

Module 3: Advanced Features – Part II: Behavioral Diagrams

3 basic building blocks of UML - Diagrams

Graphical representation of a set of elements.

Represented by a connected graph: Vertices are things; Arcs are relationships/behaviors.
5 most common views built from

UML 1.x: 9 diagram types.

Structural Diagrams

Represent the *static* aspects of a system.

- Class;
- Object
- Component
- Deployment

Structural Diagrams

- Class;
- Object
- Component
- Deployment
- Composite Structure
- Package

UML 2.0: 12 diagram types



Behavioral Diagrams

Represent the *dynamic* aspects.

- Use case
- Sequence;
- Collaboration
- Statechart
- Activity

Behavioral Diagrams

- Use case
- Statechart
- Activity

Interaction Diagrams

- Sequence;
- *Communication*
- Interaction Overview
- *Timing*

Use Case Diagrams

Behavioral Diagrams

- **Use case**
- Statechart
- Activity

Interaction Diagrams

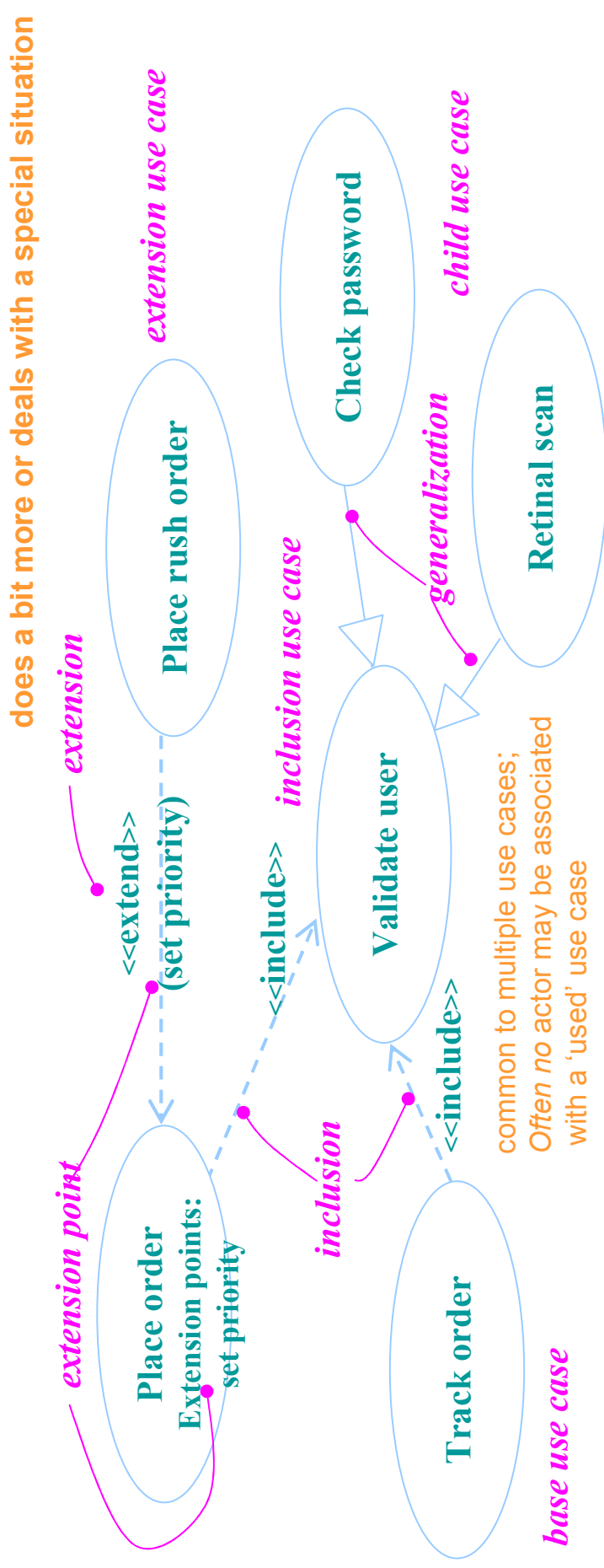
- Sequence;
Communication
- *Interaction Overview*
- *Timing*

Use Cases

- ❑ When to Use Use Cases
 - ❑ Fowler's View: do use cases first before object modeling
 - ❑ Capture the **simple, normal use-case** first
 - ❑ For every step ask "**What could go wrong?**" and how it might work out differently
 - ❑ Plot all variations as **extensions** of the given use case
 - ❑ Another view: do object modeling first, then use cases
 - ❑ Another: iterate model - use case - model - use case ...
- ❑ Scenarios describe a single path, or a particular sequence
 - ❑ E.g., Use Case: Order Goods
 - ❑ Scenario 1: all goes well
 - ❑ Scenario 2: insufficient funds
 - ❑ Scenario 3: out of stock
- ❑ System test cases: Generate a test script for each scenario (flow of events).
 - ❑ Obtain initial state from preconditions.
 - ❑ Test success against post conditions.

Organizing Use Cases

- Generalization, Extend, Include/Use, packages



- **Track Order** - Obtain and verify the order number; For each part in the order, query its status, then report back to the user.
- **Place Order** - Collect the user's order items. (set priority). Submit the order for processing.

A Use Case Template

(http://www.bredemeyer.com/pdf_files/use_case.pdf)

Use Case	Identifier: e.g., “Withdraw money”; ref # = wm3; mod history = ...
Actors	List of actors involved in use case
Brief description	Goal: E.g., “This use case lets a bank account owner withdraw money from an ATM machine”; Source: Bank doc 2.3
Preconditions	What should be true before the use case can start.
Postconditions	What should hold after the use case successfully completes.
Basic flow of events	The happy/sunny day flow. The most common successful case.
Alt. flow of events /subflows	Difference for the specific subflow
Exception flows	Subflows may be divided into 1) normal, 2) successful alternate actions, and 3) exception/error flows.
Non-Functional (optional)	List of NFRs that the use case must meet
Issues	List of issues that remain to be resolved

A Use Case Template

(http://www.bredemeyer.com/pdf_files/use_case.pdf)

Use Case (id, ref#, mod history)	2. Repairing_Cellular_Network History created 1/5/98 Derek Coleman, modified 5/5/98
Description (goal, source)	Operator rectifies a report by changing parameters of a cell
Actors	Operator (primary, Cellular network, Field maintenance engineer)
Assumptions (successful use case termination condition)	Changes to network are always successful when applied to a network
Steps	<ol style="list-style-type: none"> 1. Operator notified of network problem 2. Operator starts repair session 3. REPEAT <ol style="list-style-type: none"> 3.1 Operator runs network diagnosis application 3.2 Operator identifies cells to be changes and their new parameter values 3.3 IN PARALLEL <ol style="list-style-type: none"> 3.3.1 Maintenance engineer tests network cells 3.3.2 Maintenance engineer sends fault reports <p>UNTIL no more reports of problem</p> 4. Operator closes repair session
Variations (optional)	#1. System may detect fault and notify operator or Field maintenance engineer may report fault to Operator
Non-Functional (optional)	Performance Mean: time to repair network fault must be less than 3 hours
Issues (that remain to be resolved)	What are the modes of communication between field maintenance engineer and operator

Use Case Extension	Repair_may_fail extends 2. Repairing_Cellular_Network
Description	Deals with assumption that network changes can never fail
Steps	#3.3. if the changes to network fail then the network is rolled back to its previous state
Issues	How are failures detected? Are roll backs automatic or is Operator intervention required?

Sequence Diagrams

Behavioral Diagrams

- Use case
- Statechart
- Activity

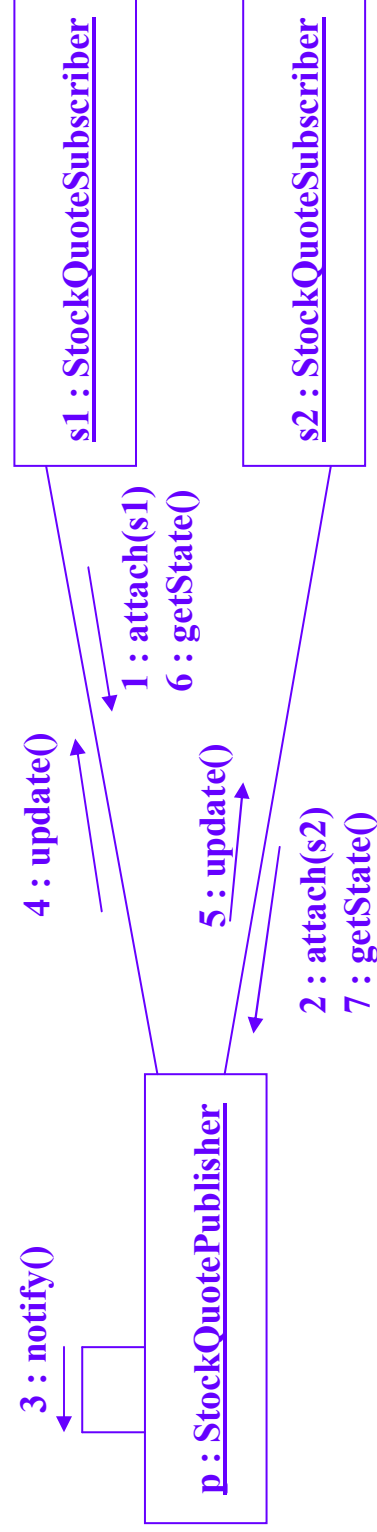
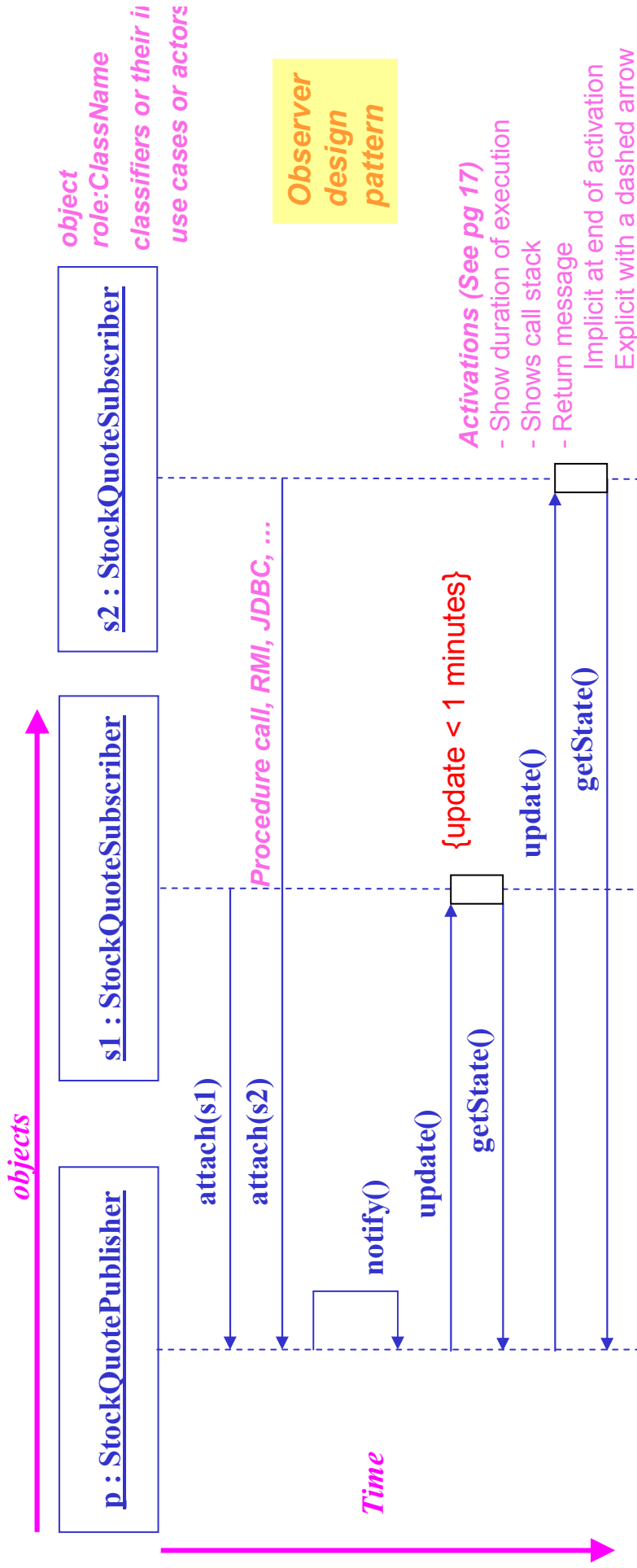
Interaction Diagrams

- **Sequence;**
Communication
- Interaction Overview
- Timing

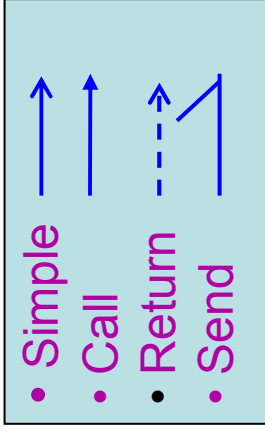
Interaction Diagrams (sd and cd)

- show the interaction of **any kind of instance** (*classes, interfaces, components, nodes and subsystems*);
- messages sent/received by those objects/instances (invocation, construction/destruction, of an operation)
- realizes use cases to model a scenario
- Interaction types (these are isomorphic, when no loops or branching)
 - **Sequence diagram** —emphasizes the **time ordering** of messages.
 - **Communication (Collaboration) diagram** — emphasizes the structural organization of objects that **directly** send and receive messages.
- Objects within an interaction can be:
 - **Concrete**: something from the real world. (e.g., [John: Person](#))
 - **Prototypical**: representative instance of something from the real world (e.g., [p: Person](#))
 - Communication diagrams use (strictly) prototypical things.
 - Prototypical instances of interfaces and abstract types are valid.

Interaction Diagram: sequence vs communication

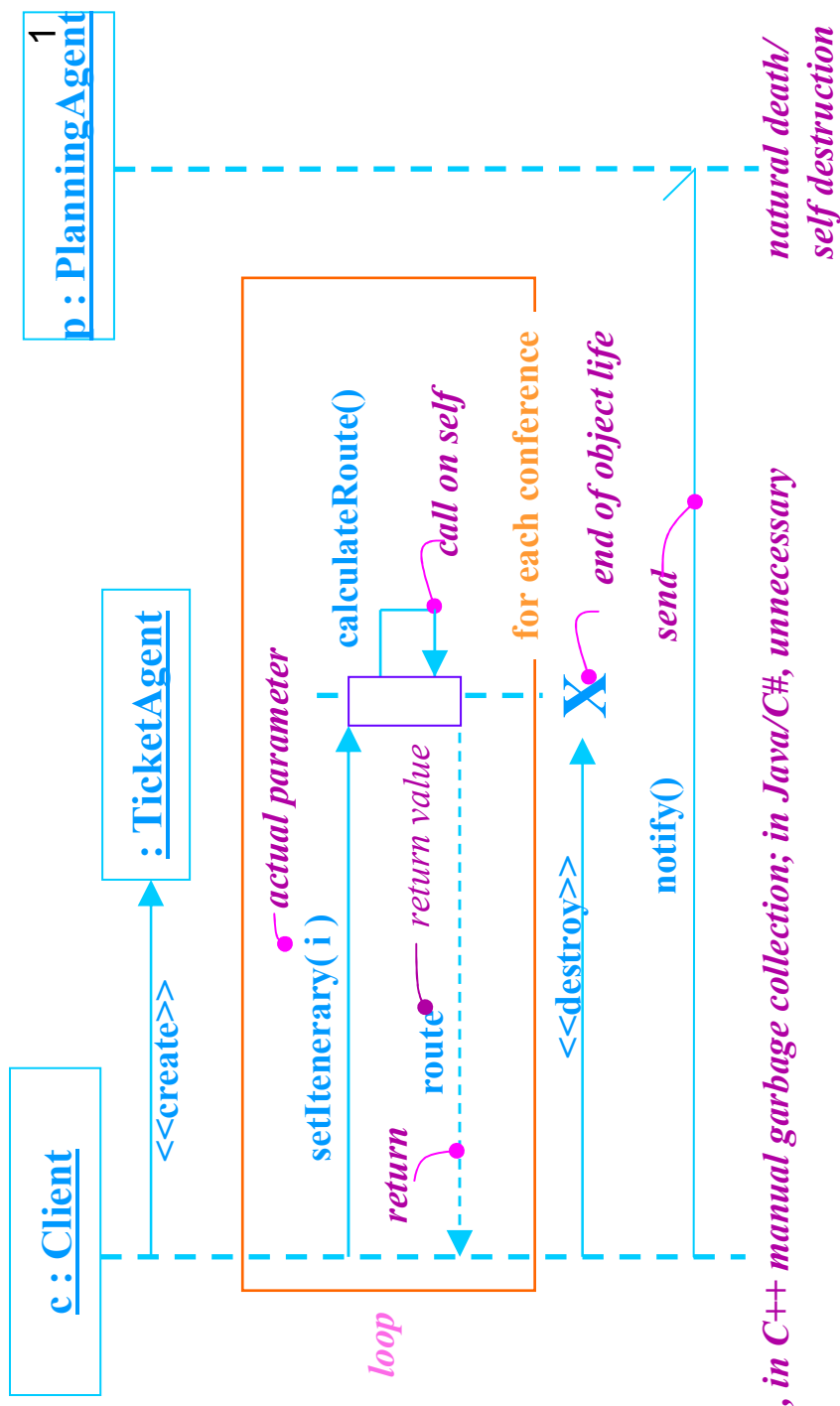


Interactions - Modeling Actions



asynchronous in 2.0 (stick arrowhead) – no return value expected at end of callee activation
 activation of caller may end before callee's (???)

half arrow in 1.x



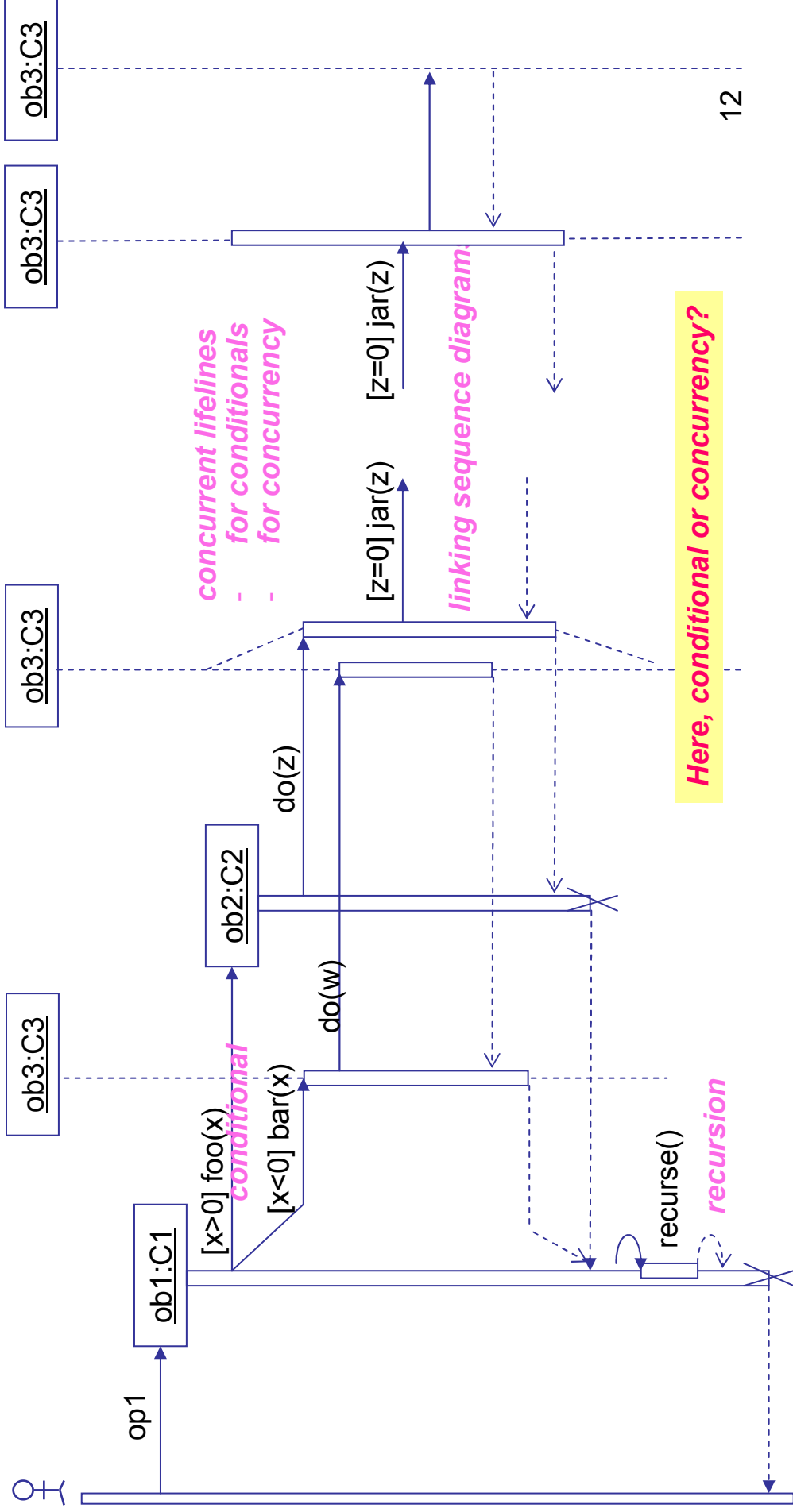
destroy: e.g., in C++ manual garbage collection; in Java/C#, unnecessary

Additional considerations

- To show nested messages, use ?.
- To show constraints like time and space, use ?.
- For formal flow of control, attach pre and post conditions to each message (?)

Sequence Diagrams – Generic vs. Instance

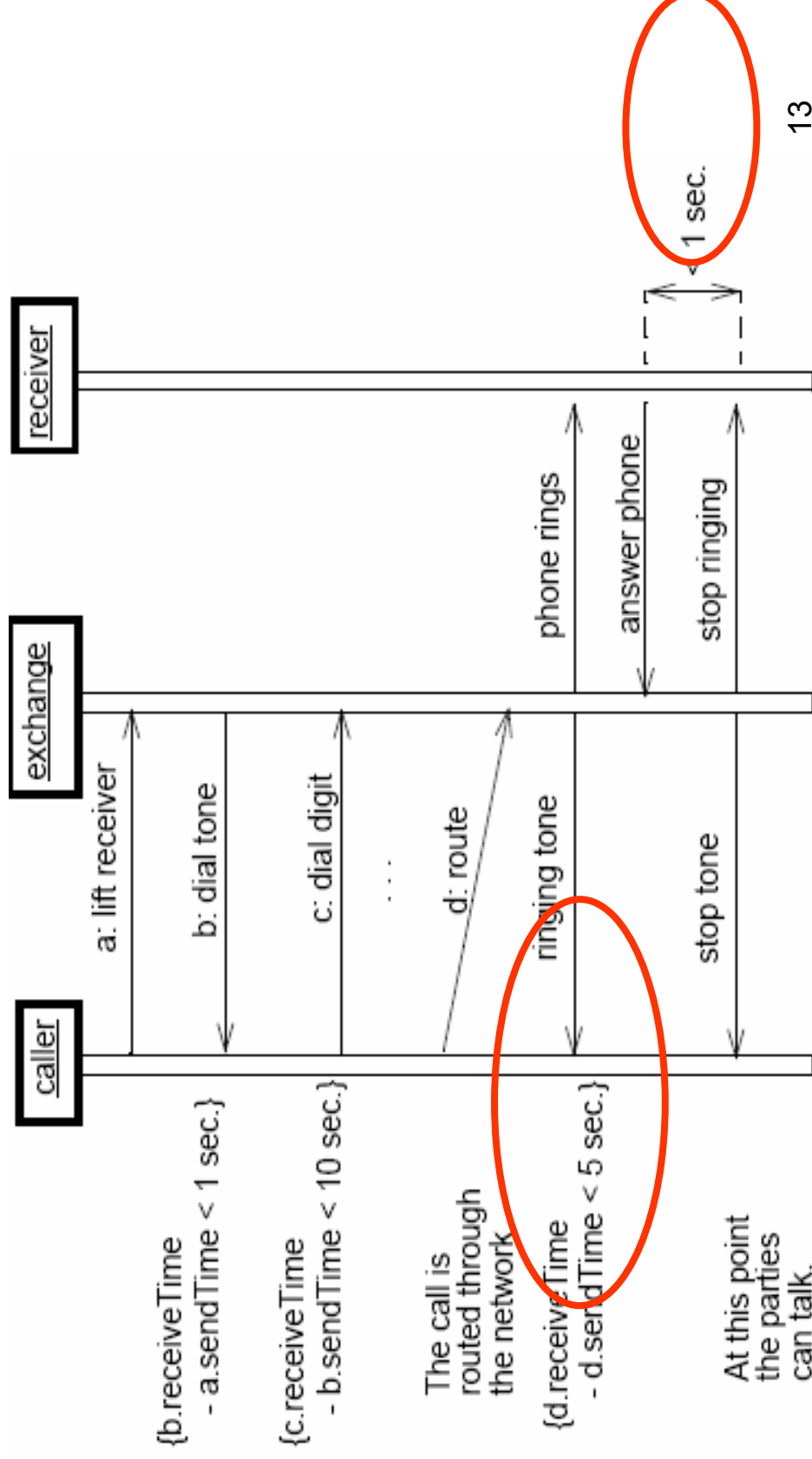
- 2 forms of sd:
 - Instance** sd: describes a specific scenario in detail; no conditions, branches or loops.
 - Generic** sd: a use case description with alternative courses.



