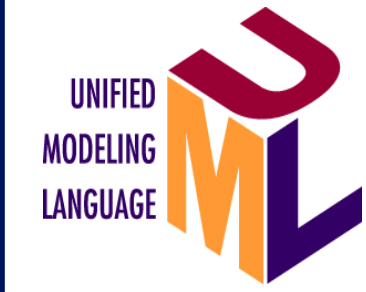
THE UNIVERSITY OF BRITISH COLUMBIA

EECE 310: Software Engineering

A Brief Introduction to the UML

adapted form Philippe Kruchten's slides



Outline

- Purpose & genesis
- Reminder on objects and classes
- UML elements
- Key UML Diagrams
- From Notation to Code
- UML Tools
- UML References & resources

What is a model?

- A miniature representation of something.
- A semantically closed abstraction of a system under study.
- A representation of a system that allows for investigation of the properties of the system.

What is UML?

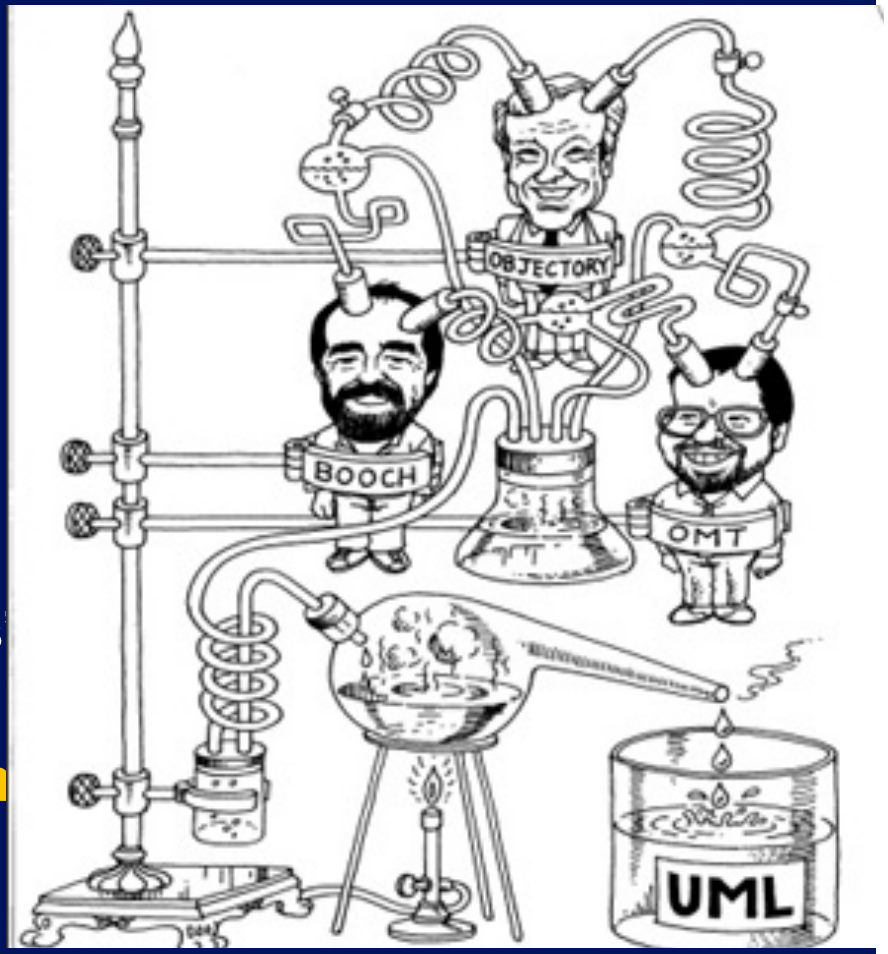
- Notation
  - Electronics analogy
  - Map analogy
- Syntax and semantics
  - Casual notation and formal notation
- Usage:
  - Illustration
  - Forward engineering: Model $\Rightarrow$ Code
  - Round-trip engineering: Model $\Leftrightarrow$ Code

