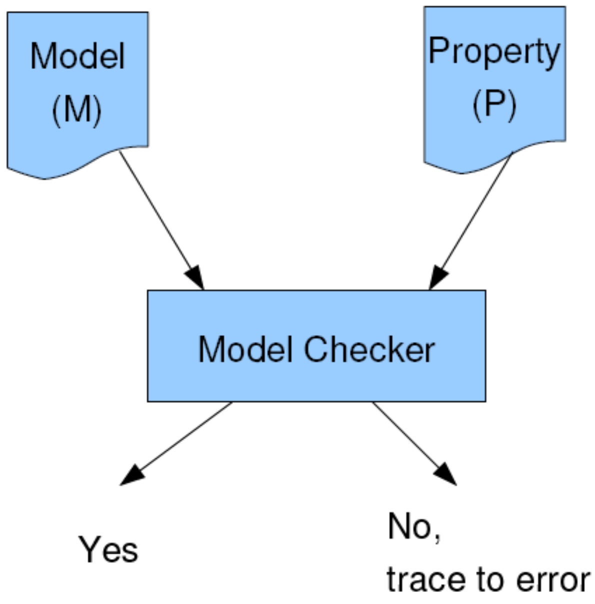
- Several possible alternatives which depend on the formalisms and tools
  - Simulation
    - Let the behavior evolve more or less randomly
    - Can be interactive
  - Test
    - Verify that certain traces are supported (or rejected) by the machine
  - Reachability analysis
    - All states can be reached and all transitions can be traversed
    - No unhandled event in each state
    - Absence of deadlocks (in communicating state machines)

State Machine-Based Analysis (2)

- Conformance checking
  - Between two machines (for example, one abstract and the other one more concrete)
  - Reduce non-determinism
  - Reduce optional behavior (compliant, but some behaviors are not supported)
  - Extension (consistent, but some new events are treated and lead to new behaviors)
- Equivalence checking
  - Between two machines (for example, one abstract and the other one more concrete)
  - Several levels of equivalence: traces, refusals, tests, observational equivalence...

State Machine-Based Analysis (3)

- Model checking
  - Verifies that the model satisfies temporal logic properties, for example:
    - If A occurs, B could possibly occur
      If C occurs, D always occurs
  - Traverse systematically all possible behaviors (execution paths) of the machine
    - Generated in advance or on the fly



  - Model checker verifies M ⇒ P (if not a trace of states and transitions leading to the violation of P is produced)
  - Major obstacle is state explosion
- Theorem proving
  - Prove by deduction or other formal approaches some properties of the state machine - tools often allow interactive proving