Timing constraints

- Useful in real-time applications
- useful to specify race condition behaviour
- Two ways to specify (in 1.x)

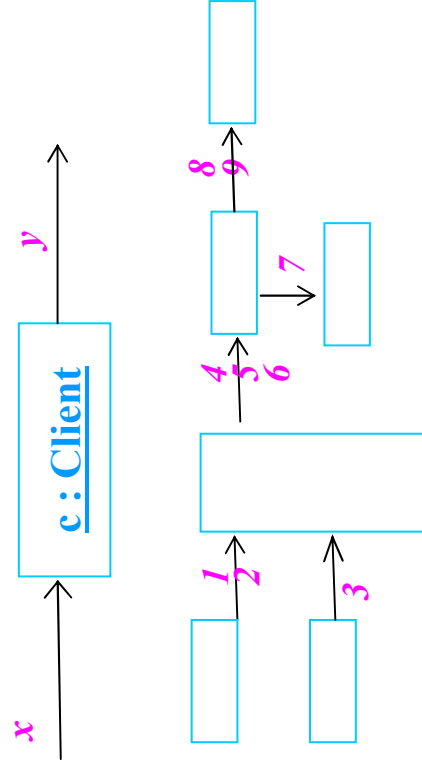
any example?



Interactions - Procedural Sequencing vs. Flat Sequencing

Flat Sequencing

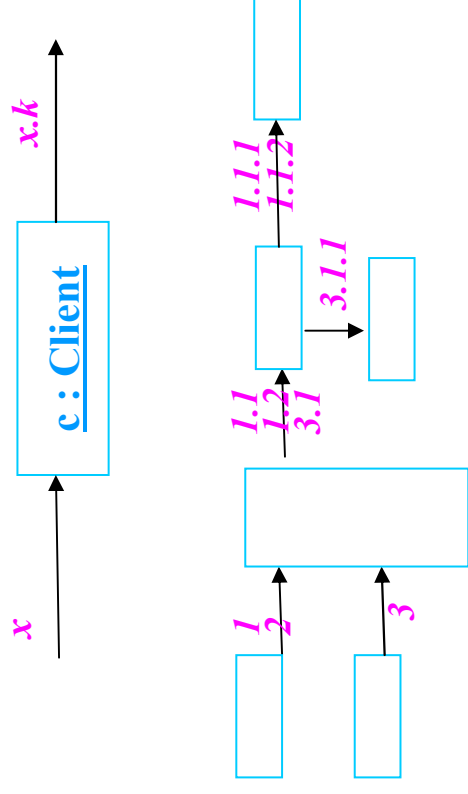
- Infrequent: *Not recommended for most situations.*
- Each message is numbered sequentially in order of timing.
- Rendered with **stick** arrowhead.



Procedural Sequencing

(Dewey decimal system)

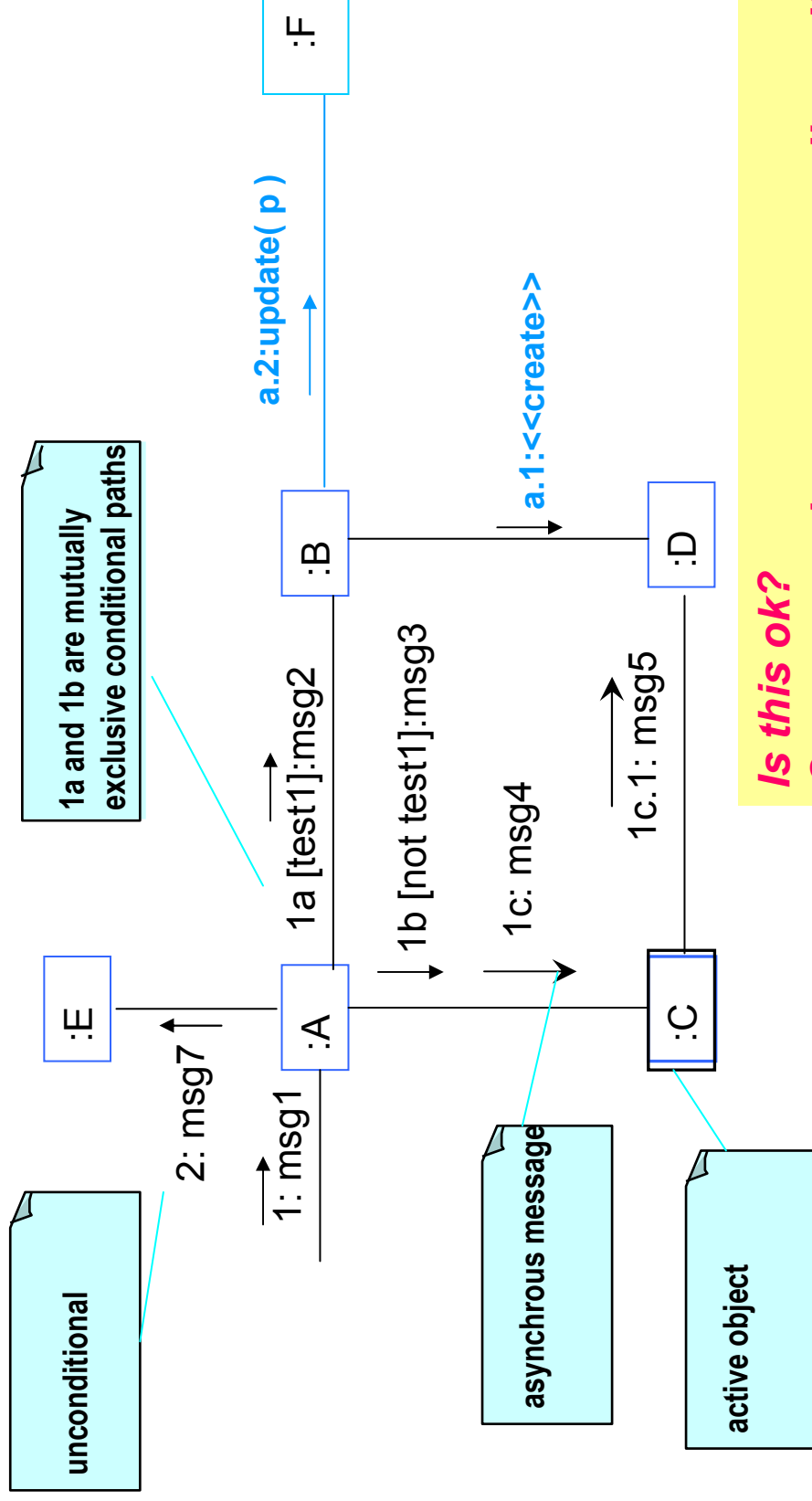
- Most common.
- Each message within the same operation is numbered sequentially.
- **Nested** messages are prefixed with the sequence number of the invoking operation.
- Rendered with filled **solid** arrow.



CAN'T TELL RELATIONSHIPS

Interactions – conditional paths, asynchronous message

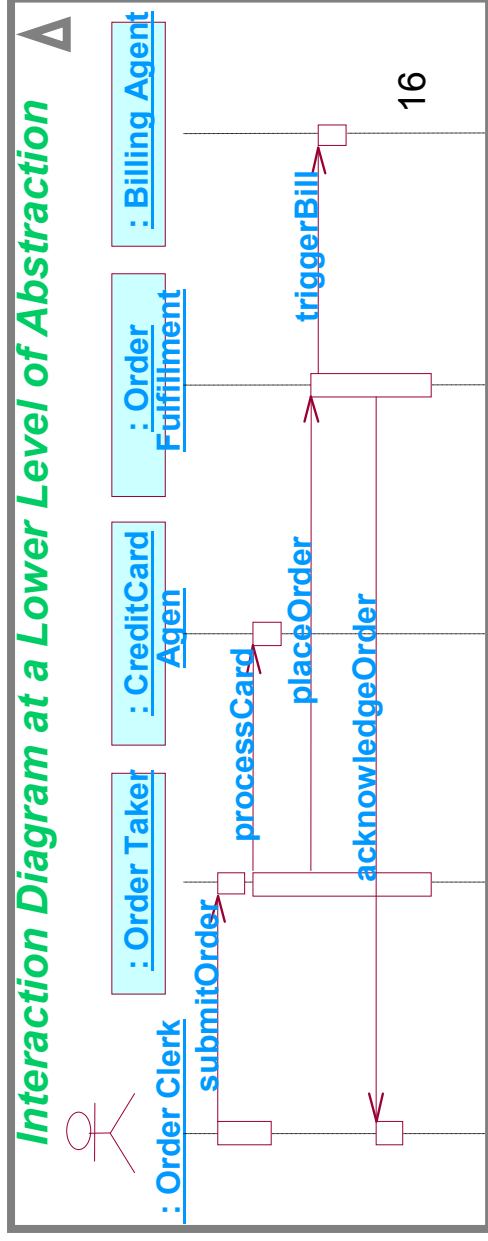
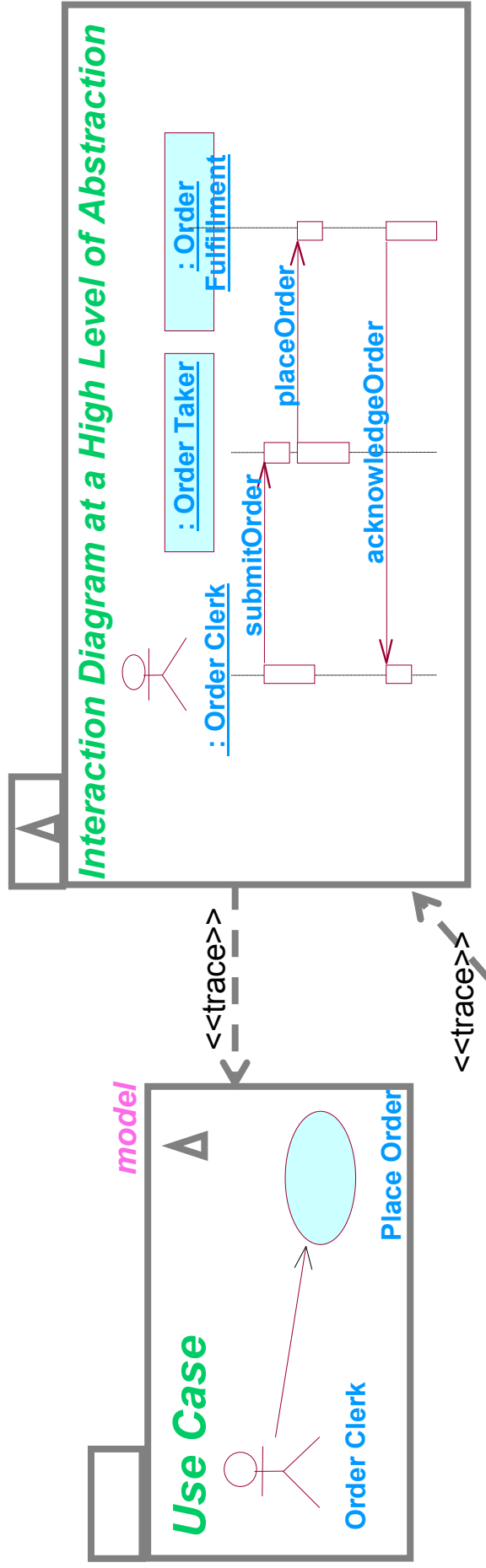
[Craig Larman] [<http://www.phptr.com/articles/article.asp?p=360441&seqNum=6&r1=1>]



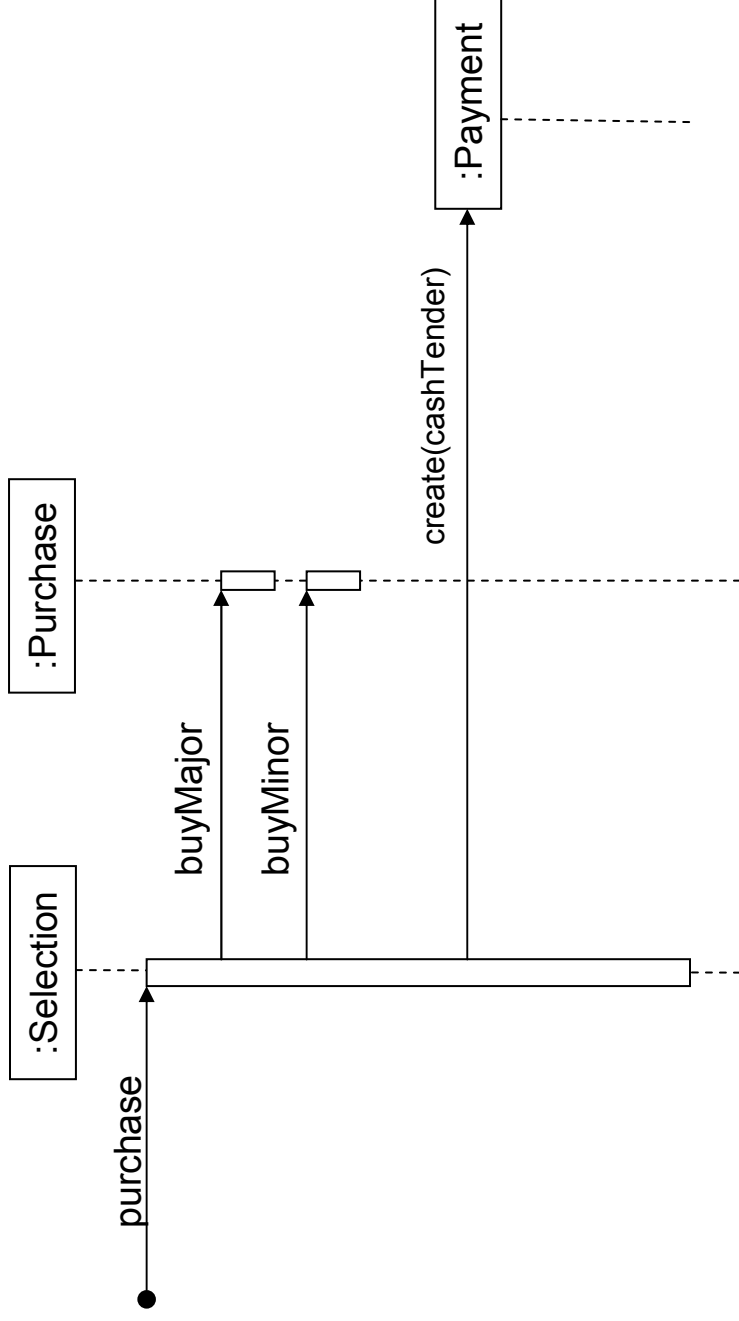
*Is this ok?
Can you produce a corresponding sd?
Is there a unique sequence of paths?*

Modeling Different Levels of Abstraction

- Establish trace dependencies between high and low levels of abstraction
- Loosely couple *different levels of abstraction*
 - Use Cases trace to **Collaborations** in the **Design** Model, to a society of **classes**
 - **Components** trace to the elements in the design model, then to **Nodes**



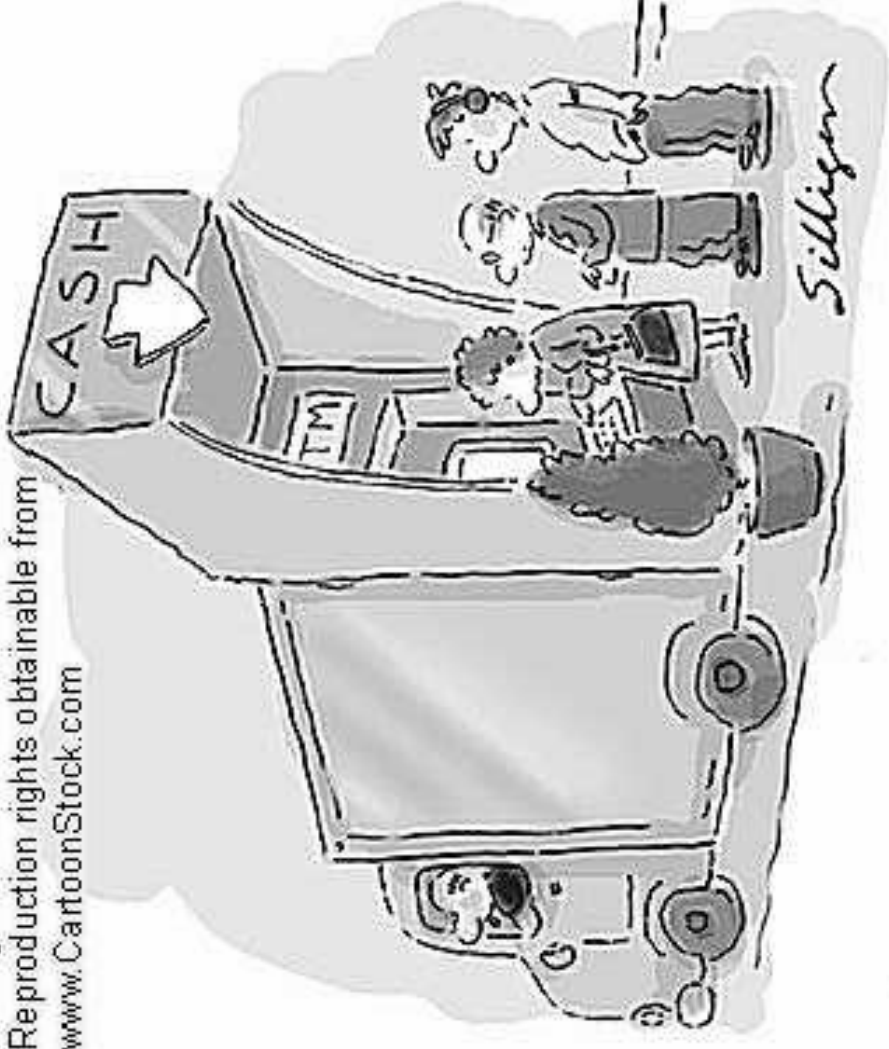
Sequence Diagrams & Some Programming



public Class Selection

```
{ private Purchase myPurchase = new Purchase();
  private Payment myPayment;
  public void purchase()
  { myPurchase.buyMajor();
    myPurchase.buyMinor();
    myPayment = new Payment( cashTender );
    //..
  }
  //..
}
```

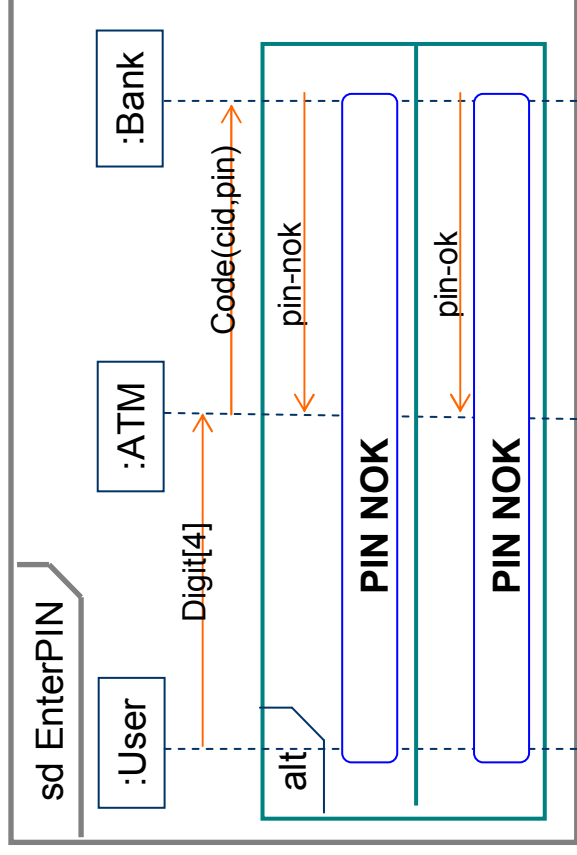
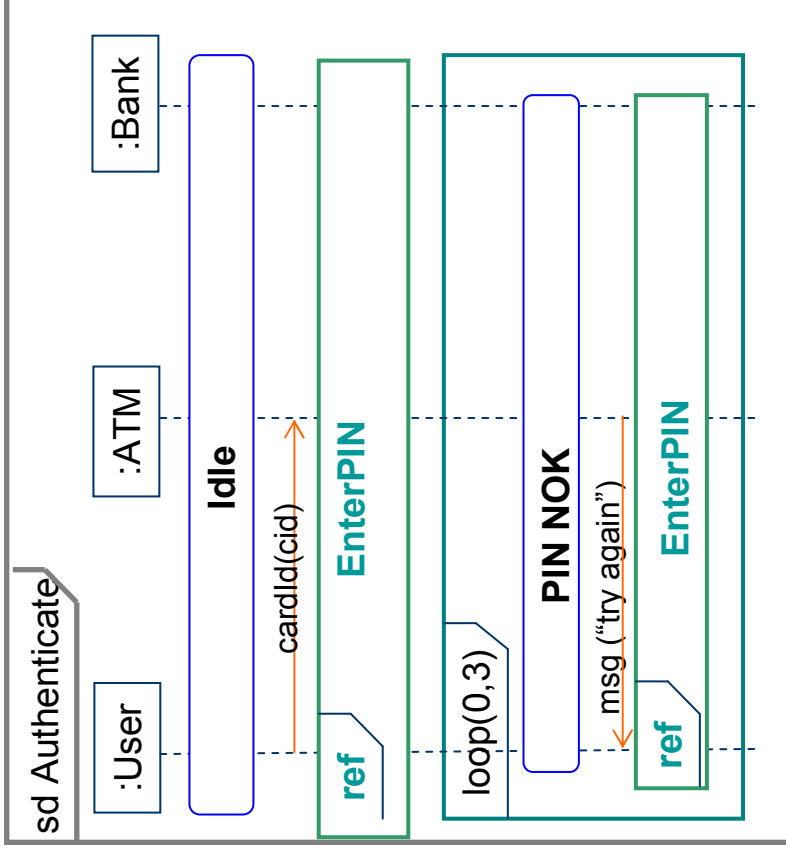
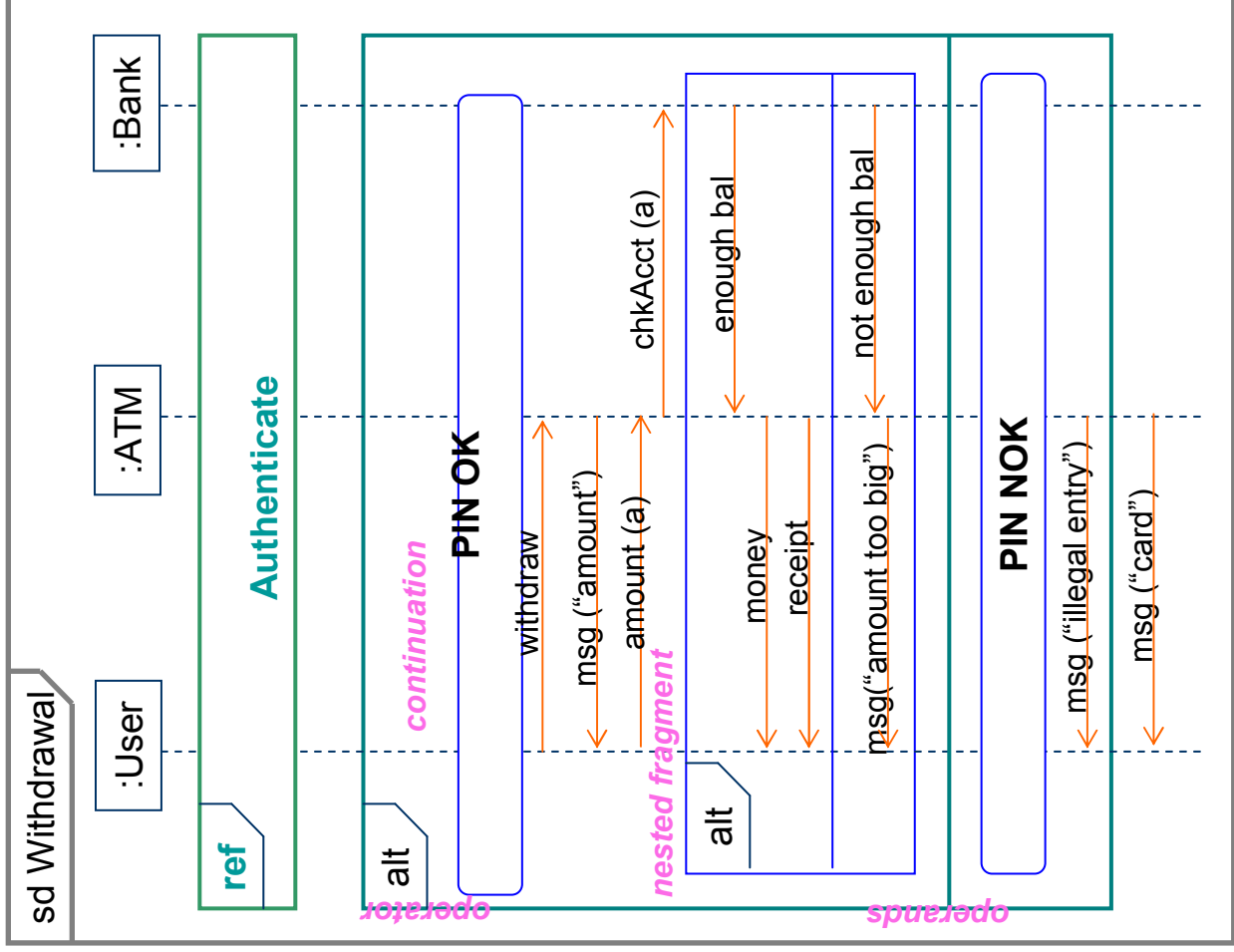
© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com



Internet Commen trick cash from customers.