What UML is not

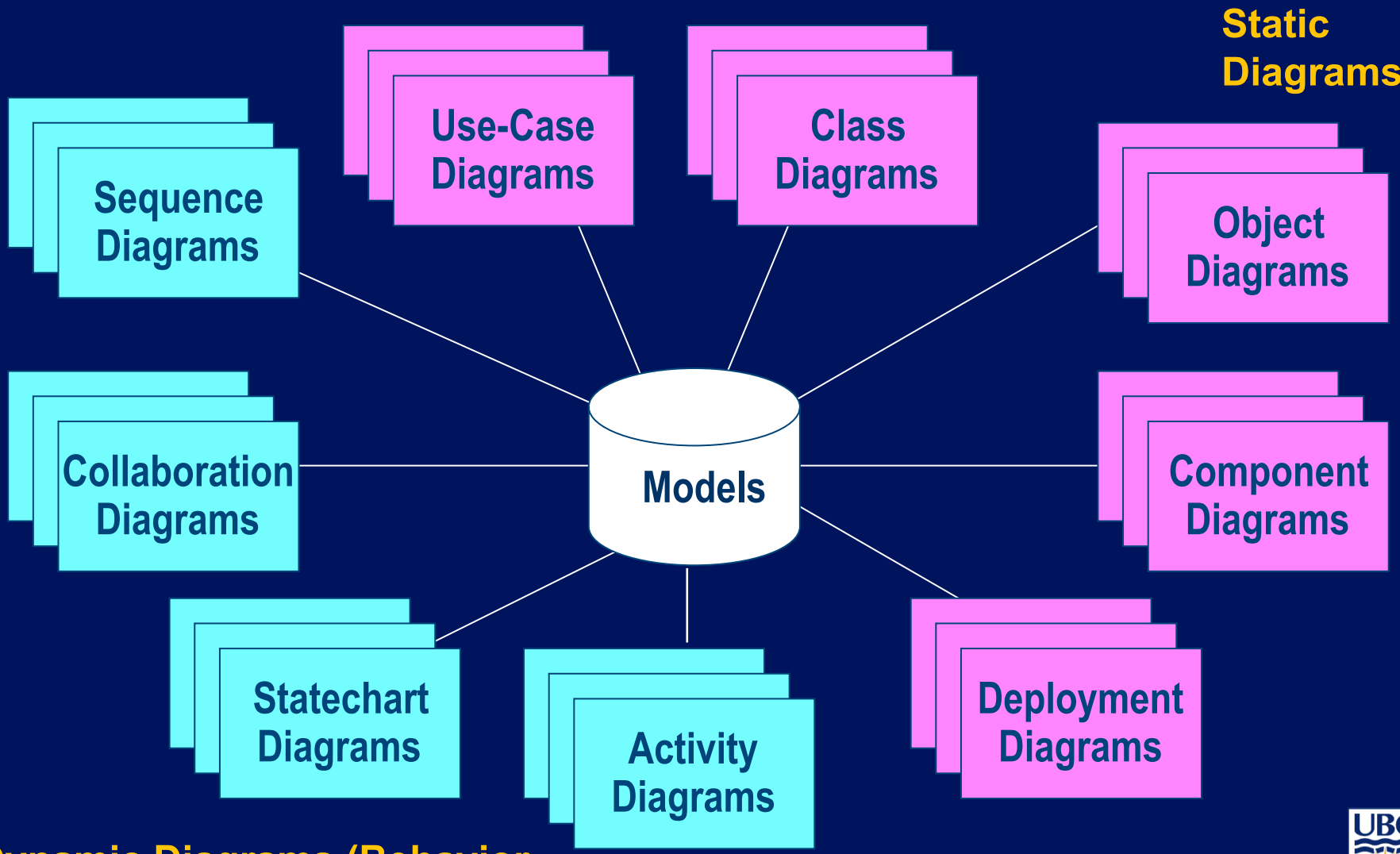
- *Not* a *method* in itself
  - A notation designed to support various methods for requirement analysis and software design
    - E.g., (IBM) Rational Unified Process (RUP)

A Brief History of UML

- Language ‘wars’ (1985-95):
  - OOPSLA conferences as the main battlefield
- Contenders
  - OMT (JimRumbaugh)
  - Booch method and notation (Grady Booch)
  - OOSE (Ivar Jacobson)
  - OML (Brian Henderson-Sellers)
  - *And many others.*
- Rational Software and the “three amigos”
- Object Management Group (OMG)
- **ISO/IEC 19501:2005 Information Technology Standard— Open Distributed Processing — Unified Modeling Language (UML)**



Two types of UML diagrams



Key UML diagrams

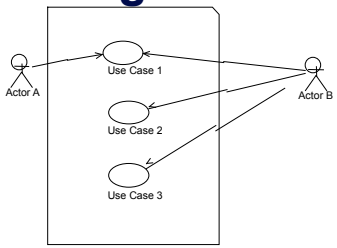
- **Class diagram**
- **Sequence diagram**

- Object diagram
- State diagram or Statechart
- Activity diagram
- Deployment diagram
- Use-case diagram
- Collaboration diagram

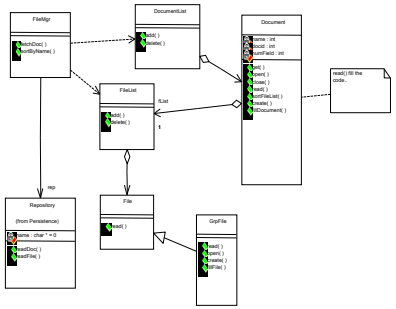
In decreasing order of usefulness for the average developer

Visual modeling of a software system

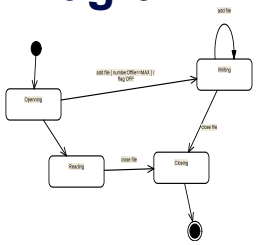
Use-Case Diagram



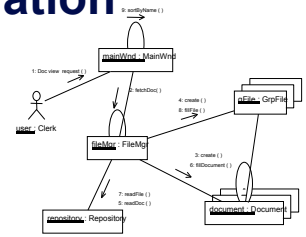
Class Diagram



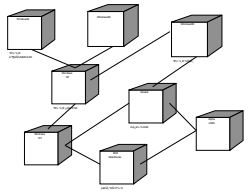
State Diagram



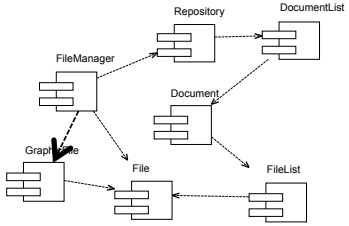
Collaboration Diagram



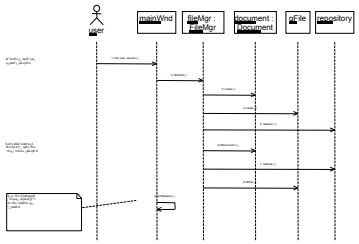
Deployment Diagram



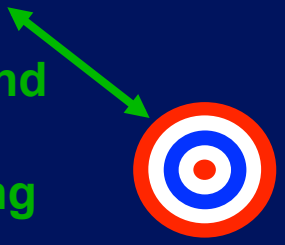
Component Diagram



Sequence Diagram



Forward and Reverse Engineering



Three views over the system

- *Functional requirements view*
  - Emphasizes the functional requirements of the system from the user's point of view.
  - Includes use case diagrams.
- *Static structural view*
  - Emphasizes the static structure of the system using objects, attributes, operations, and relationships.
  - Includes class diagrams and collaboration diagrams
- *Dynamic behavior view*
  - Emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects.
  - Includes sequence diagrams, activity diagrams and state machine diagrams.