- ❖ **Why do people call it an ATM machine, but they know it's really saying Automated Teller Machine Machine?**
- **Why do people say PIN number when that truly means Personal Identification Number Number?**

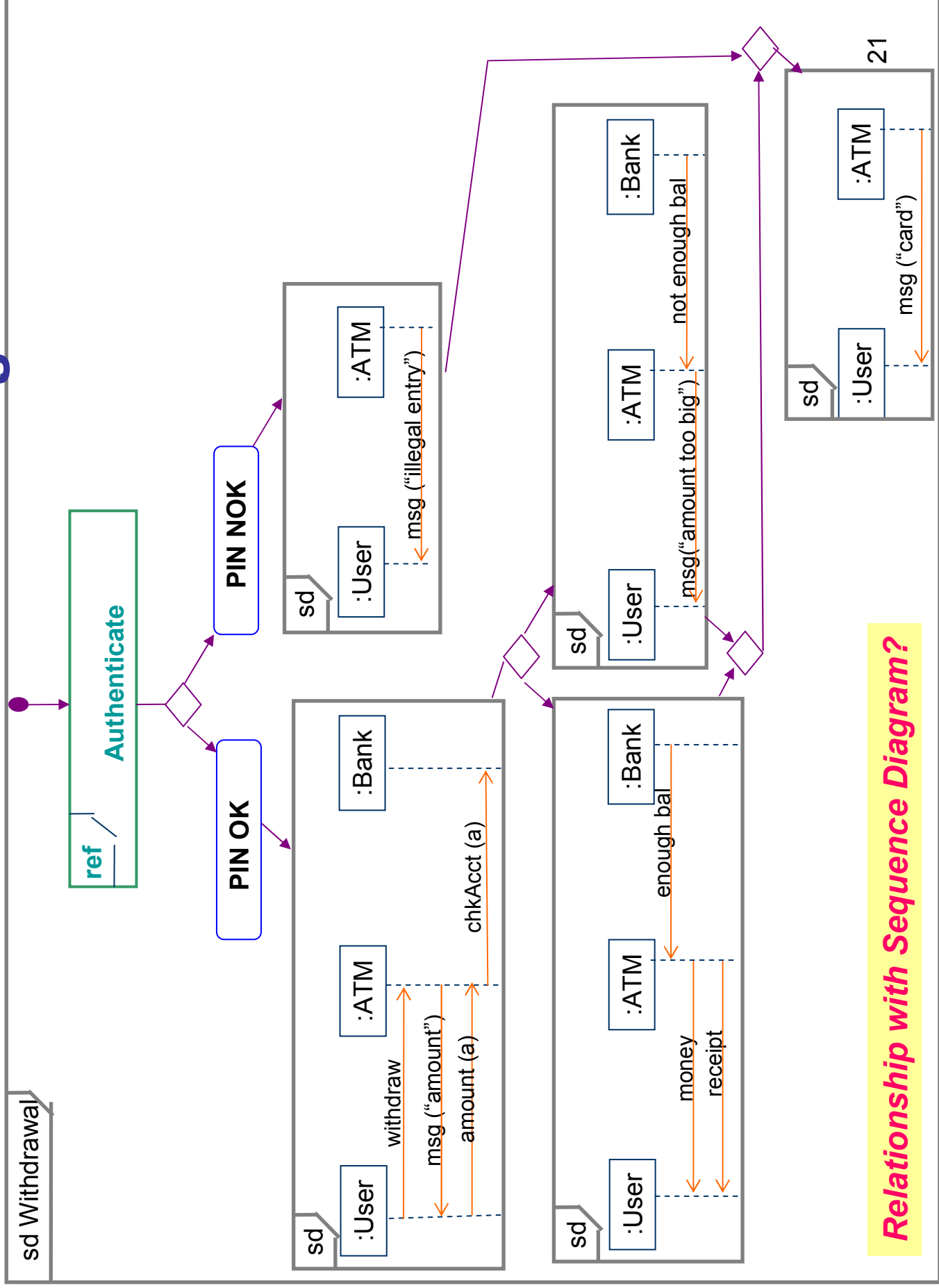
Frames: References



Interaction Overview Diagram

- variants on [UML activity diagrams](#) which overview control flow.
- nodes are frames instead of activities.
- two types of frame:
 - > interaction frames - any type of UML interaction diagram ([sd](#), [cd](#), [td](#), [iod](#)) or
 - > interaction occurrence frames (**ref**) which indicate an activity or operation to invoke.

Interaction Overview Diagram



Relationship with Sequence Diagram?

Frames & Interaction Fragment Operators

Frame: as the graphical boundary, and a labeling mechanism

Frame name: “frame type name[(param type: param name)] [: return type: return name]”

Flow of Control

- sd: named sequence diagram ref: reference to “interaction fragment”

Naming

- loop: repeat interaction fragment
- opt: optional “exemplar” (cf. *break*)
- alt: selection
- par: concurrent (parallel) regions (e.g., mo.cookFood -> par(nukeFood, rotateFood))

[guard condition] can appear as the first item underneath

Ordering

- seq: partial ordering (default) (aka “weak”)
- strict: strict ordering
- criticalRegion: identifies “atomic” fragments

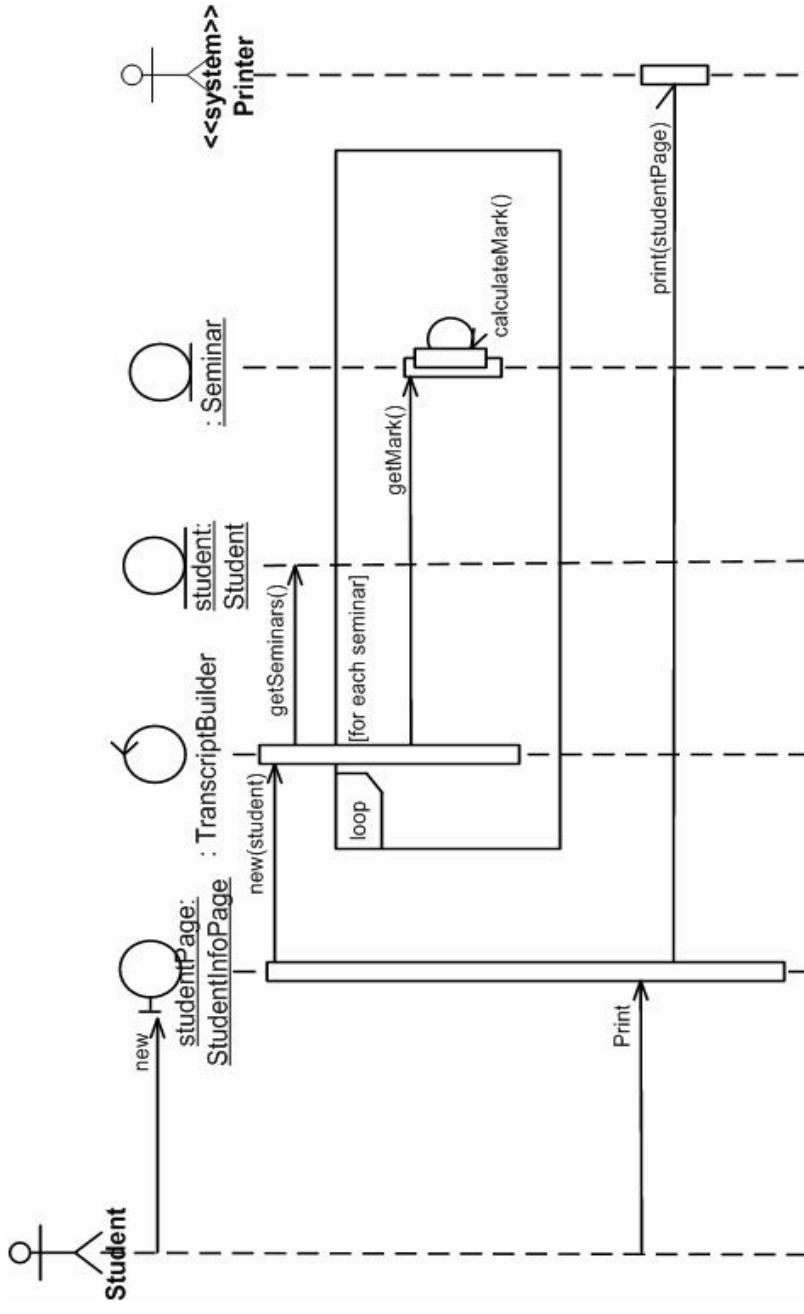
Causality

- assert: required (i.e. causal)
- neg: “can’t happen” or a negative specification
- Ignore/consider: messages outside/inside causal stream

What can be in the top boxes?

(<http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>)

Outputting transcripts



Boundary/interface elements: software elements such as screens, reports, HTML pages, or system interfaces that actors interact with.



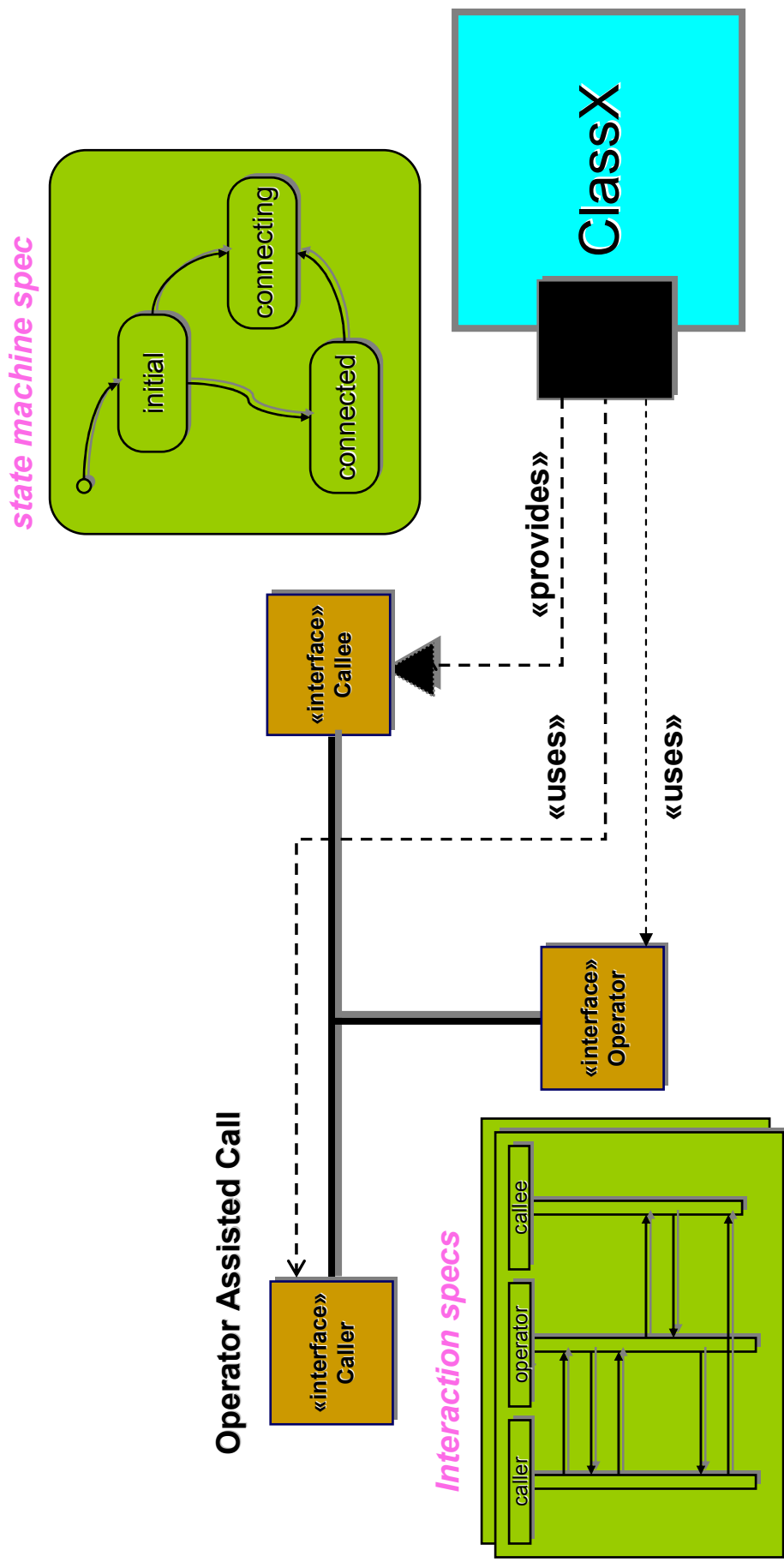
Control/process elements (controllers): These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions. Often implemented using objects, but simple ones using methods of an entity or boundary class.



Entity elements

Modeling Protocols - Associating Protocols with Ports

- By a set of **interconnected interfaces**, invoked according to a formal **behavioral** specification.

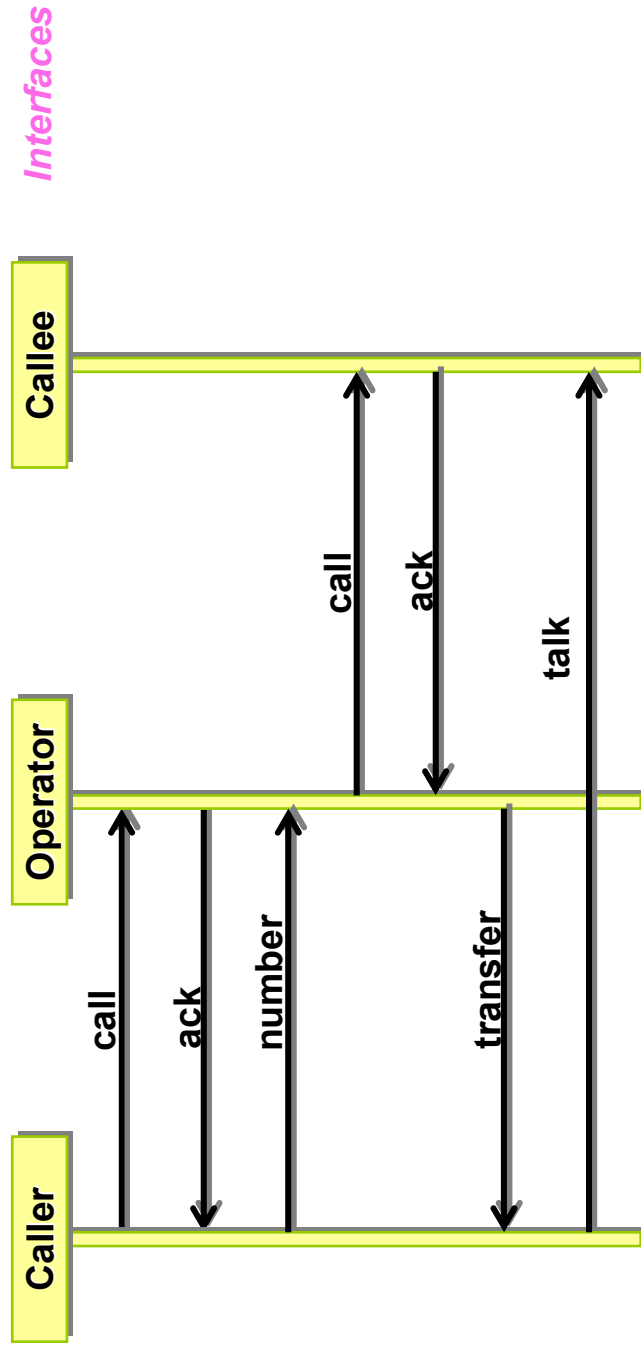


Can you depict this using balls & sockets?

Protocols: Reusable Interaction Sequences

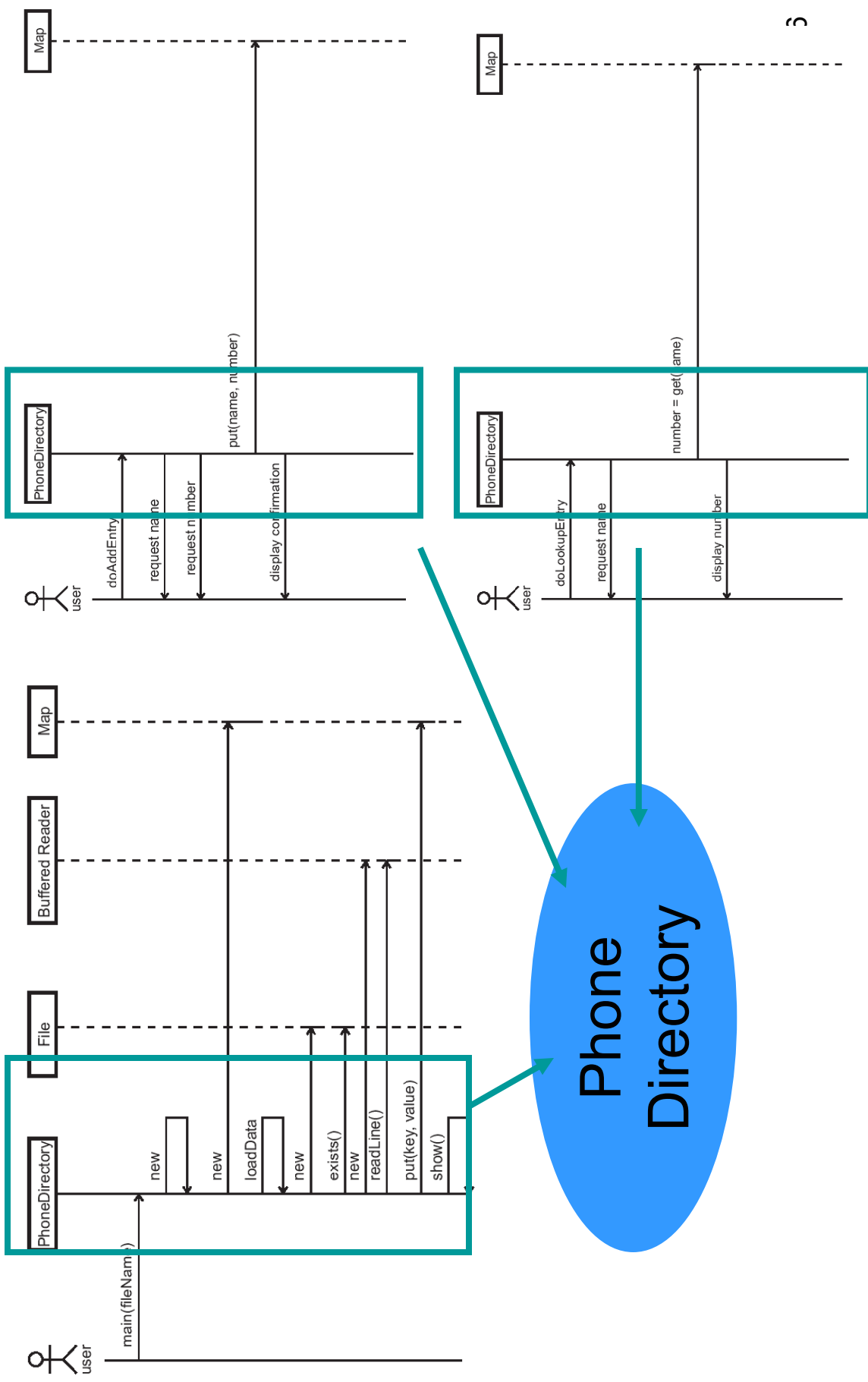
(<http://cot.uni-mb.si/ots2003/ppt/Selic-UML2.0-tutorial.030504.pdf>)

- ❑ Communication sequences that
- ❑ always follow a pre-defined dynamic order
- ❑ occur in different contexts with different specific participants (i.e., instances)



From Diagrams to Objects

Collect all messages to define object's methods and state transitions !



State Transition Diagrams

Behavioral Diagrams

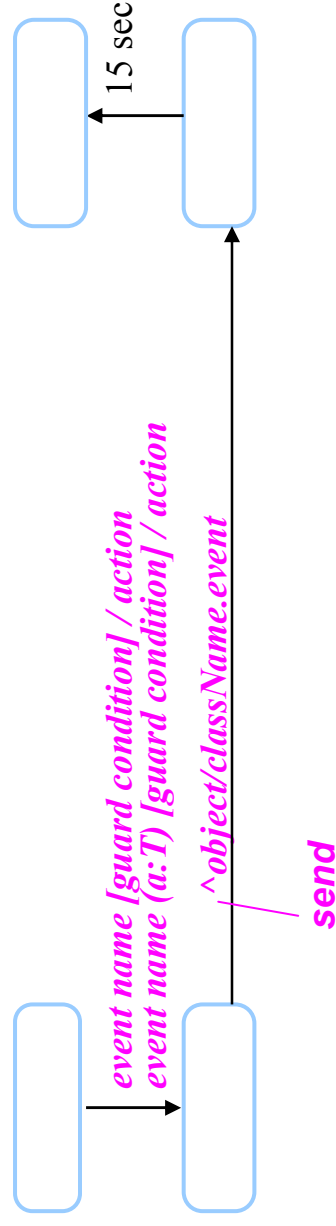
- Use case
- **Statechart**
- Activity

Interaction Diagrams

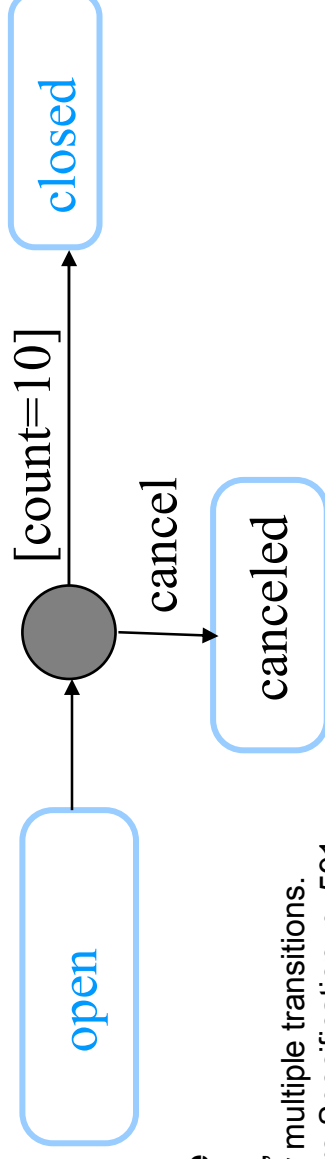
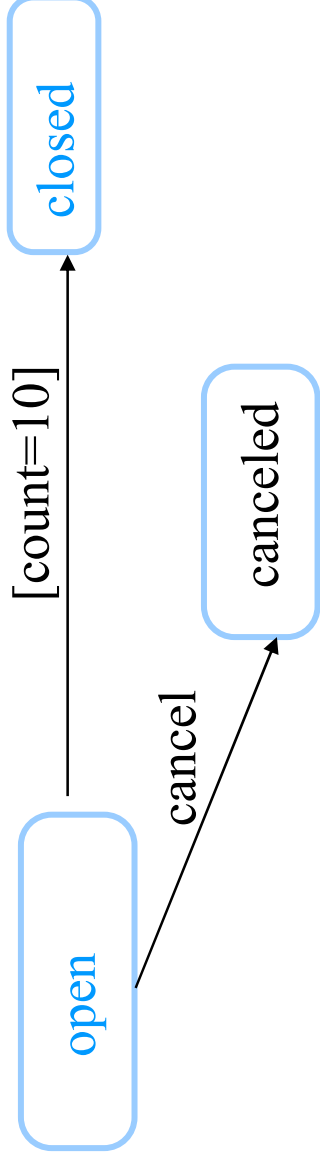
- Sequence;
Communication
- Interaction Overview
- Timing

State Transitions

- **State machine** - **event-ordered** behavior that specifies the sequences of **states** an object/instance (of class/interface/collaboration/.../system) goes through during its lifetime; **events** trigger **transitions** and cause **responses**.
(StateChart is one particular kind of state machine by David Harel)
- **State** - condition or situation during which an object/instance may perform some activity; The state of an object is characterized by the value of one or more of its attributes.
- **Activity** - ongoing **non-atomic** execution within a state machine.
- **Action** - executable **atomic** computation that results in a change in state of the model or the return of a value.