Elements of UML Diagrams

- Model elements
- Connectors
- Adornments
- Annotations

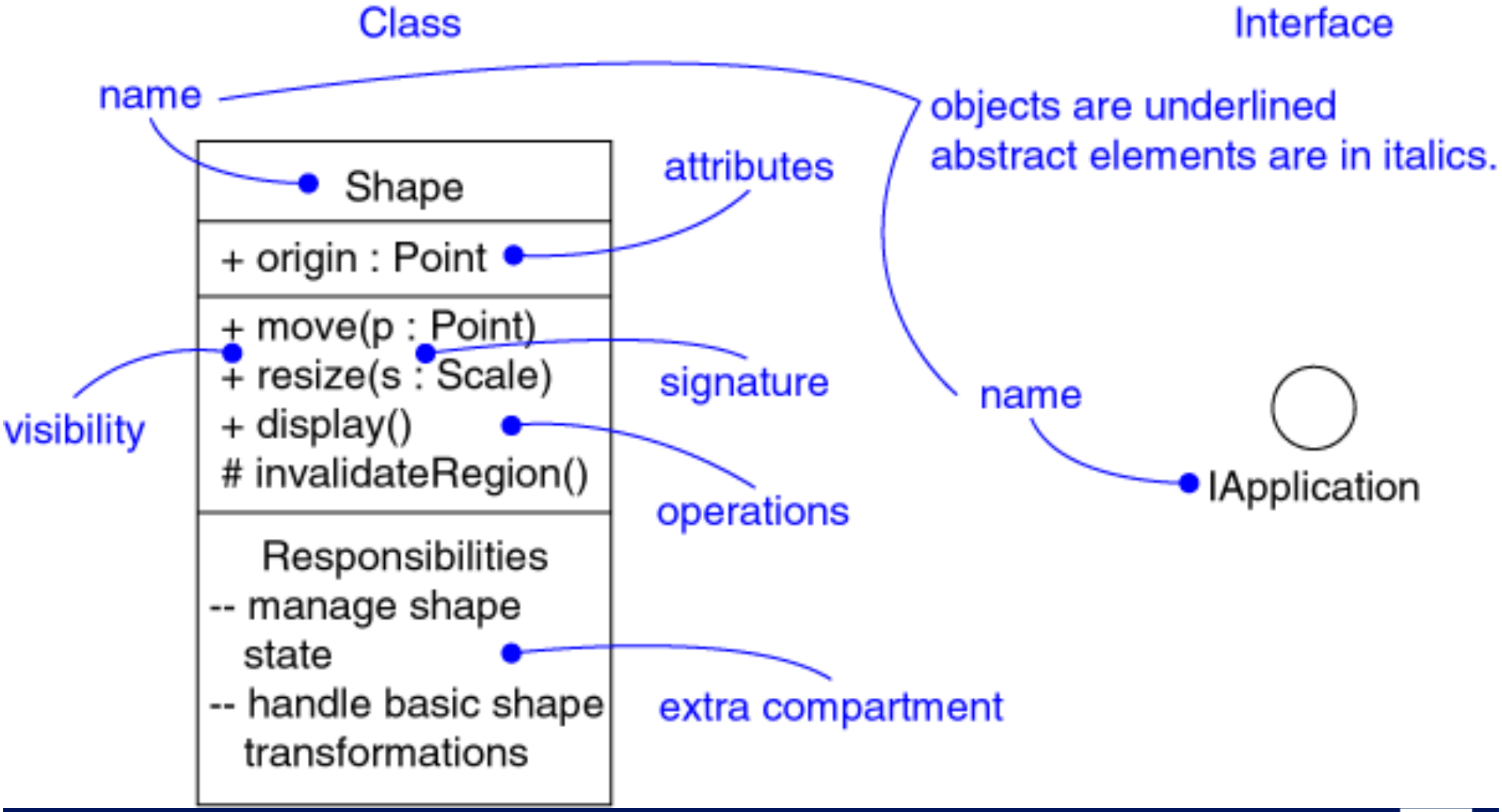
Reminder: Class

- A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics.
- A class may use a set of interfaces to specify collections of operations it provides to its environment

Reminder: Object

- An entity with a well-defined boundary and identity that encapsulates *state* and *behavior* .
  - State is represented by attributes and relationships;
  - Behavior is represented by operations, methods, and state machines.
- An object is an instance of a class.

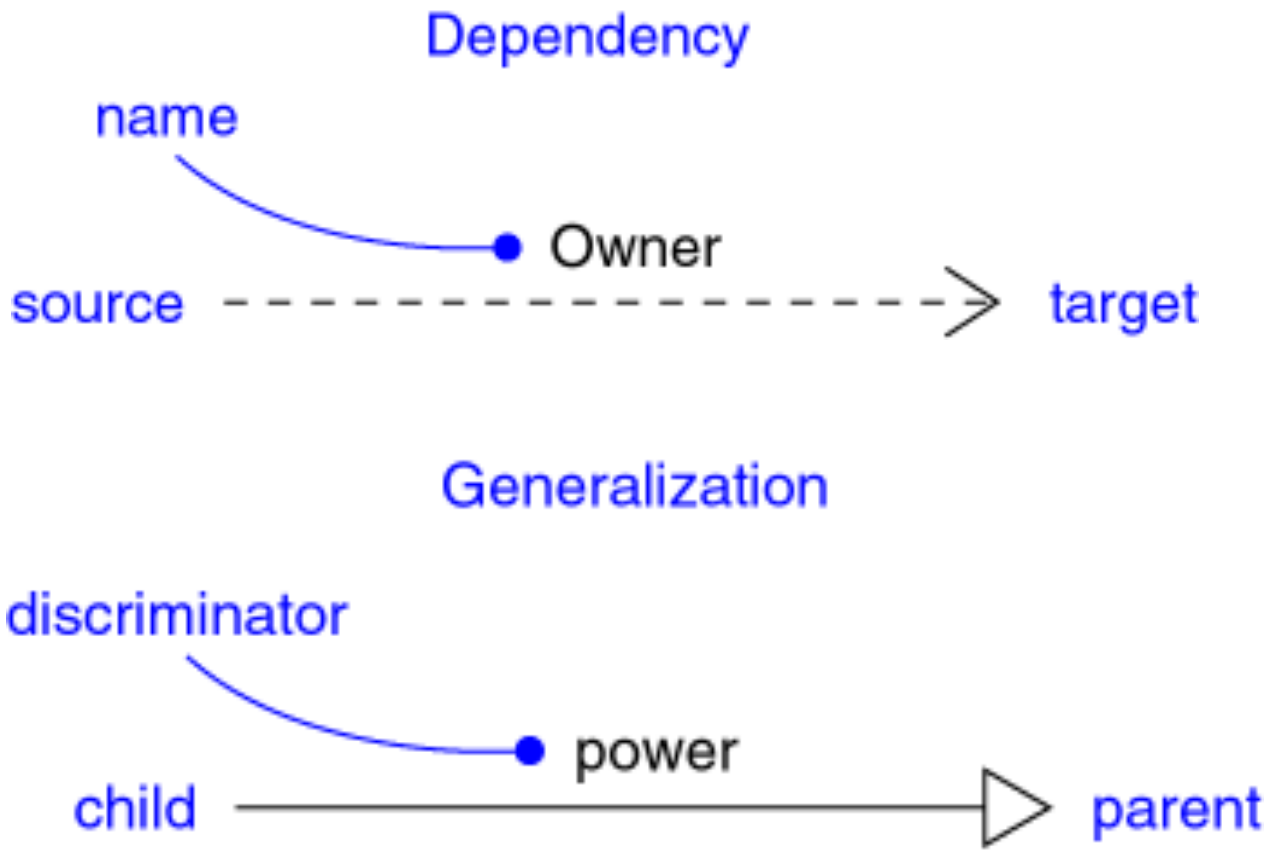
Modeling elements: class, interface



Types of relationship in class diagrams

- Class level:
  - Dependency:
    - x *depends on* y (for implementation, for example)