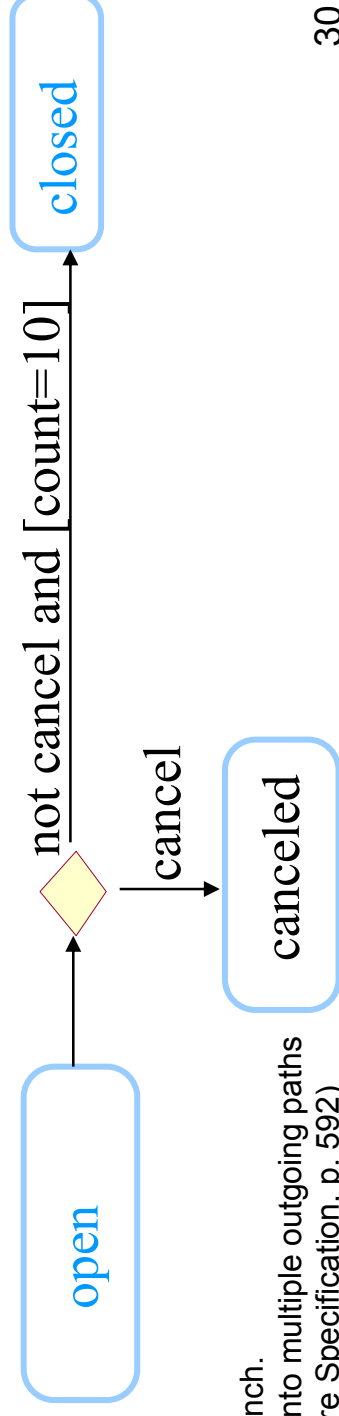


State Transitions – notational variation



Junction pseudo state

- semantic-free vertices,
- used to chain together multiple transitions.
- UML 2.0 Superstructure Specification, p. 591

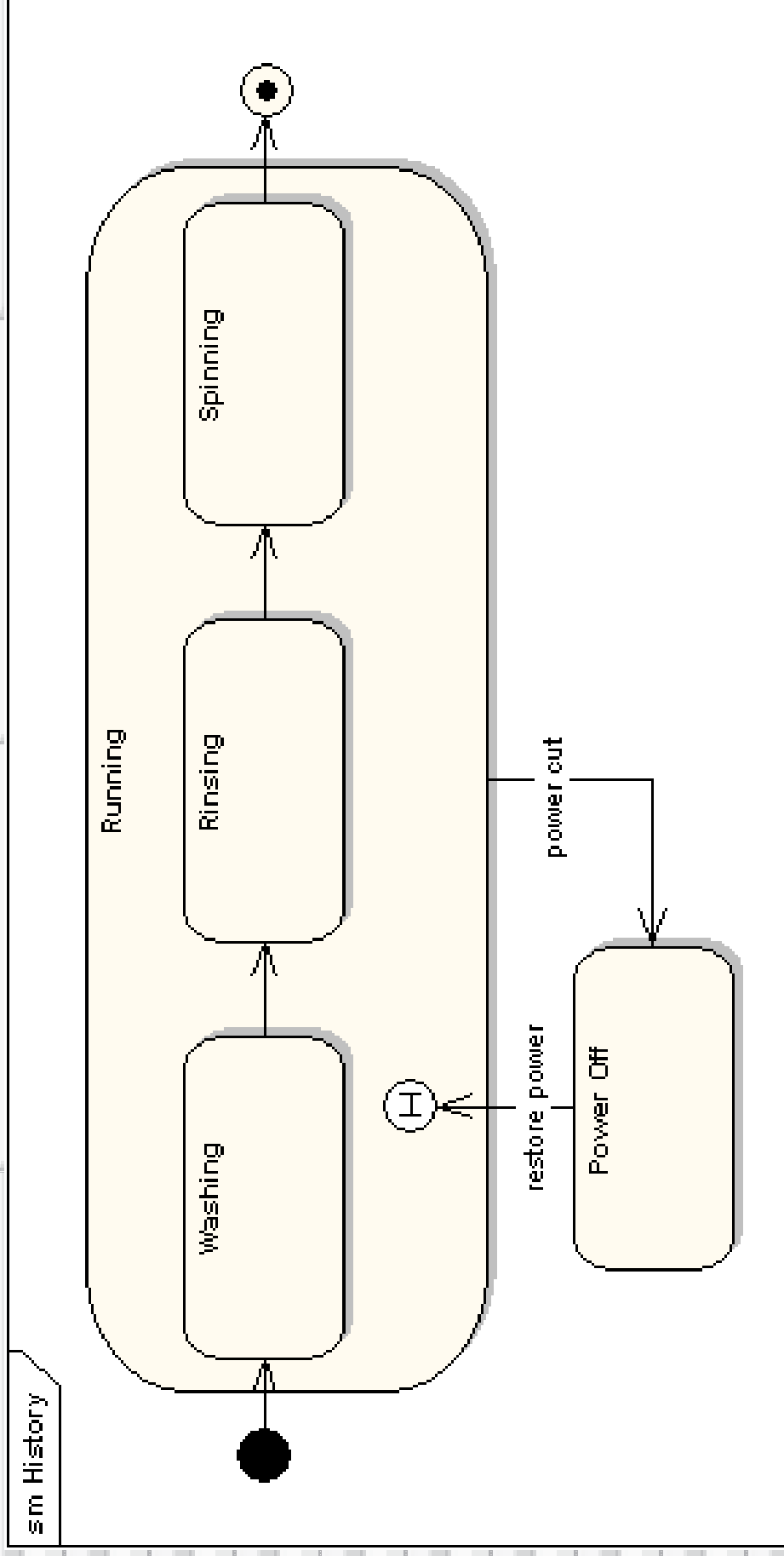


Choice pseudo state

- dynamic conditional branch.
- splitting of transitions into multiple outgoing paths
- UML 2.0 Superstructure Specification, p. 592)

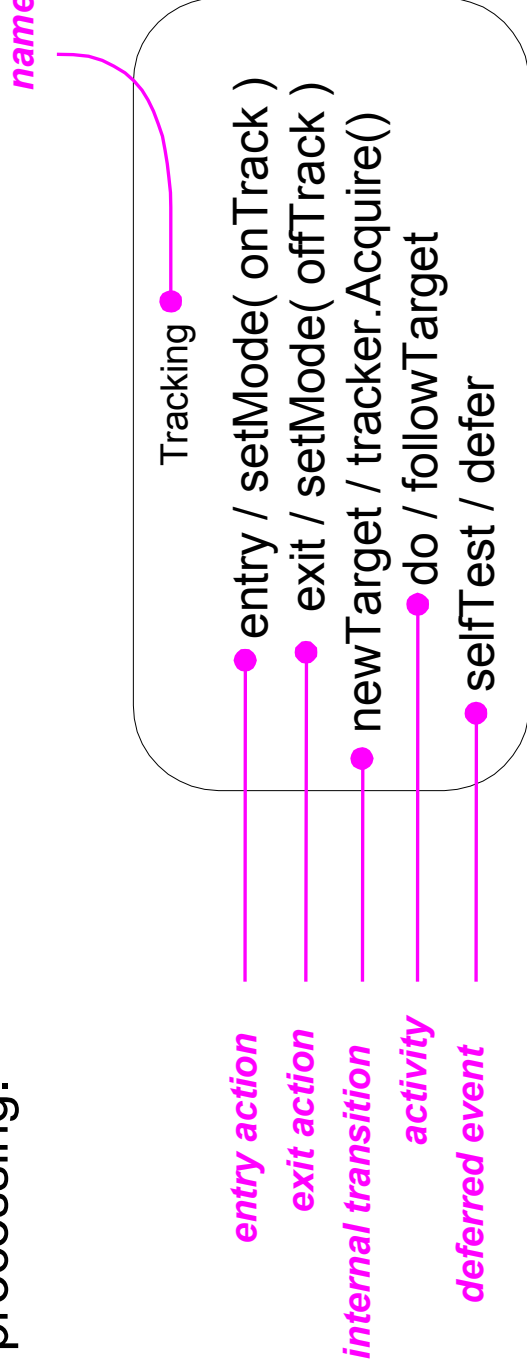
State Transitions – History States

-used to remember the previous state of a state machine when it was interrupted.



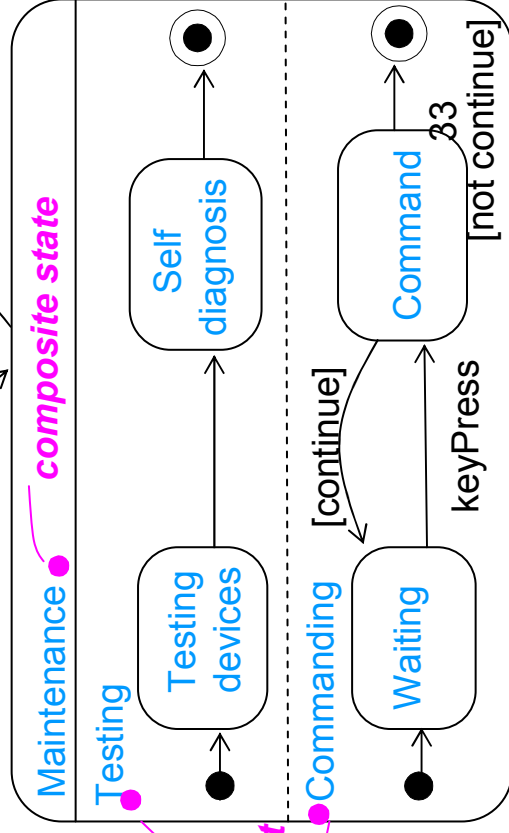
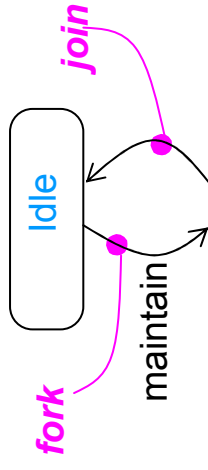
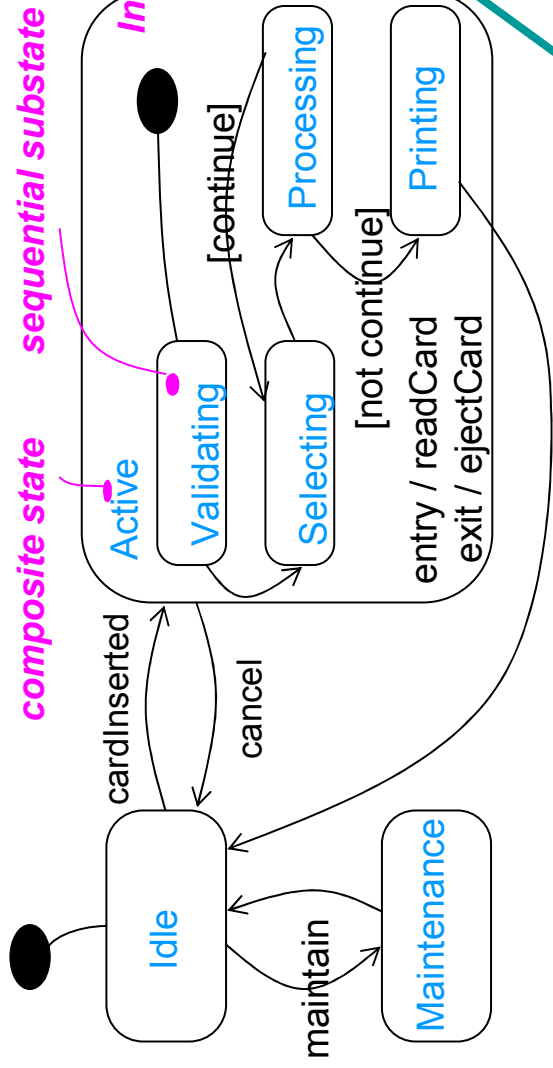
Advanced States

- ❑ **Entry & exit actions** - actions that always occur upon entry into or exit away from a state regardless of transition.
- ❑ **Internal Transitions** - triggered by events but don't change state.
- ❑ **Activities** - ongoing behavior which continues until interrupted.
- ❑ **Deferred events** - events ignored by the current state, but postponed for later processing.



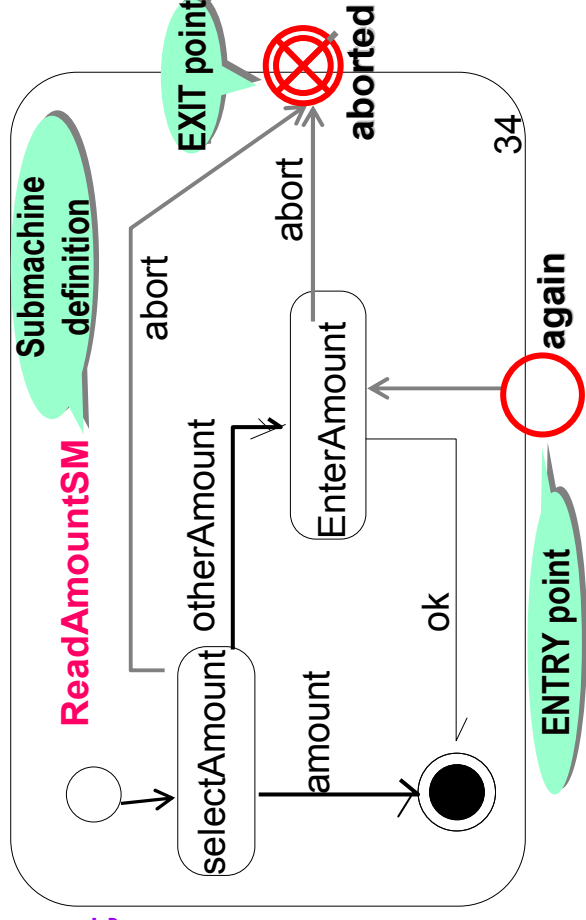
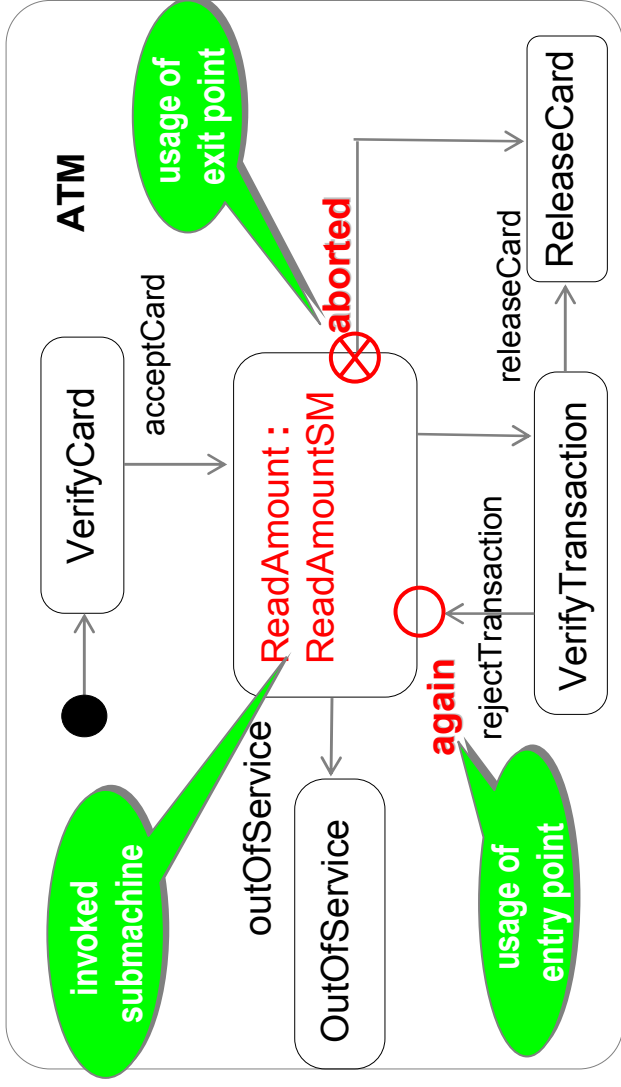
Substates

- Substate -- state nested inside of another state.
- Sequential substates (then a nonorthogonal state)
- Concurrent substates (then an orthogonal state)



Modular Submachines

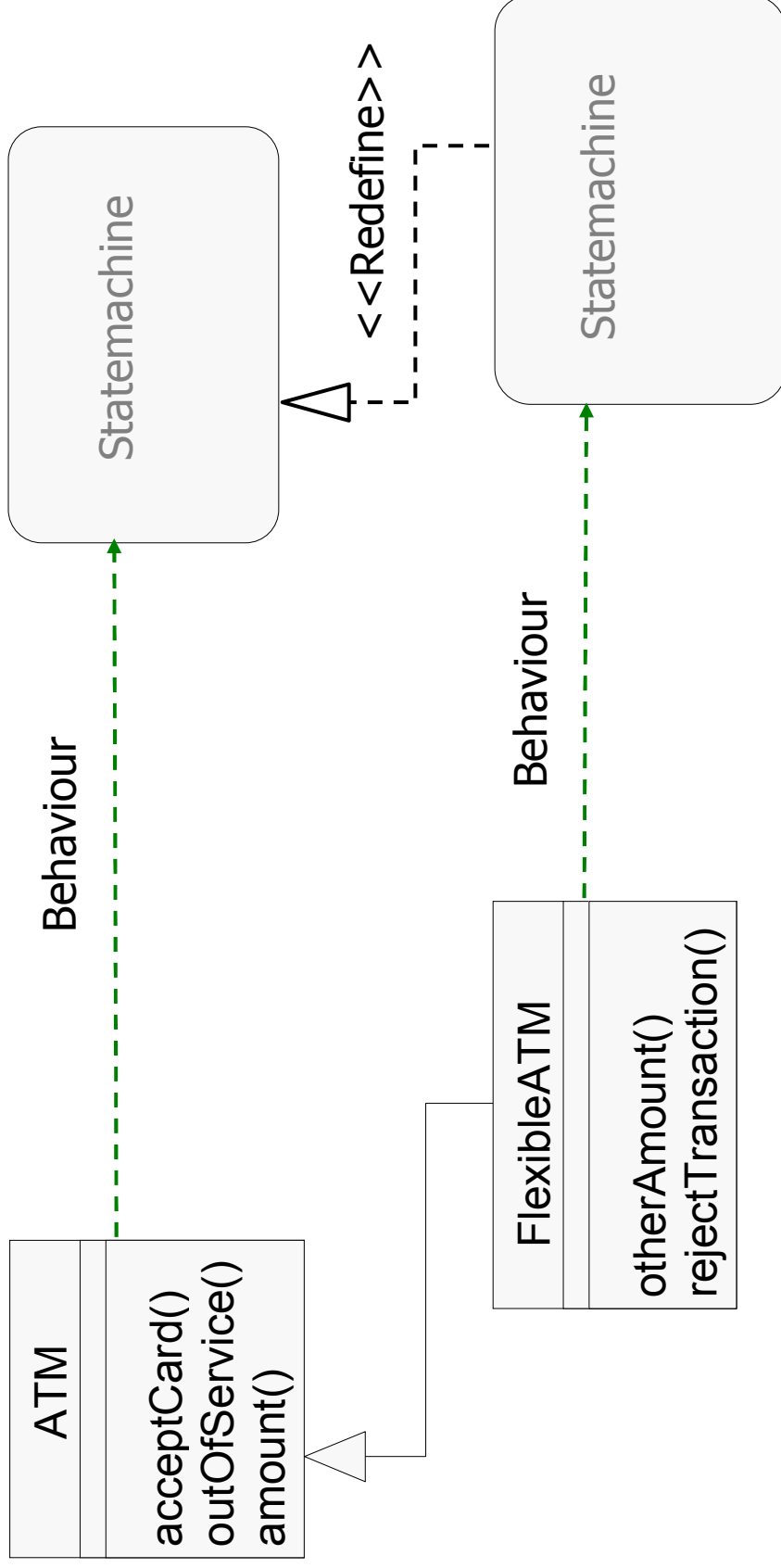
<http://www.xpdian.com/UML2.0changes.html>



- A composite state may have several entry and exit points;
- how does entry point differ from H/H*?
- promotes reuse (next slide)

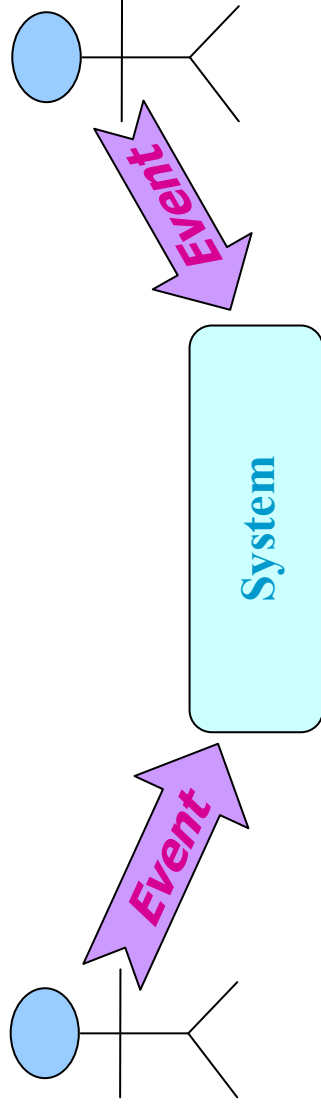
Specialization

- Redefinition as part of standard class specialization

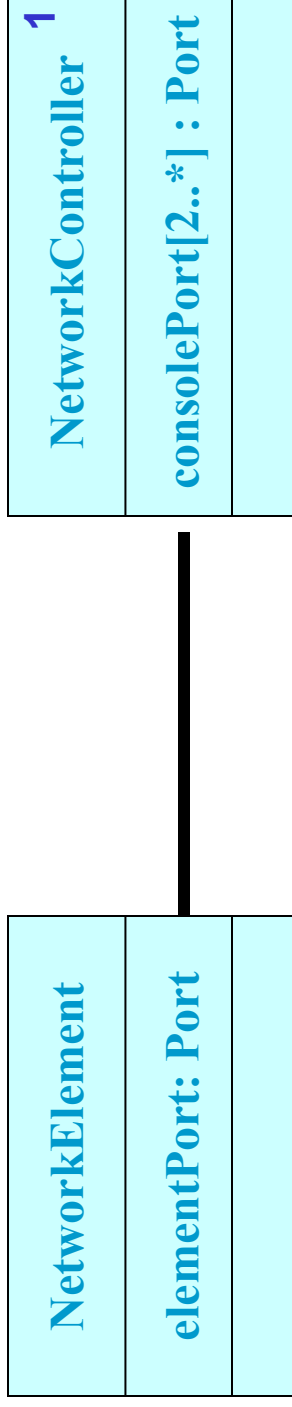


Events – External vs. Internal Events

- Events can be categorized into external or internal events.
- **External** events are those that pass between the Actors and the system.

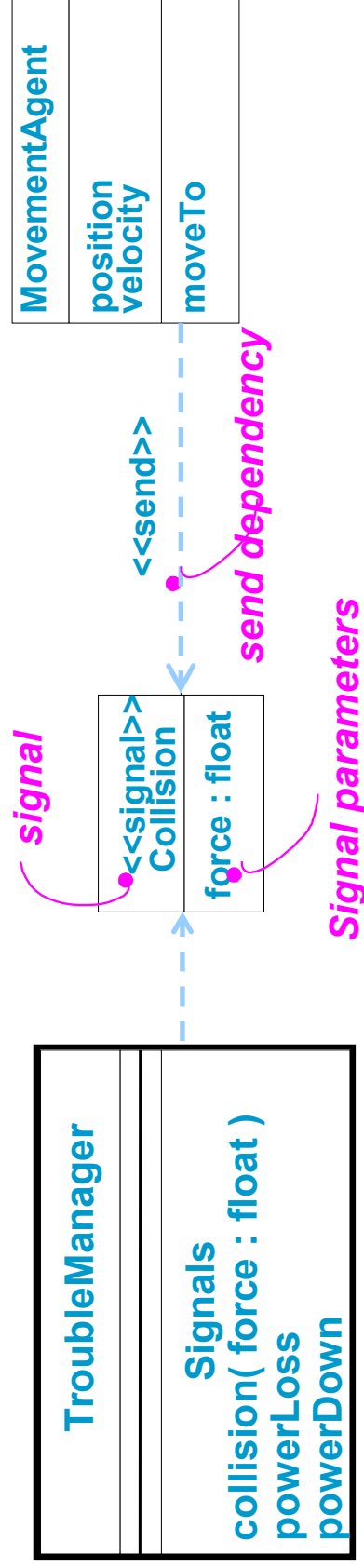


- **Internal** events pass between objects residing within the application system.