    - A dependency exists between two defined elements if a change *to the definition* of one would result in a change to the other.
  - Generalization (& specialization):
    - x *is a kind of* y (taxonomy, subclassing)
- Instance level
  - Association:
    - x *is a part of* y

Class Level: Dependency and generalization



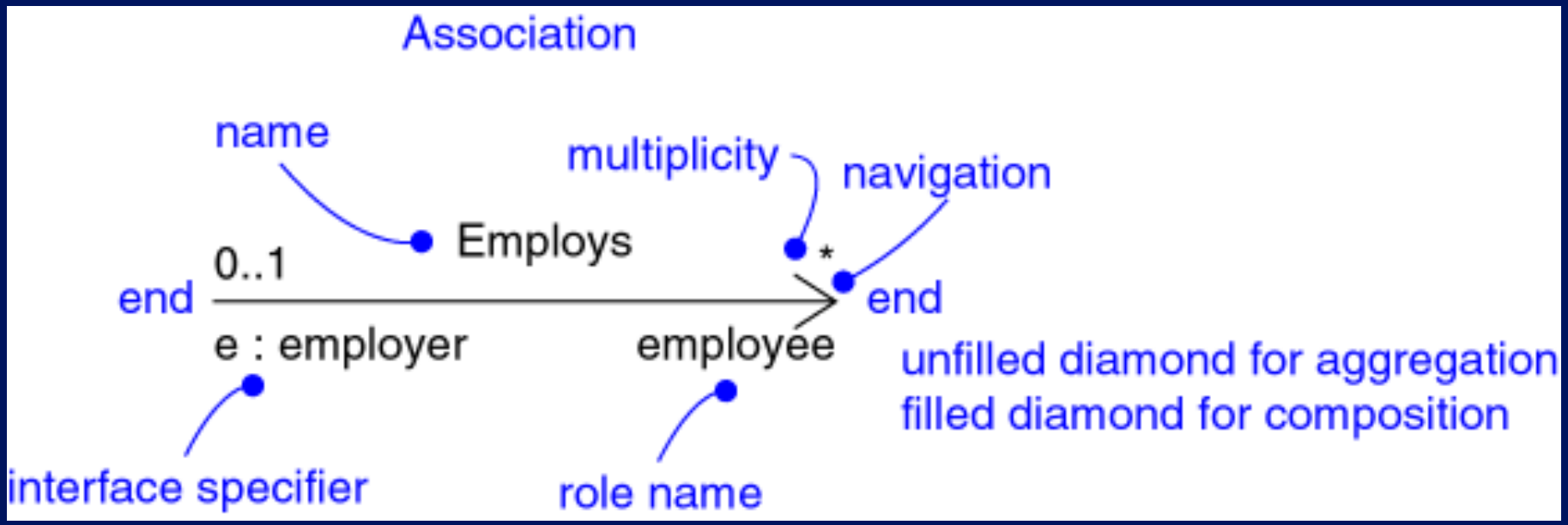
Instance-Level Relationships

Link

- The basic relationship among objects.
  - Represented as a line connecting *two or more* object boxes.
  - Shown on an object diagram or class diagram.
  - A link is an instance of an association.