Events – 4 Kinds: *Signals*; Calls; Passing of Time; Change in State

- *Signal* - kind of event that represents the specification of an **asynchronous** stimulus communicated between instances.
- Modeled as a class
- Dispatched (thrown) by one object and continues flow of execution
- Received (caught) by another object at some future point in time.
- Can be sent as:
 - Action of a state transition
 - Message in an object interaction
 - Dependencies `<<send>>` show signals sent by a class

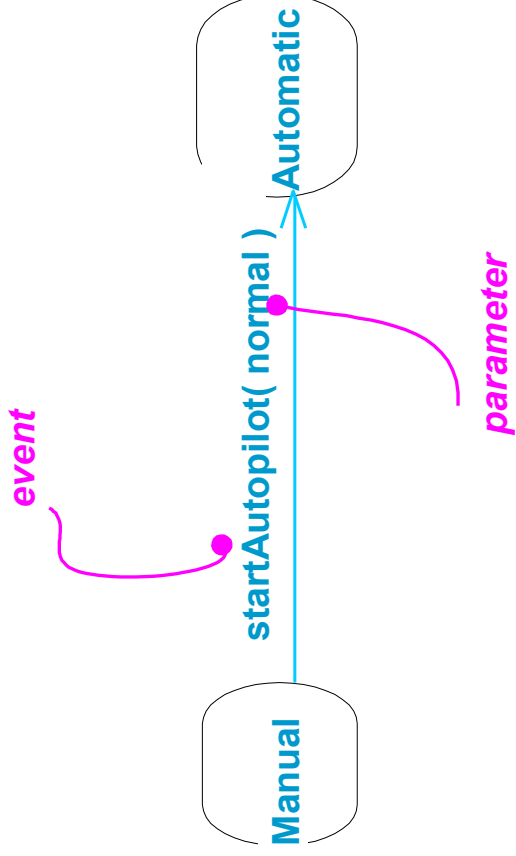


- Modeling Signal Receiver: as an active class; Consider 4th compartment for signals.

Events – 4 Kinds: Signals; Calls; Passing of Time; Change in State

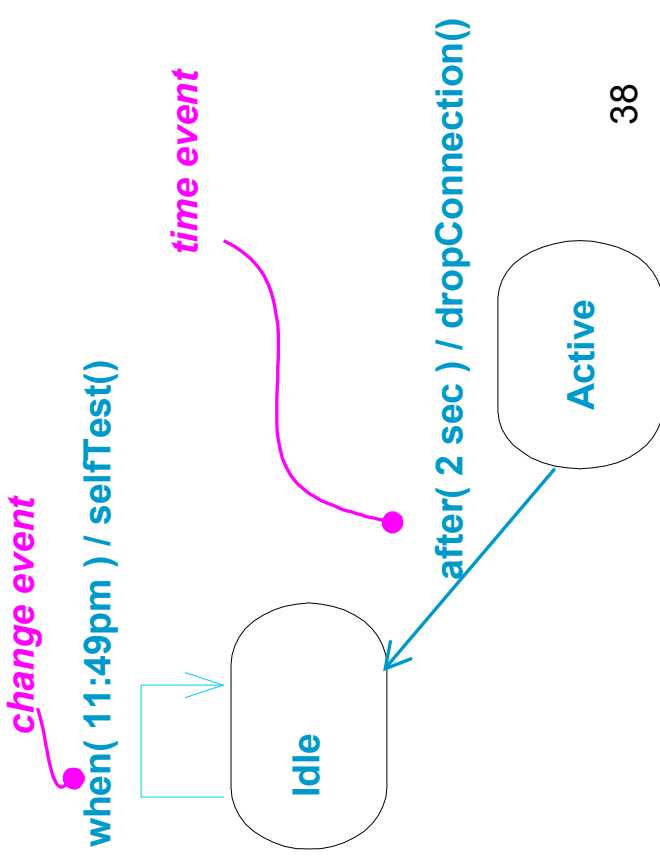
Call Events

- Represents the dispatch of an operation
- **Synchronous**



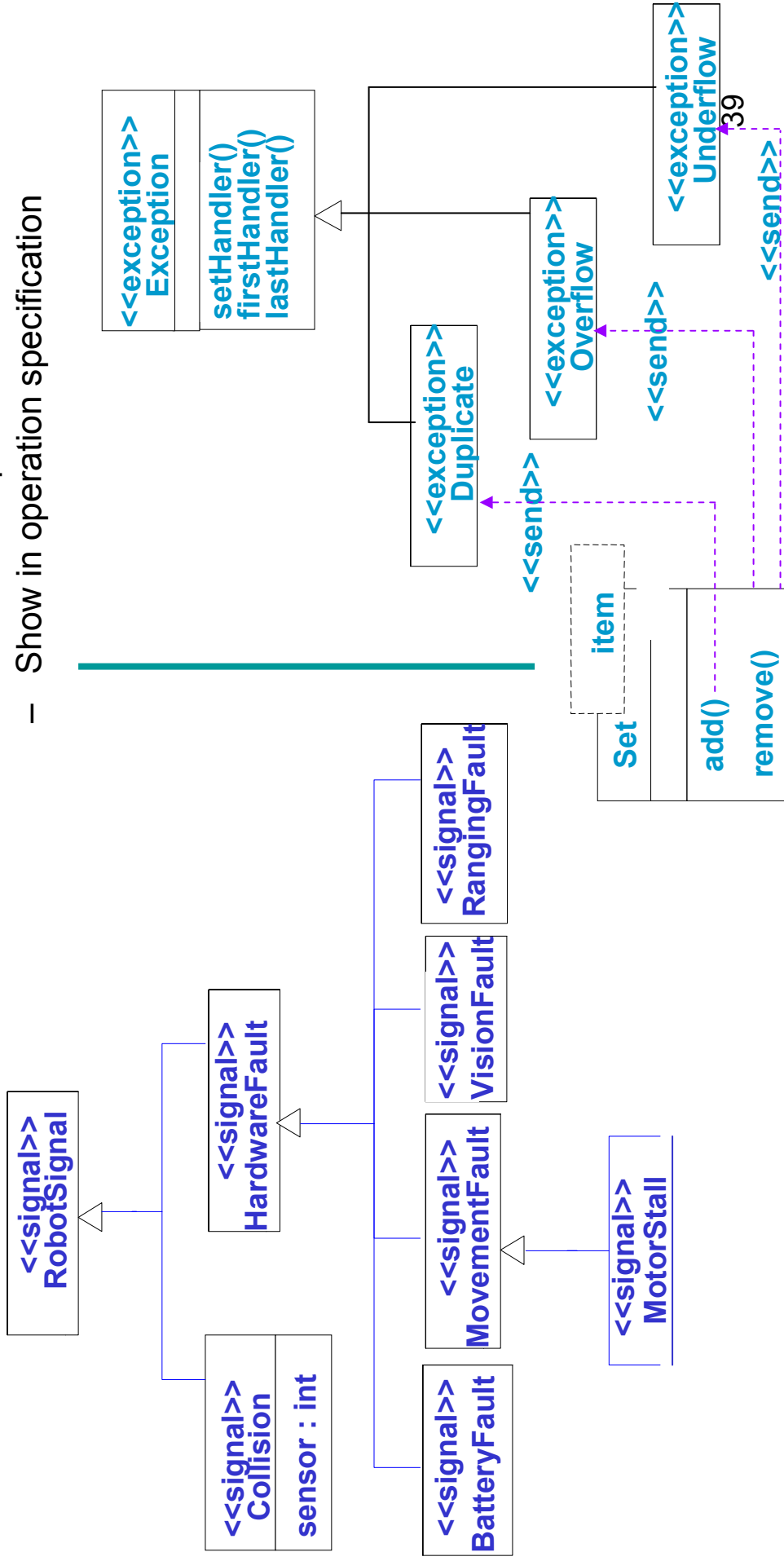
Time & Change Event

- Time event - represents the passage of time:
after(periodOfTime)
- Change event - represents a change in state or the satisfaction of some condition:
when(booleanExpr)



Modeling Family of Signals and Exceptions

- ❑ Signal events are typically hierarchical.
 - ❑ Look for common generalizations.
 - ❑ Look for polymorphic opportunities.
- ❑ Consider the exceptional conditions of each class
 - ❑ Arrange exceptions in generalization hierarchy.
 - ❑ Specify the exceptions that each operation may
 - Use send dependencies
 - Show in operation specification



Activity Diagrams

Behavioral Diagrams

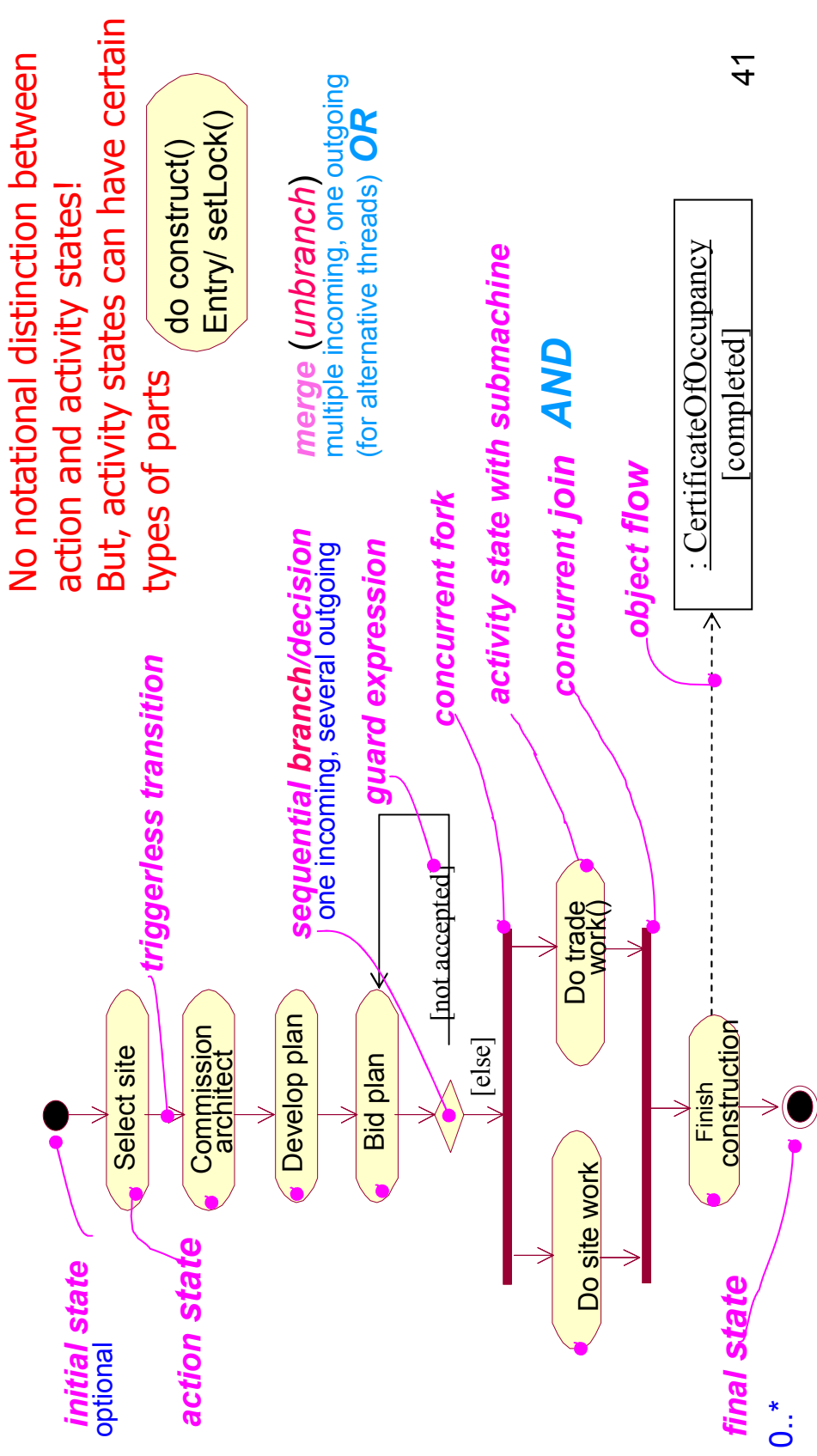
- Use case
- Statechart
- **Activity**

Interaction Diagrams

- Sequence;
Communication
- Interaction Overview
- Timing

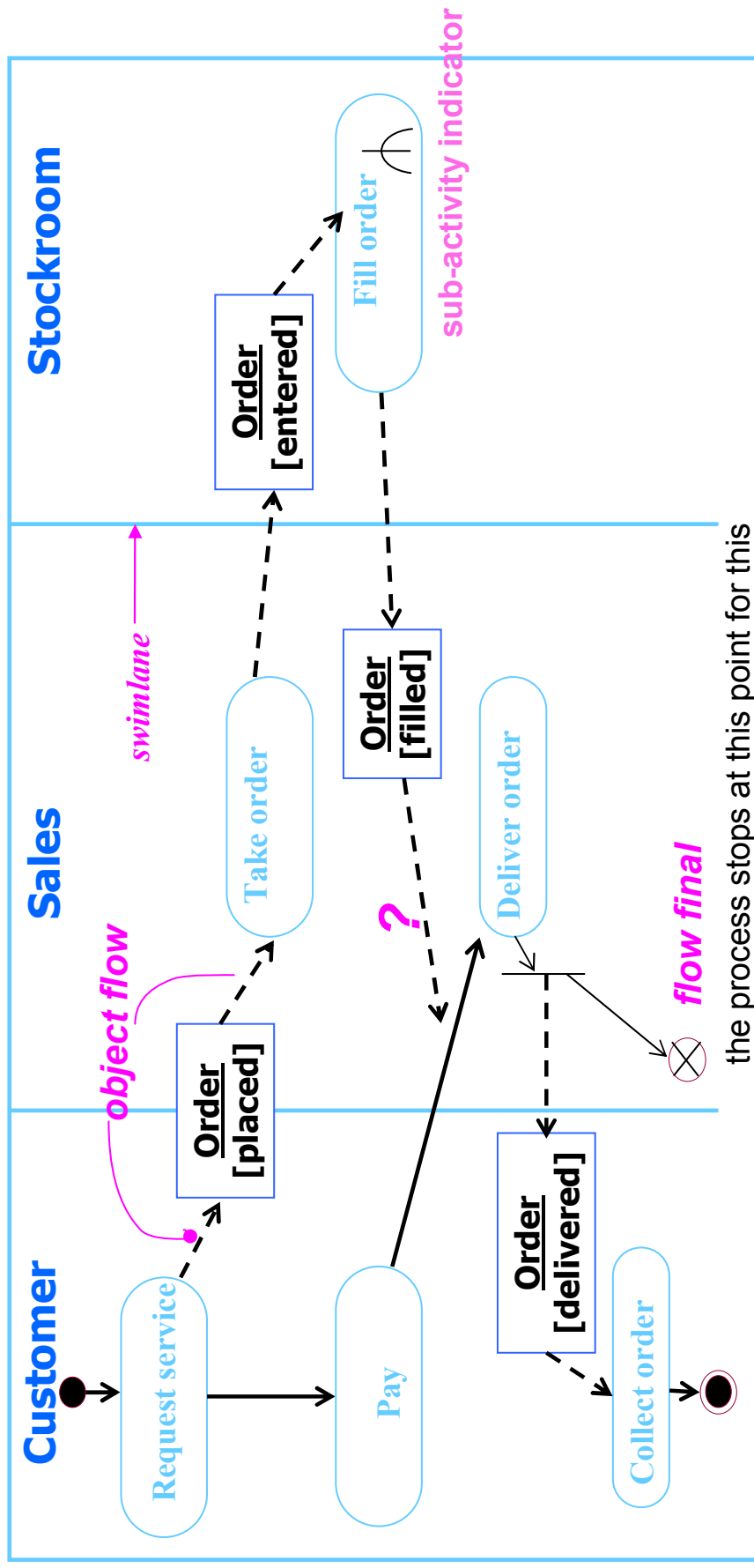
Activity Diagram Basics

- **Activity Diagram** – a special kind of Statechart diagram, but showing the flow from activity to activity (not from state to state).
- **Activity state** – *non-atomic* execution, ultimately result in some action; a composite made up of other activity/action states; can be represented by other activity diagrams
- **Action state** – *atomic* execution, results in a change in state of the system or the return of a value (i.e., calling another operation, sending a signal, creating or destroying an object, or some computation); non-decomposable



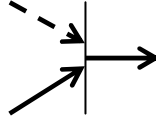
Swimlanes & Object Flow

- A **swimlane** is a kind of package.
- Every activity belongs to exactly one swimlane, but transitions may cross lanes.
- **Object flow** – objects connected using a dependency to the **activity or transition** that creates, destroys, or modifies them

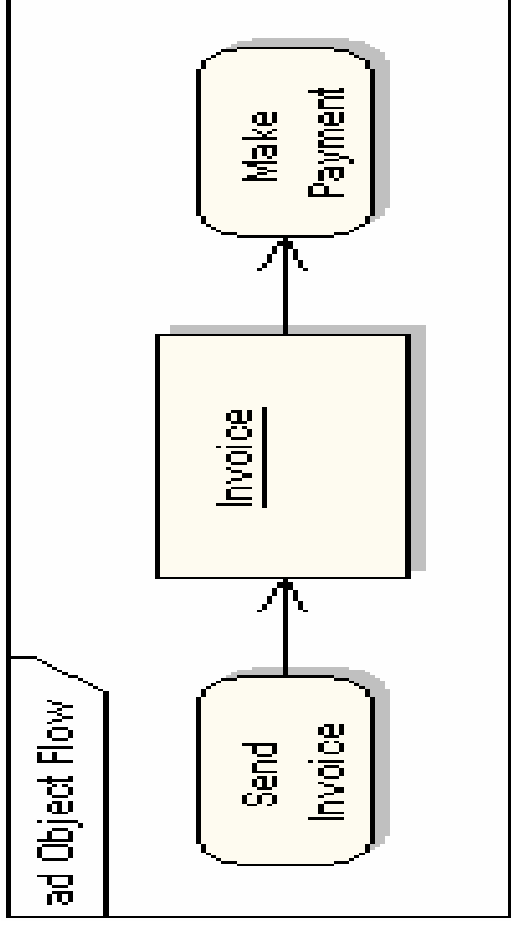


the process stops at this point for this part of the activity diagram

What if using pins?

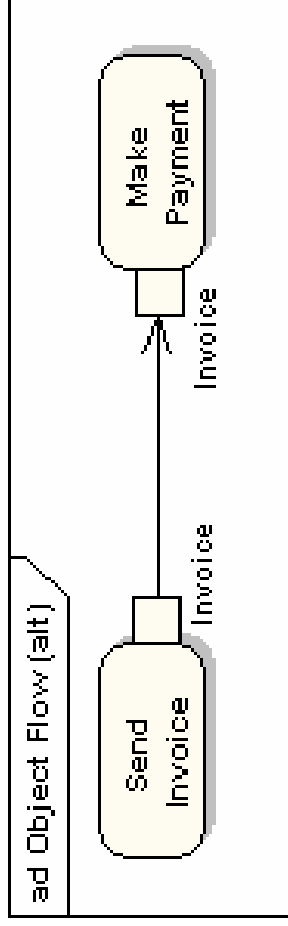


Object Flows and Pins

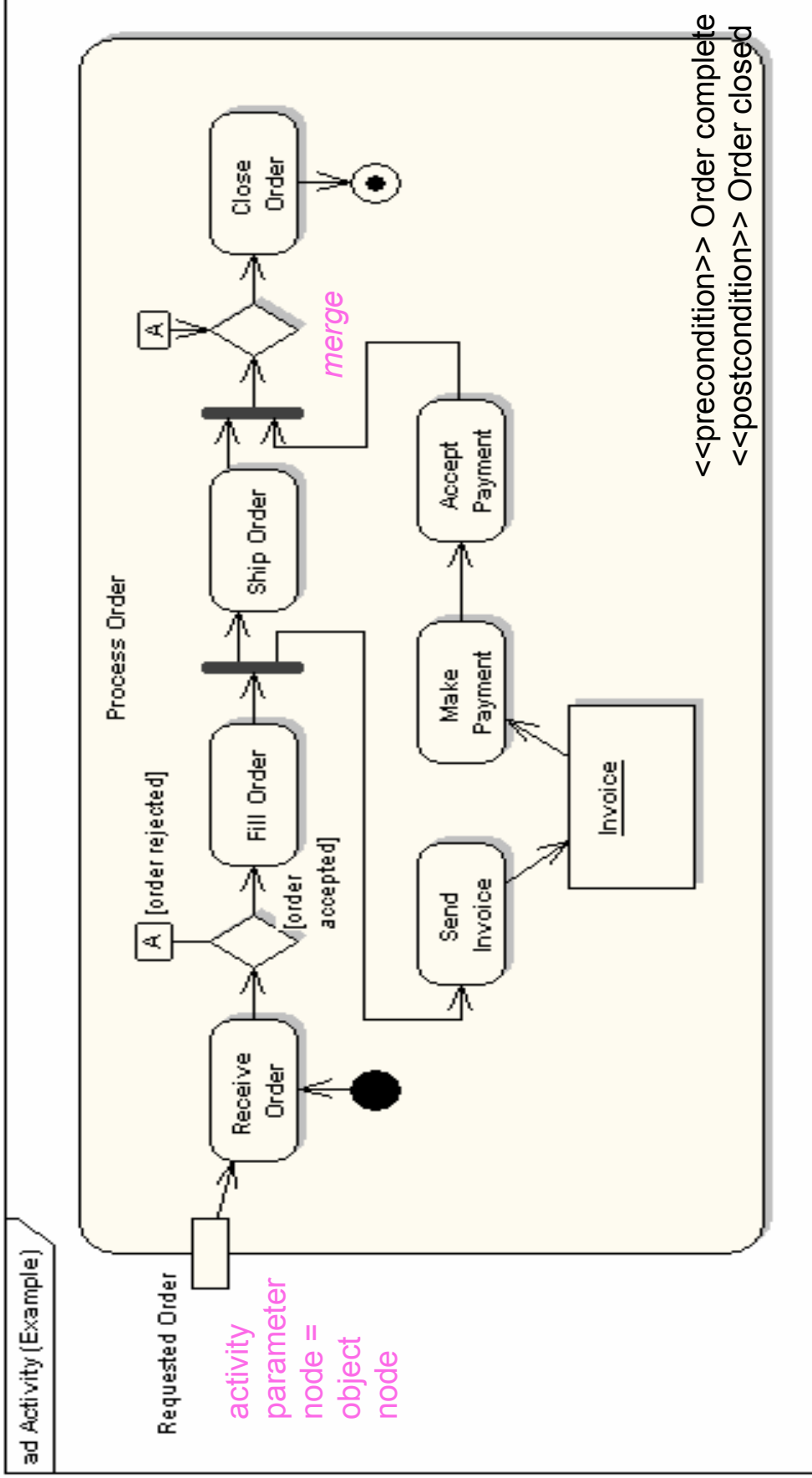


```
Invoice inv;  
inv = new Invoice;  
FillOrder(inv);
```

A shorthand notation: use input pins and output pins (parameters).

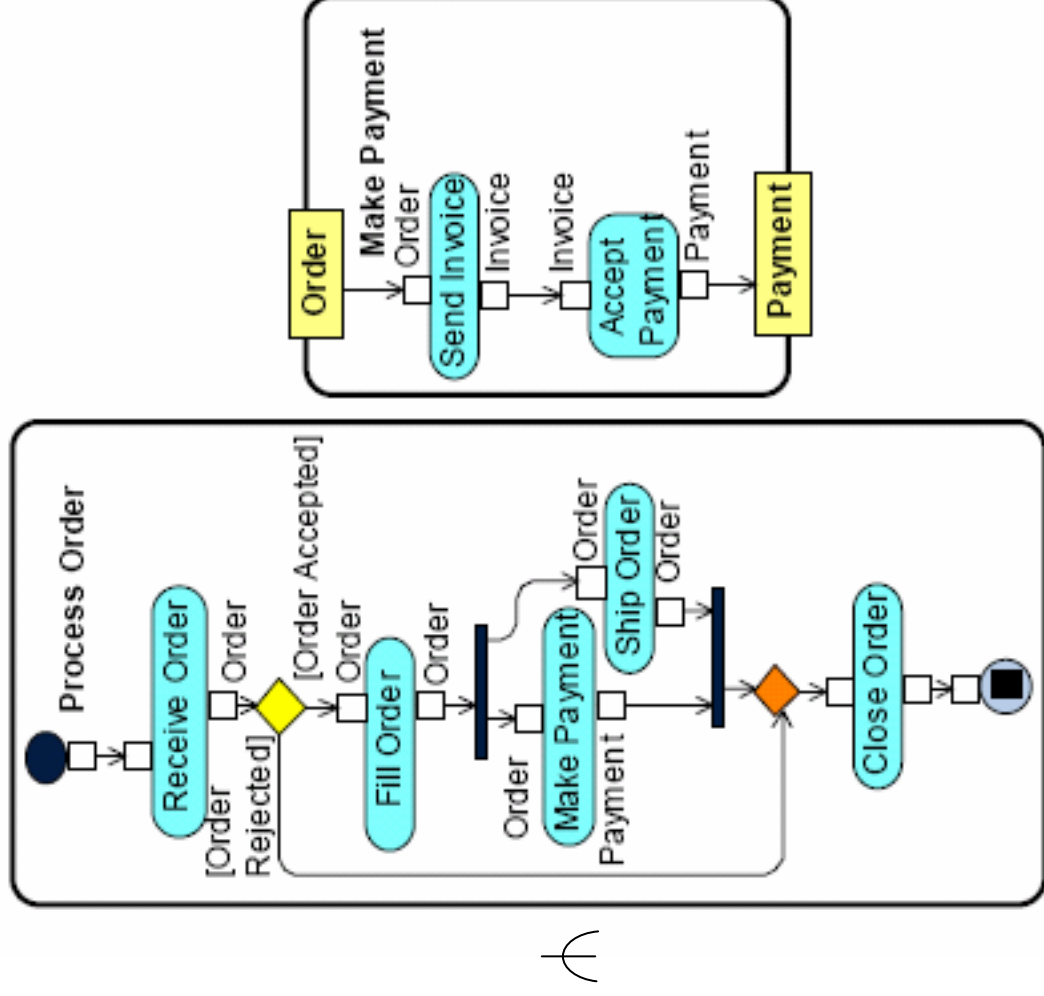


A Simple Example – Order Processing



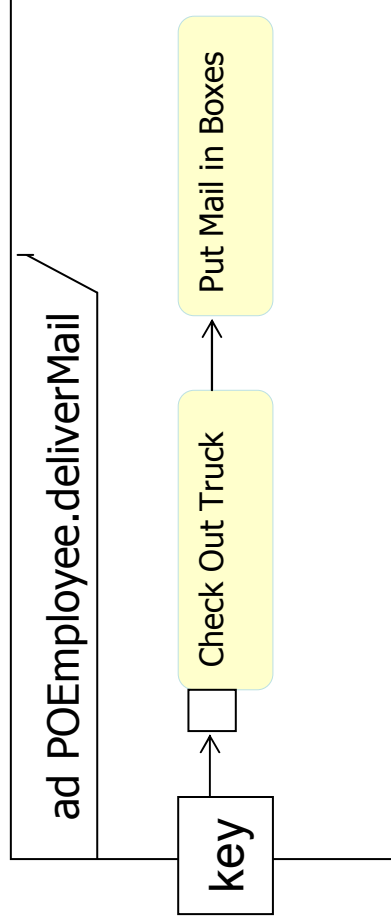
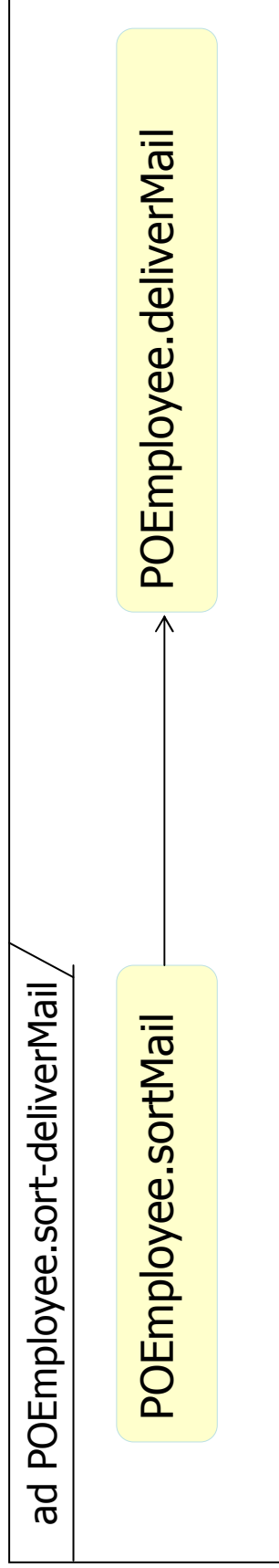
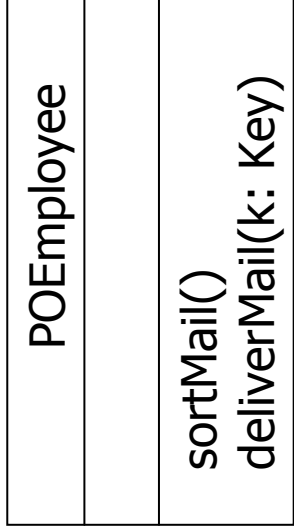
What if using swimlanes?

A Simple Example – Order Processing Using sub-activity



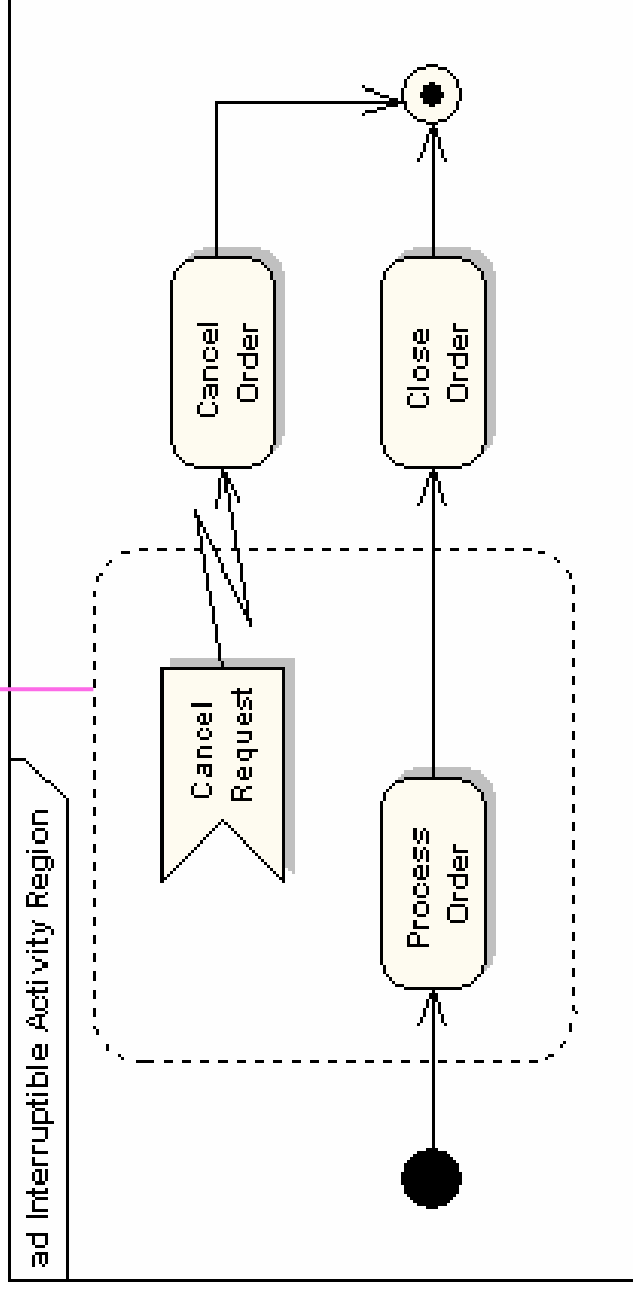
Is this the same as the previous one?

Activity Diagram even as Method



Interruptible Activity Region

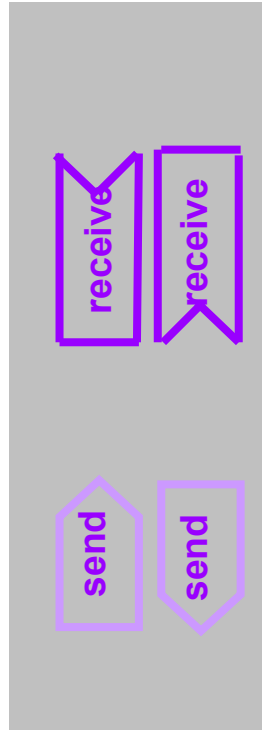
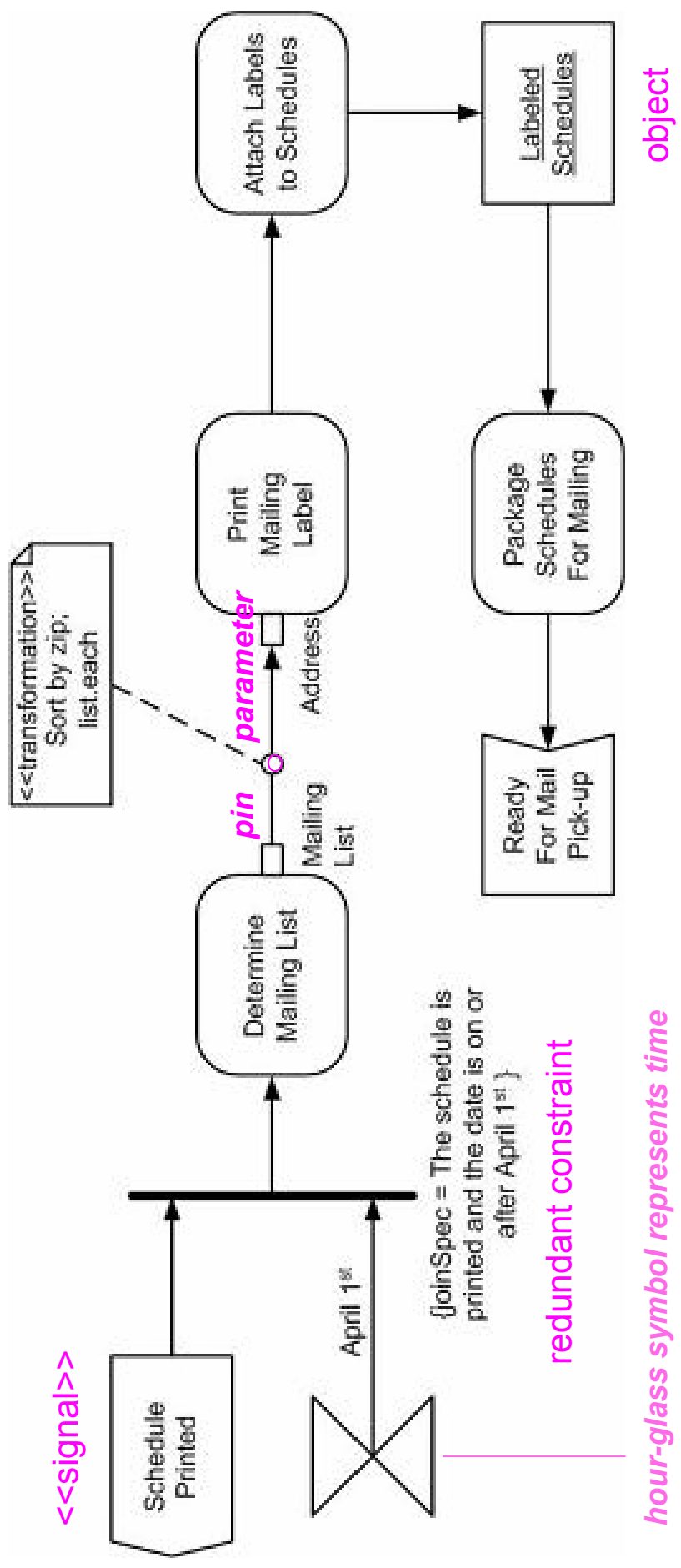
- An *interruptible activity region* surrounds a group of actions that can be interrupted.



- the Process Order action will execute until completion, then pass control to the Close Order action, unless a Cancel Request interrupt is *received* which will pass control to the Cancel Order action.

An Activity Diagram – Distributing schedules

<http://www.agilemodeling.com/artifacts/activityDiagram.htm>



Pins, Parameters, Effects

(www.jot.fm/issues/issue_2004_01/column3.pdf)

- ❑ **effect** that their actions have on objects that move through the pin: one of the four values *create*, *read*, *update*, or *delete*.
- ❑ Take Order creates an instance of Order and Fill Order reads it.
- ❑ The *create* effect is only possible on *outputs*; and the *delete* effect is only possible on *inputs*.

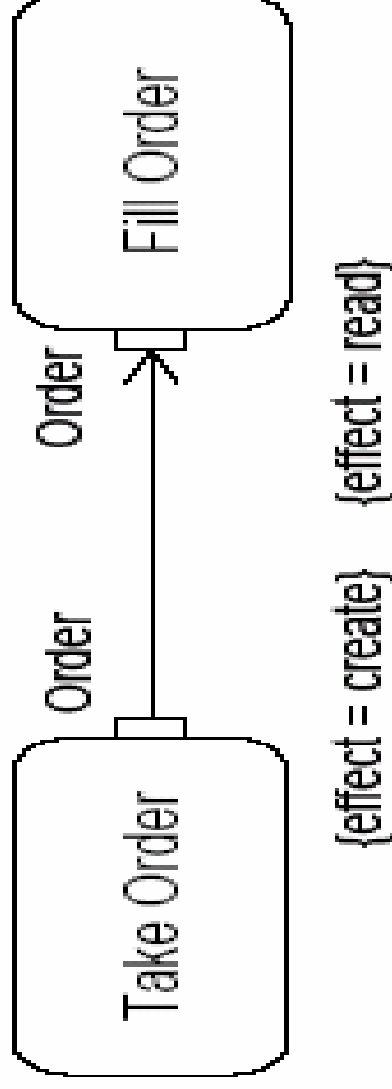
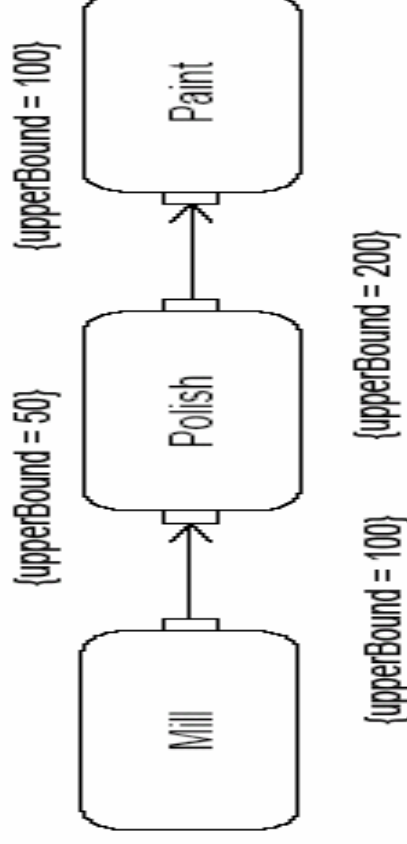


Figure 4: Effect

Multiple Tokens

- ❑ Object nodes can hold more than one value at a time, and some of these values can be the same.
- ❑ **Upper bound**: the maximum number of tokens an object node can hold, including any duplicate values.
- ❑ At runtime, when the number of values in an object node reaches its upper bound, it cannot accept any more.
- ❑ If painting is delayed too much for some reason, the input pin will reach its upper bound, and parts from polishing will not be able to move downstream;
If painting is delayed further, the output pin of polishing will fill up and the polishing behavior will not be able to transfer out polished parts;
Unless the polishing behavior has an object node internal to it that buffers output parts, it will not be able to take parts from its input pin, which will likewise fill up and propagate the backup; Only when the input pin to PAINT goes below its upper bound will parts be able to flow again.



Multiple Tokens - Ordering

- Object nodes holding multiple values can specify the **order** in which values move downstream.
- The default is FIFO (a pipe); users can change this to LIFO (a queue), or specify their own behavior to select which value is passed out first.

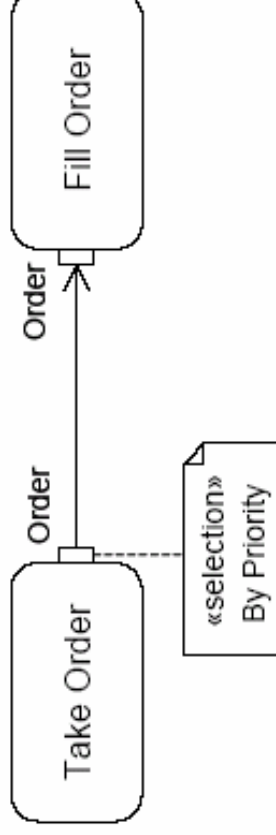


Figure 6: Selection Behavior

Non-Determinism

(http://www.jot.fm/issues/issue_2004_01/column3.pdf)

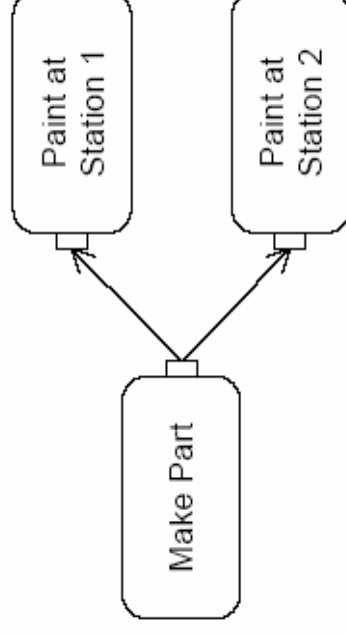


Figure 10: Token Competition

Parameter Multiplicity & Object Flow Weight

- **Minimum multiplicity** on an **input** parameter means a behavior or operation cannot be invoked by an action until the number of values available at each of its input pins reaches the minimum for the corresponding parameter, which might be zero

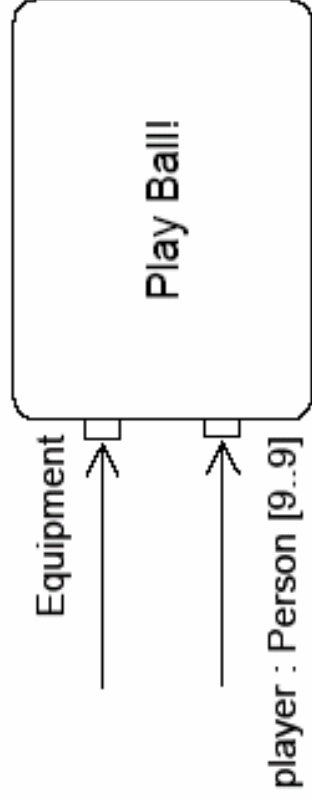


Figure 8: Parameter Multiplicity

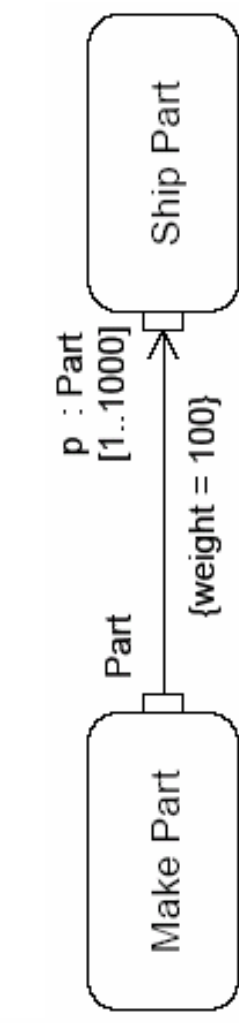


Figure 9: Object Flow Weight

- Weight on object flow edges specifies the minimum number of values that can traverse an object flow edge at one time.

Interaction Overview Diagrams

Behavioral Diagrams

- Use case
- Statechart
- Activity

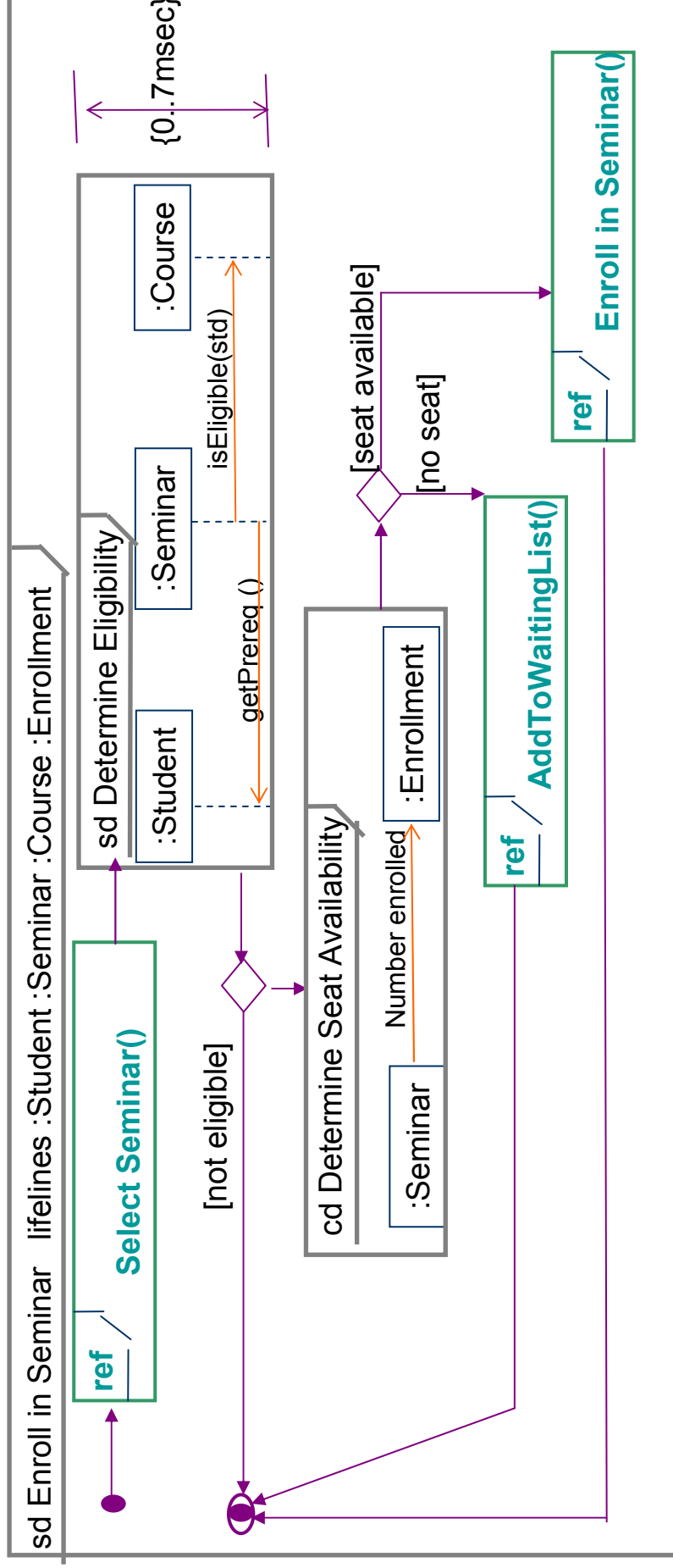
Interaction Diagrams

- Sequence;
Communication
- **Interaction**
- **Overview**
- Timing

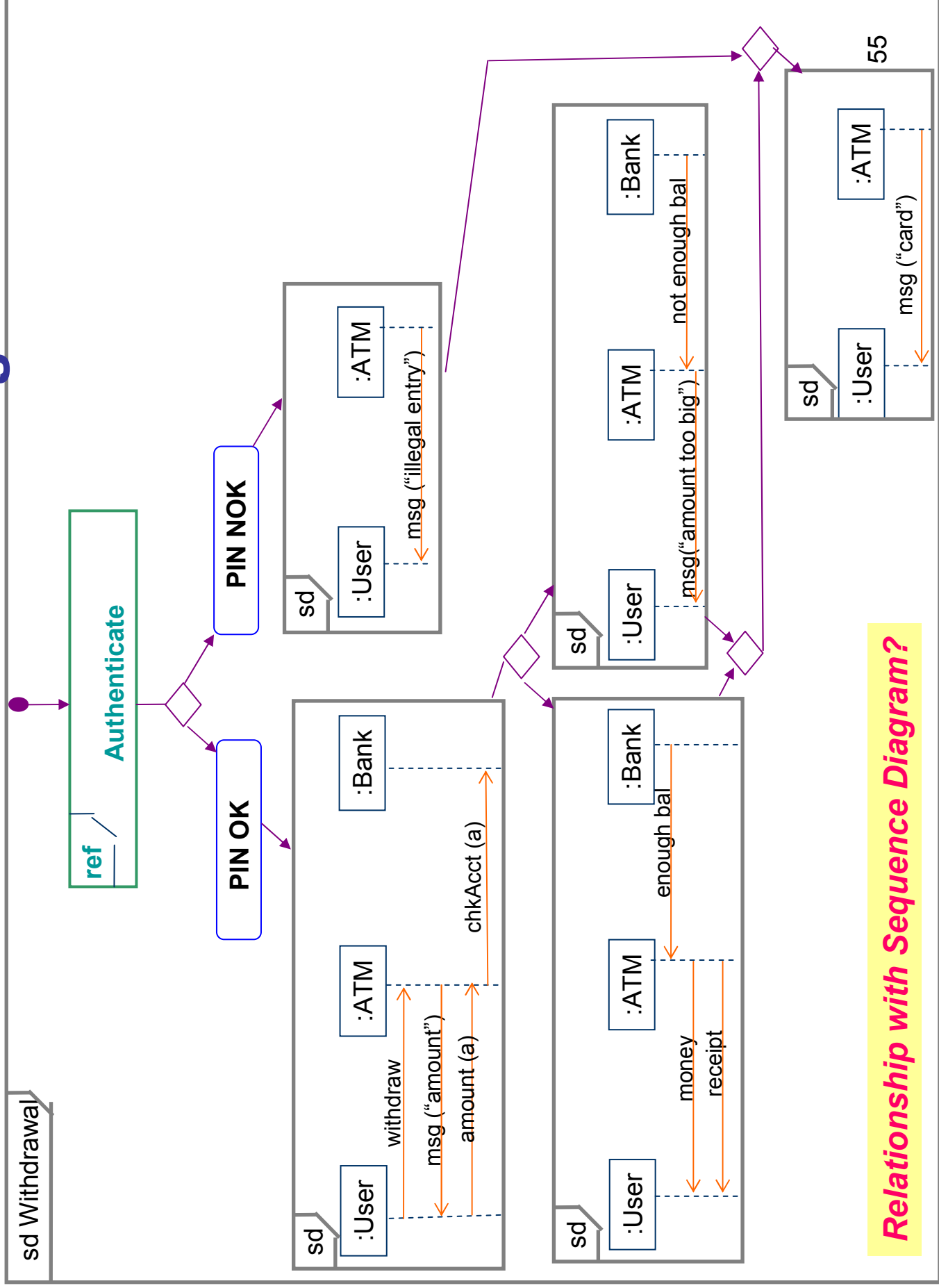
Interaction Overview Diagram

(<http://www.agilemodeling.com/artifacts/interactionOverviewDiagram.htm>)

- ❑ variants on UML activity diagrams which overview control flow.
- ❑ The nodes within the diagram are **frames**, not activities
- ❑ Two types of frame shown:
 - ❑ interaction frames depicting any type of UML interaction diagram (sequence diagram: *sd*, communication diagram: *cd*, timing diagram: *td*, interaction overview diagram: *iod*)
 - ❑ interaction occurrence frames (*ref*; typically anonymous) which indicate an activity or operation to invoke.



Interaction Overview Diagram



Relationship with Sequence Diagram?

Timing Diagrams

Behavioral Diagrams

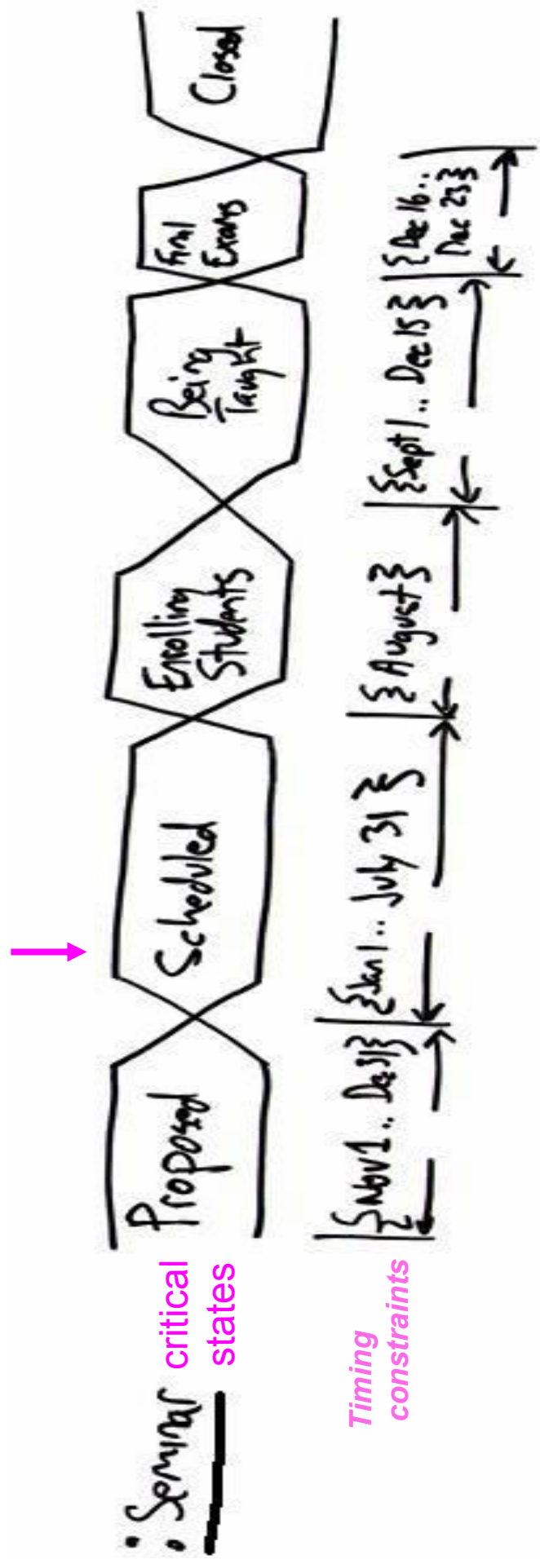
- Use case
- Statechart
- Activity

Interaction Diagrams

- Sequence;
Communication
- Interaction Overview
- **Timing**

Interaction Diagram: Timing Diagram

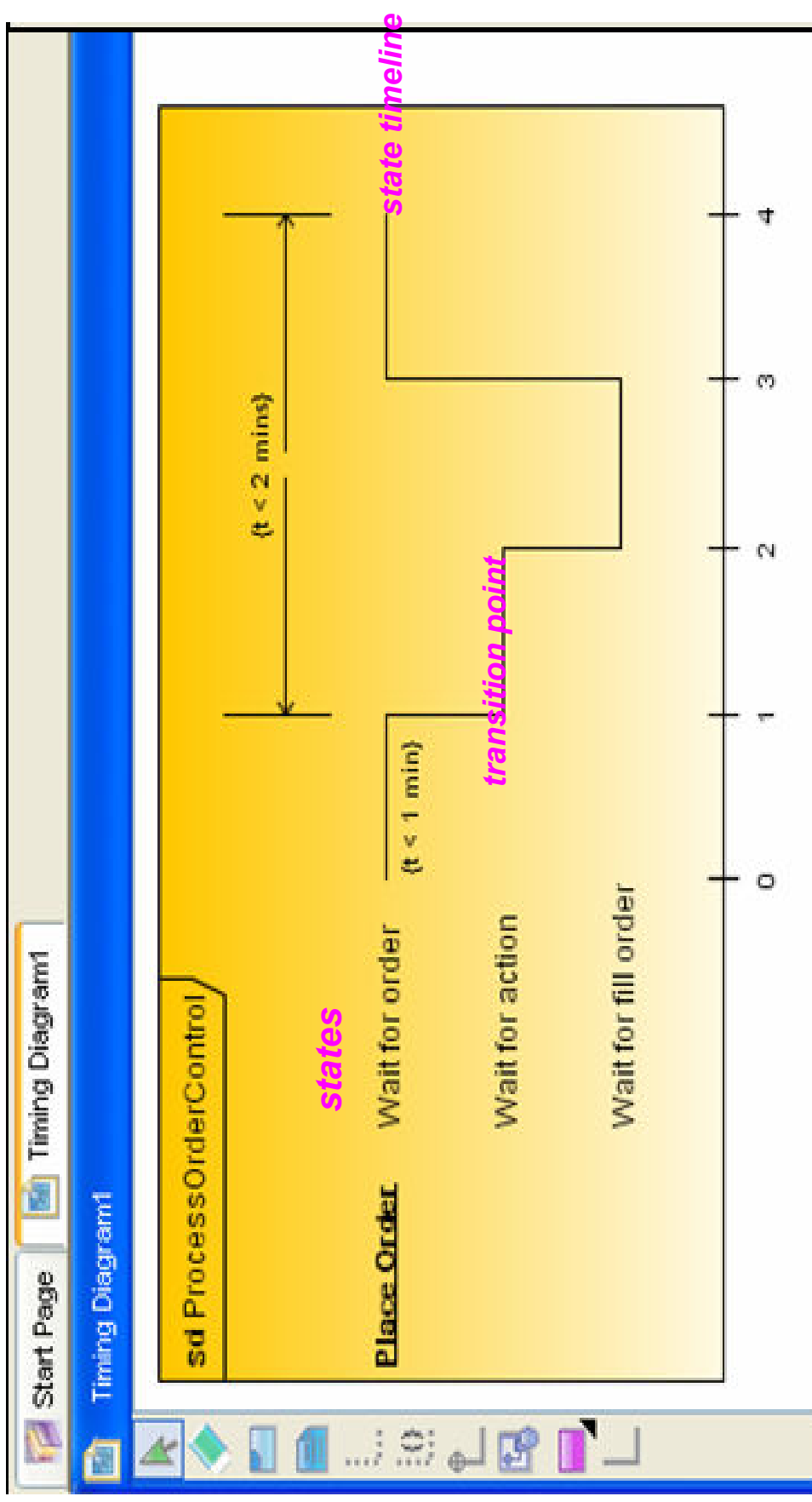
- To explore the behaviors of 0..* objects throughout a given period of time.
- Two basic flavors: **concise** notation and robust notation



- The lifecycle of a single seminar**
- The **critical states** – Proposed, Scheduled, Enrolling Students, Being Taught, Final Exams, Closed
 - The two lines surrounding the states are called a **general value lifeline**.
 - When the two lines cross one another it indicates a **transition point** between states.
 - **Timing constraints** along the bottom of the diagram, indicating the period of time during which the seminar is in each state.

Interaction Diagram: Timing Diagram (robust notation)

<http://www.visual-paradigm.com/highlight/highlightuml2support.jsp>



timing ruler w. tick marks

Can you transform this into a concise notation?

Appendix: Miscellaneous

Role Names

/ ClassifierRoleName : ClassifierName



A role name is preceded by a '/'

Example:

/ Parent : Person

/ Parent

: Person

A classifier name is preceded by a ':'

instanceName / ClassifierRoleName : ClassifierName

Example:

:Person

Charlie

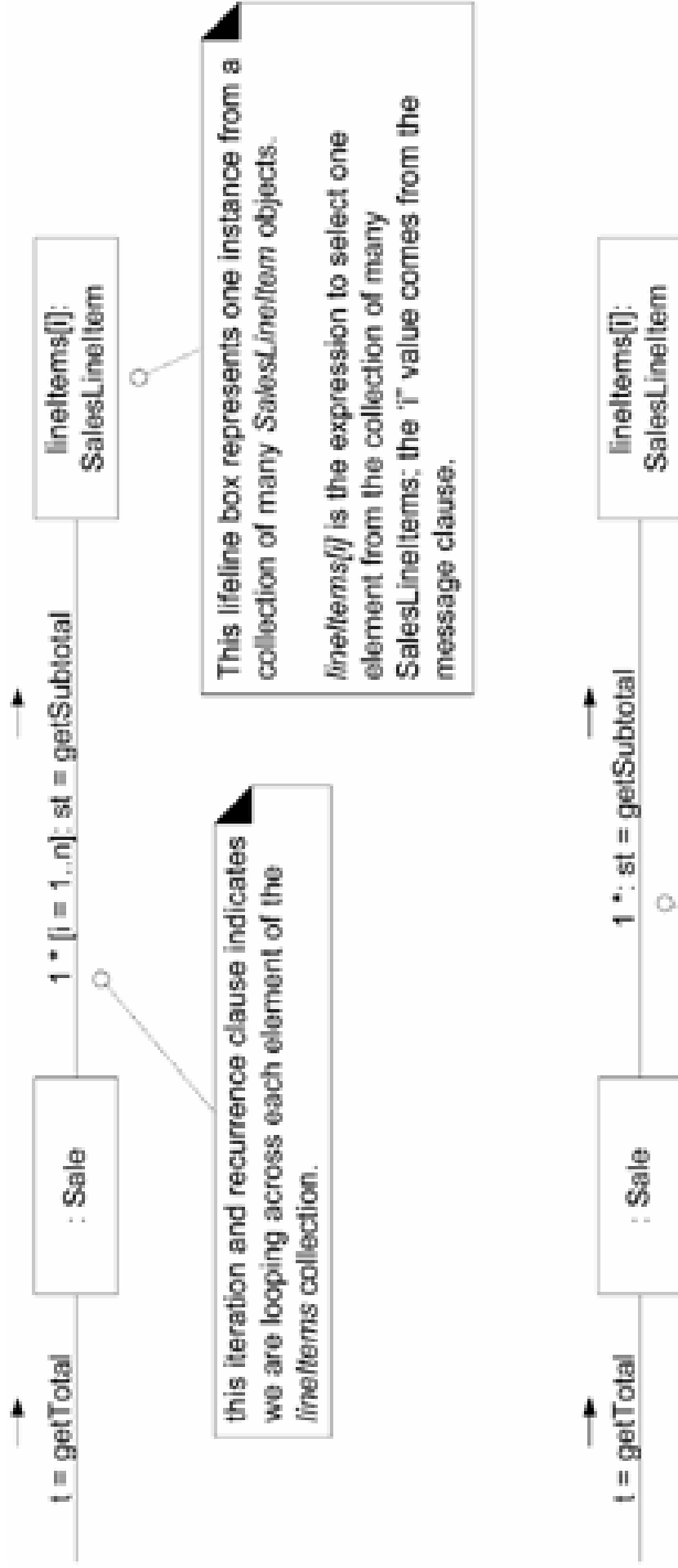
Charlie : Person

Charlie / Parent

Charlie / Parent : Person

Iterative Messages

[Craig Larman] [<http://www.phptr.com/articles/article.asp?p=360441&seqNum=6&rl=1>]



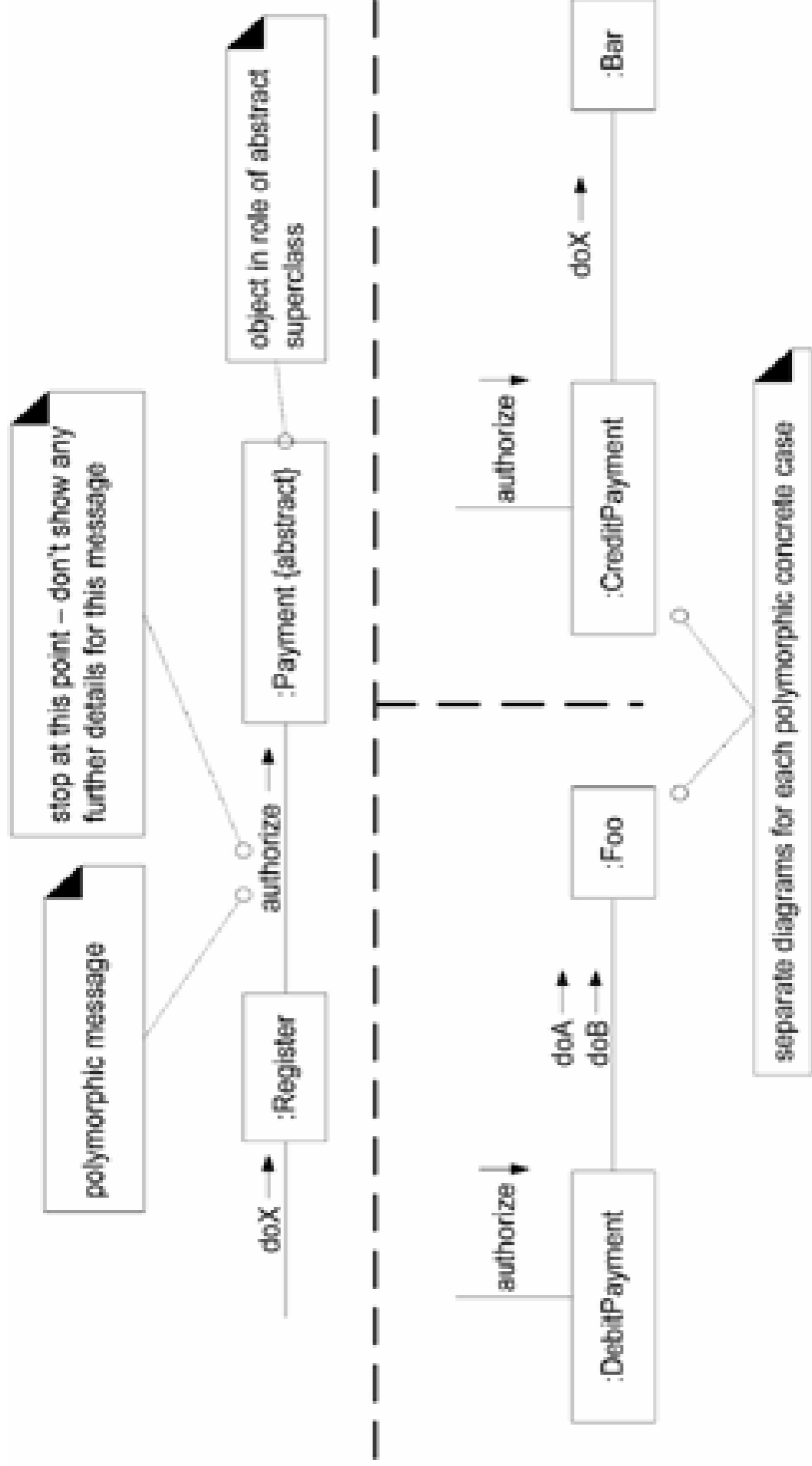
this iteration and recurrence clause indicates we are looping across each element of the `lineItems` collection.

This lifeline box represents one instance from a collection of many `SalesLineItem` objects. `lineItems()` is the expression to select one element from the collection of many `SalesLineItems`; the `i` value comes from the message clause.

Less precise, but usually good enough to imply iteration across the collection members

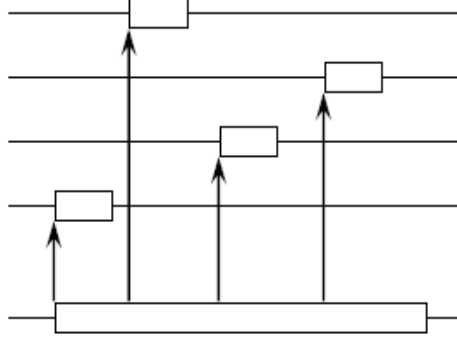
Polymorphic Message

[Craig Larman] [<http://www.phptr.com/articles/article.asp?p=360441&seqNum=6&rl=1>]



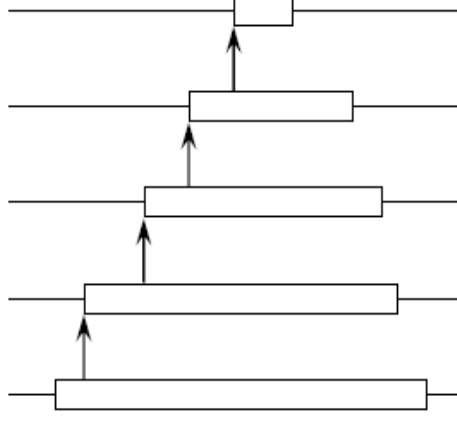
Sequence Diagram Shapes

Fork - centralised



- operations can change order
- new operations may be added

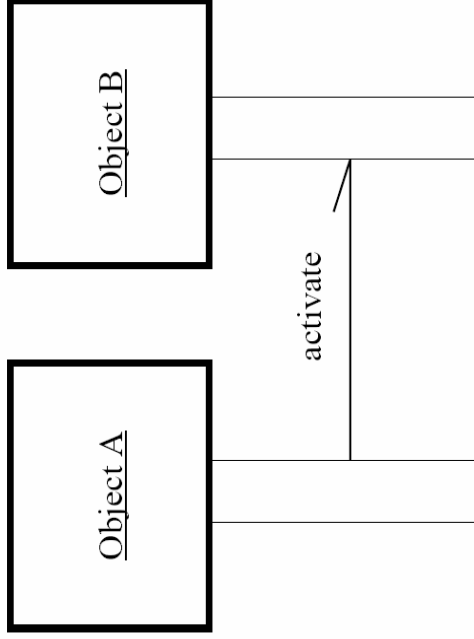
Stair - decentralised



- Operations have strong connections
- performed in same order
- behaviour is encapsulated

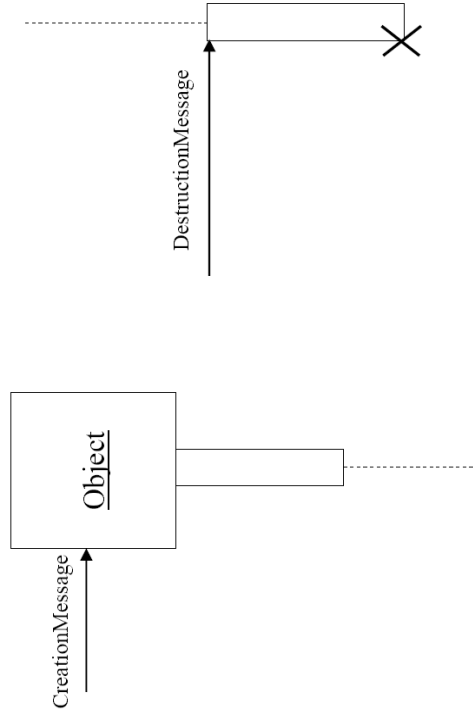
Concurrency

In some systems, objects run concurrently, each with its own thread of control. If the system uses concurrent objects, it is shown by activation, by asynchronous messages and by active objects.



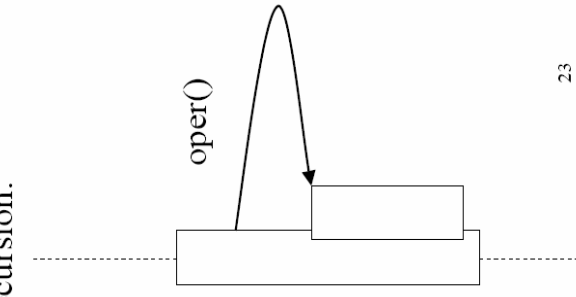
21

Creation and Destruction

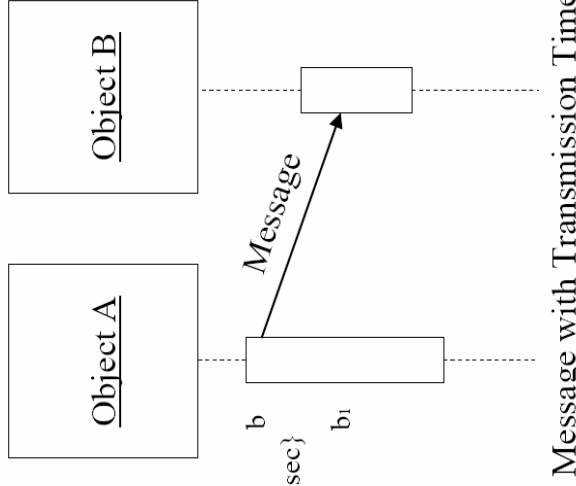


22

Recursion:



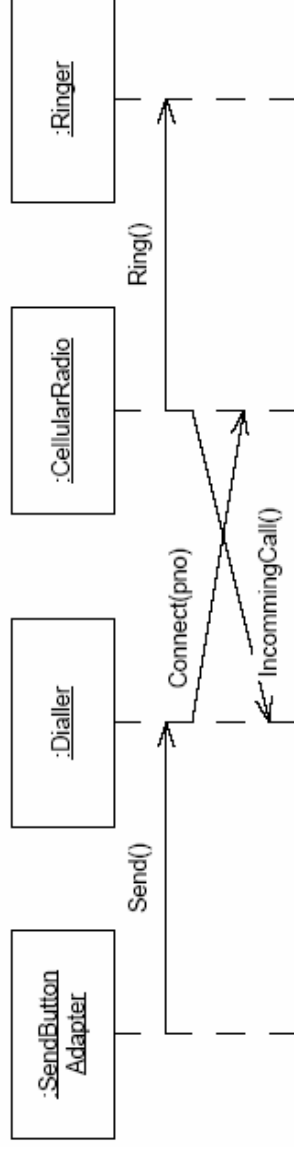
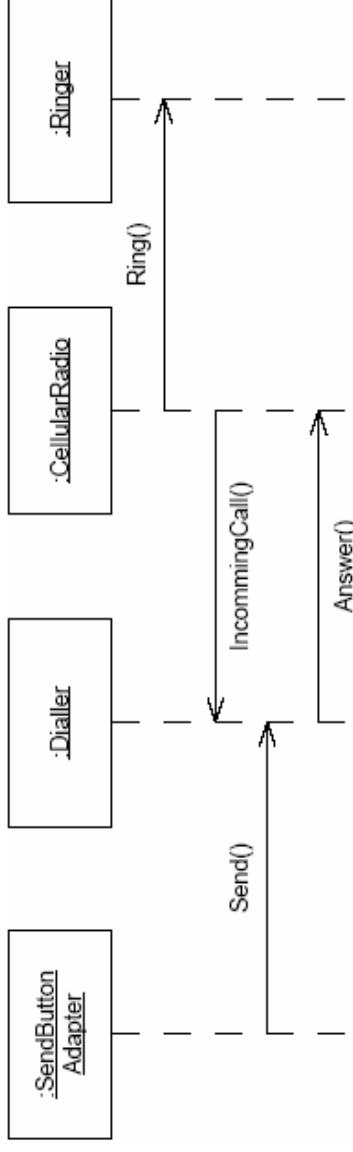
23



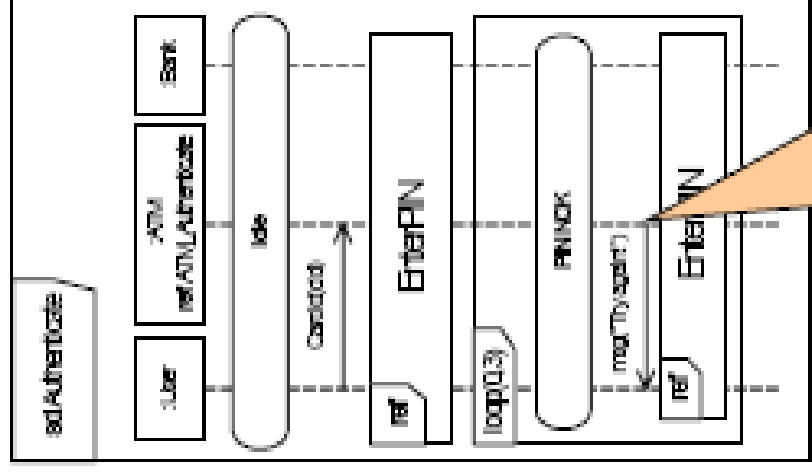
Message with Transmission Time

Race conditions

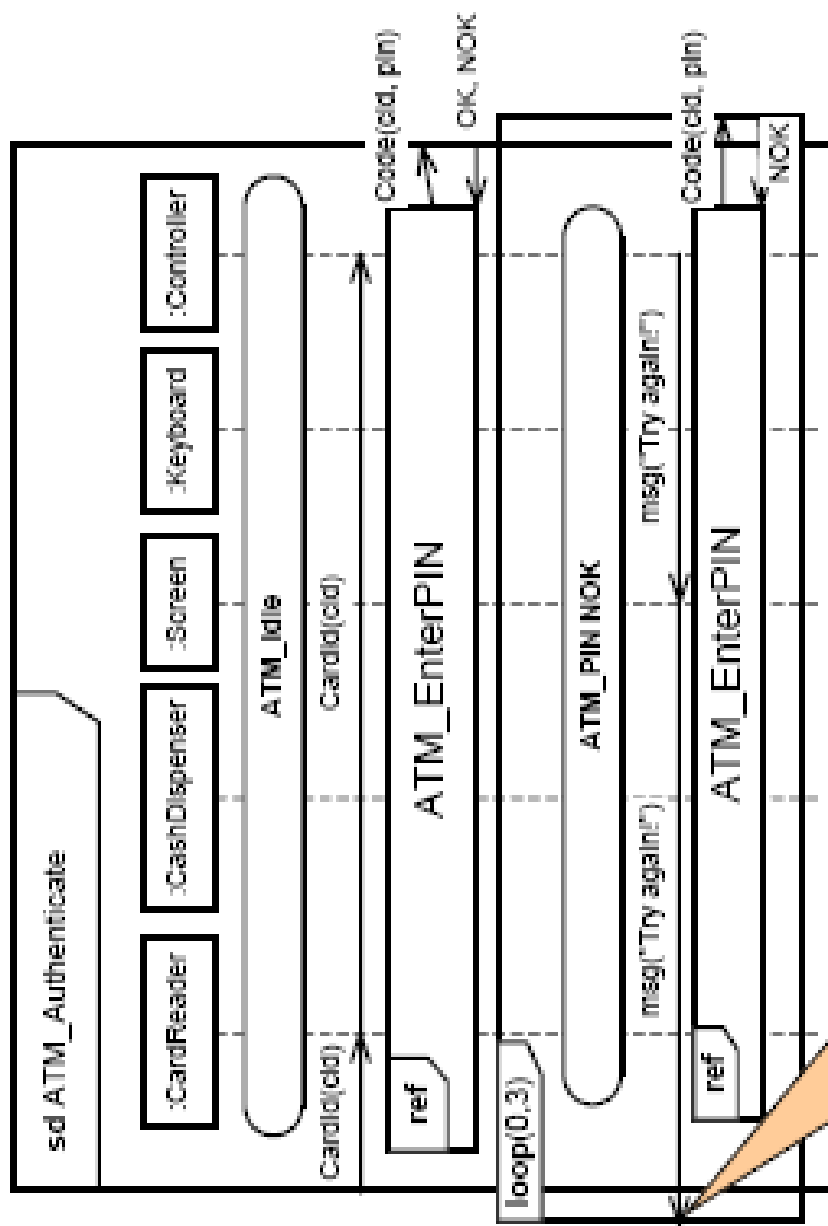
- E.g. an object receives two messages
- Order of arrival changes behaviour
- Only one order leads to correct behaviour
 - CellularRadio expects Answer() not Connect(pno)
- Two states when Send can be pressed
 - To make outgoing call (after dialling digits)
 - To answer incoming call
- State diagram can be useful here
 - To help realize there is a race condition
 - To specify what should happen
- Angled arrows- Show message delay



Decomposition



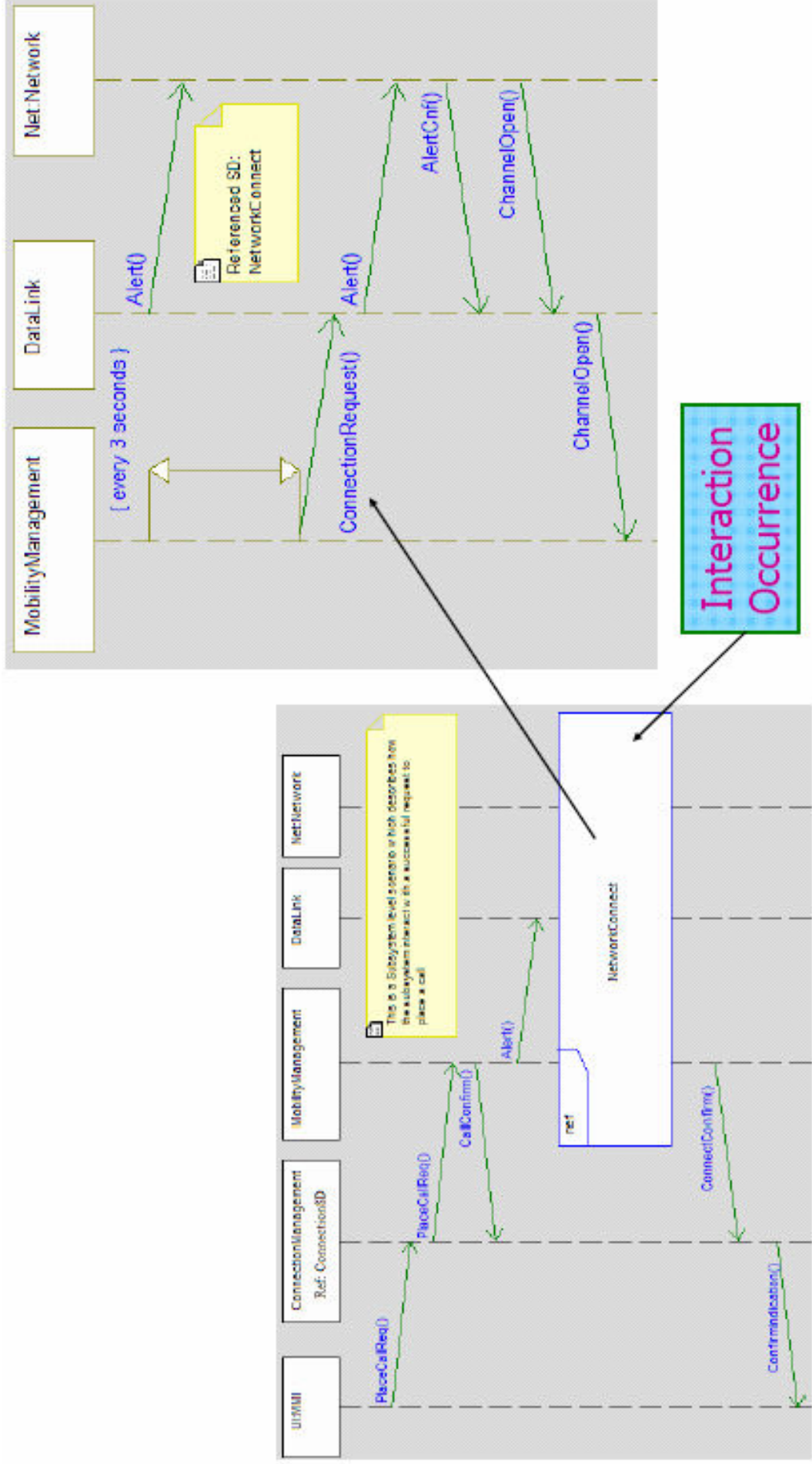
notice the correspondence!



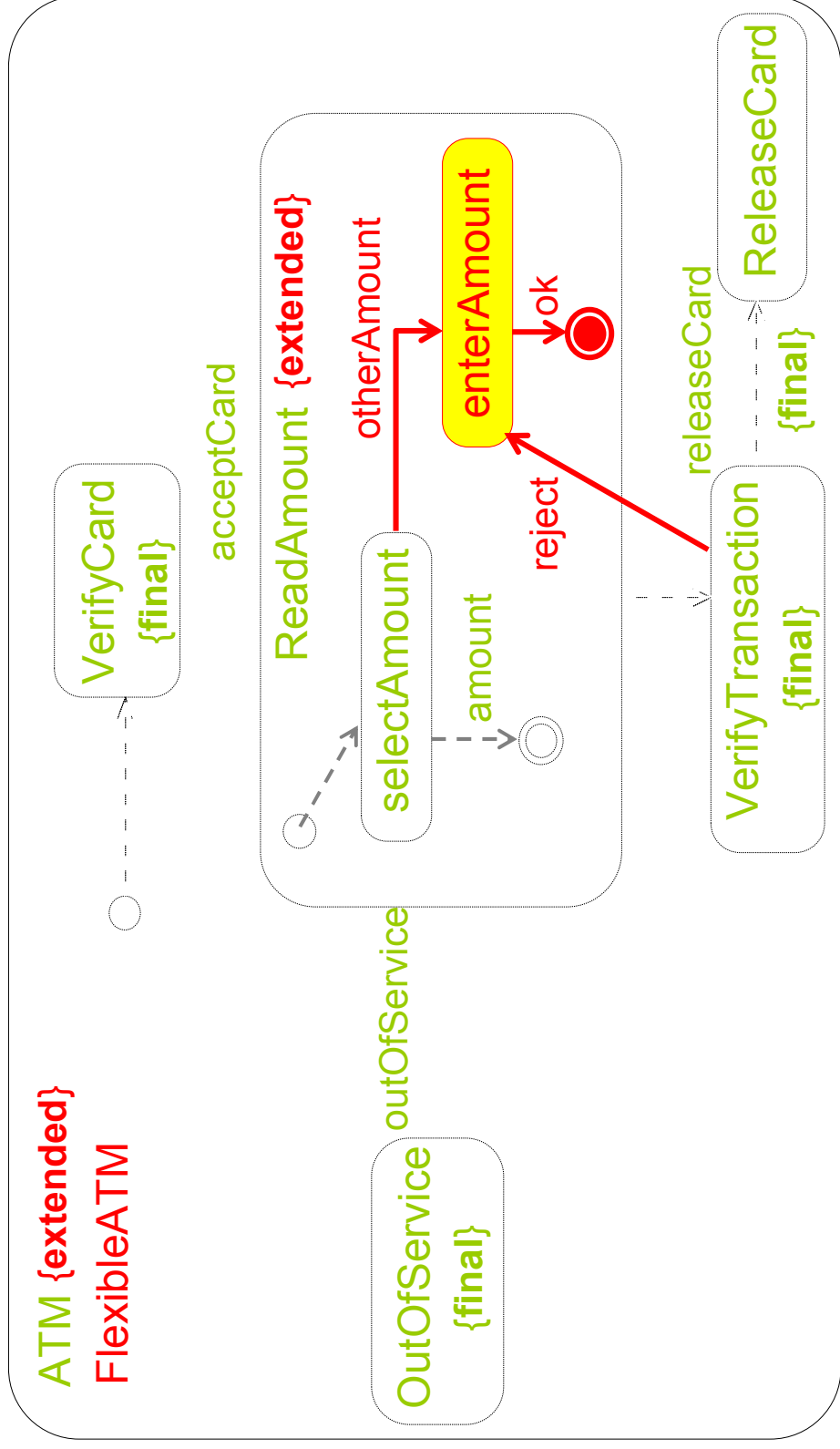
notice the correspondence!

Sequence Diagram - Reference

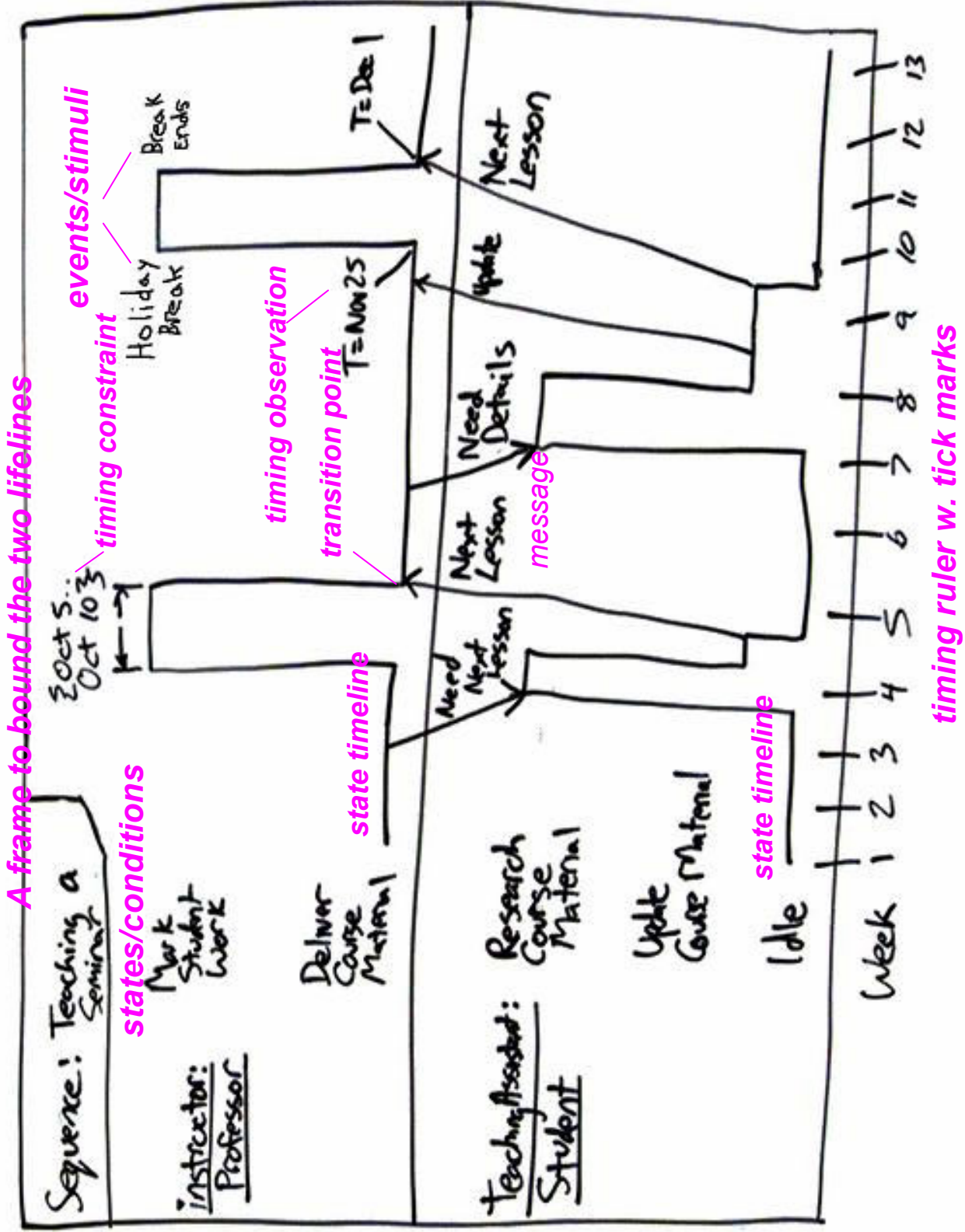
(www.cs.tut.fi/tapahtumat/olio2004/richardson.pdf)



State Machine Redefinition

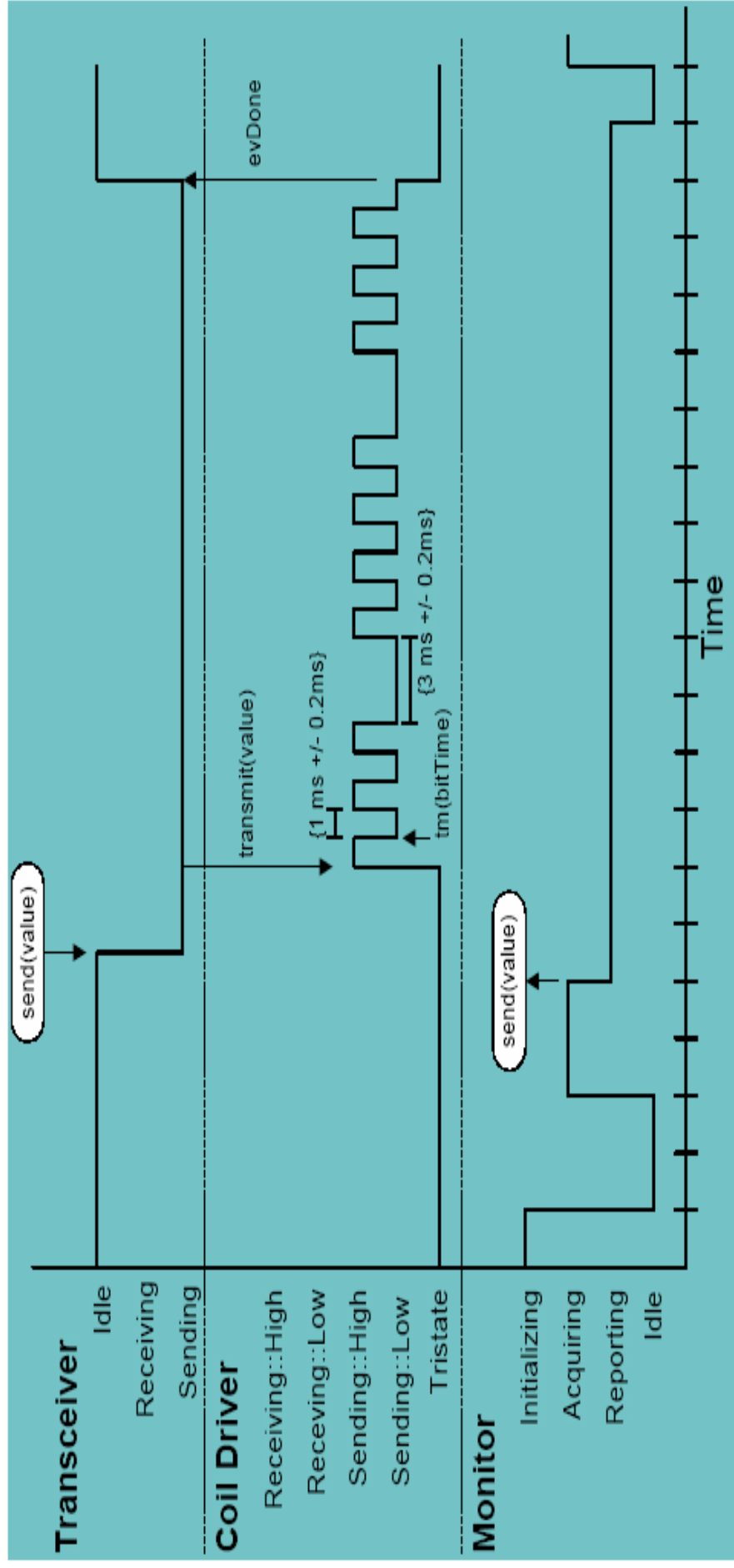


Interaction Diagram: Timing Diagram (robust notation)

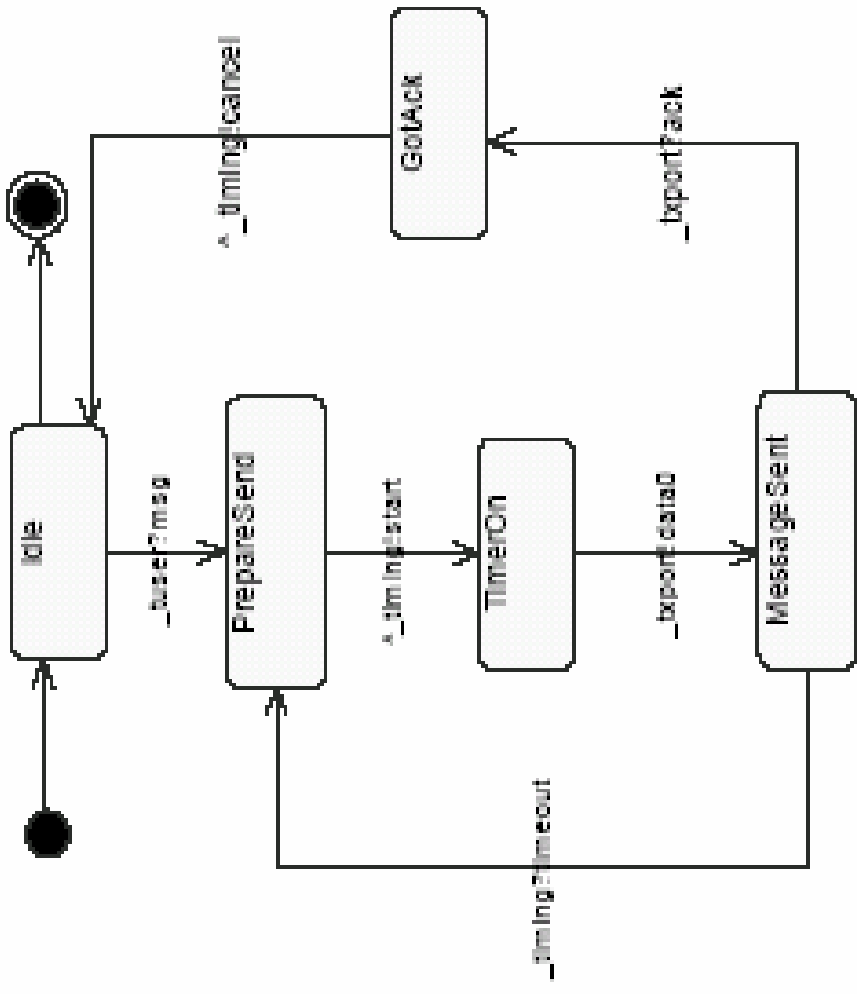
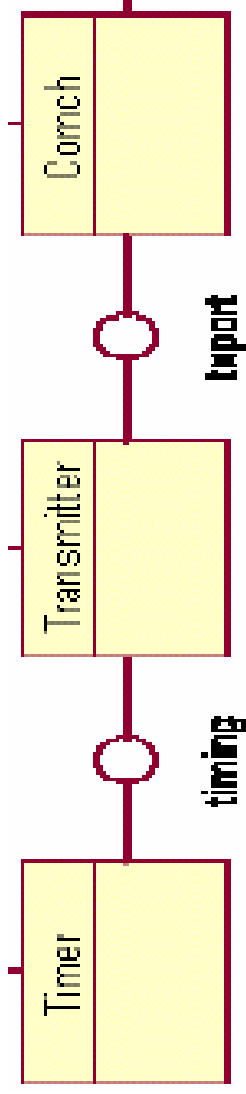


Timing Diagram – another example

(www.cs.tut.fi/tapahtumat/olio2004/richardson.pdf)



Real-Time Extensions: Using CCS



_timing?timeout ^_txport!data0

_timing: interface/connection

?: receive

timeout: event

^: send

_txport: interface/connection

!: send

data0: event

+: multiple receive

“.”: multiple send

Figure 4: Normalized Transmitter Component