Association

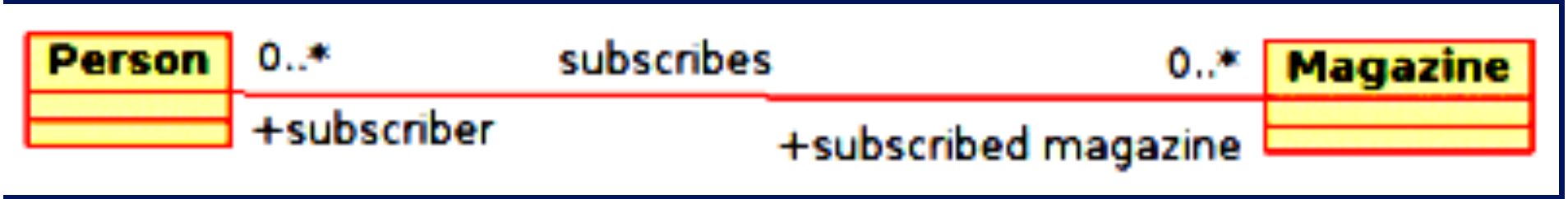
- A relationship that models a bi(or multi)-directional semantic connection among instances.
- An association represents a family of links



Multiplicity

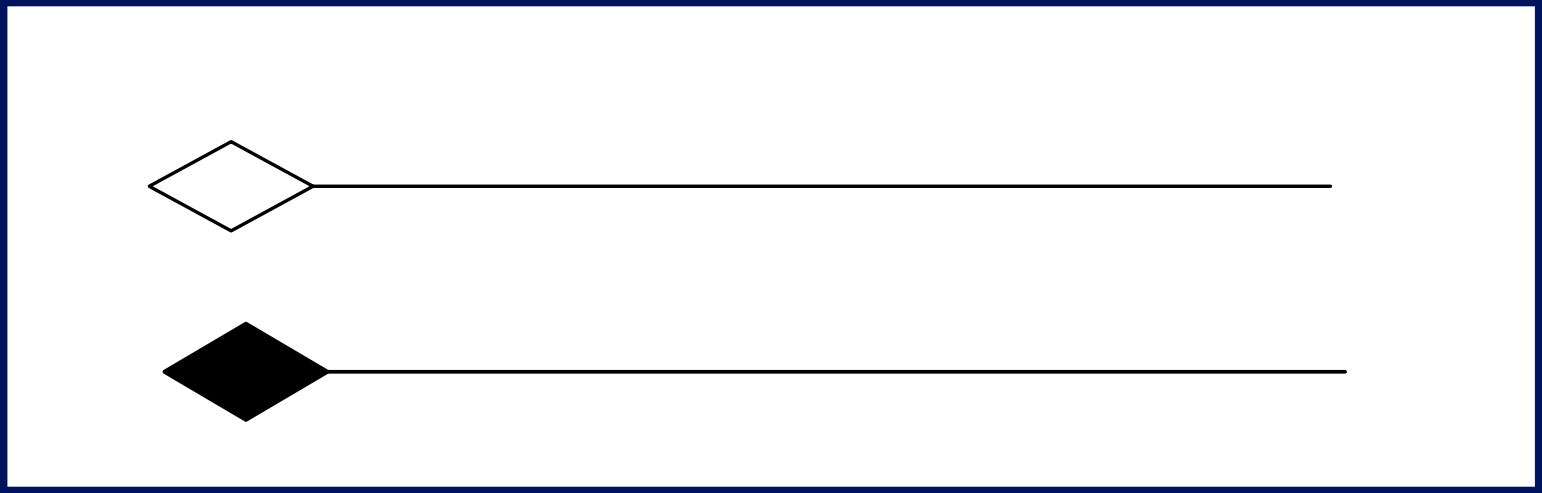
- How many object can be associated
- 1 = exactly one
- 0 .. 1 = optional (zero or one)
- 1 .. N = at least one
- * = 0 .. N = any number
- N
  - For example 4, for 4 wheels in car
- m .. n

Association example



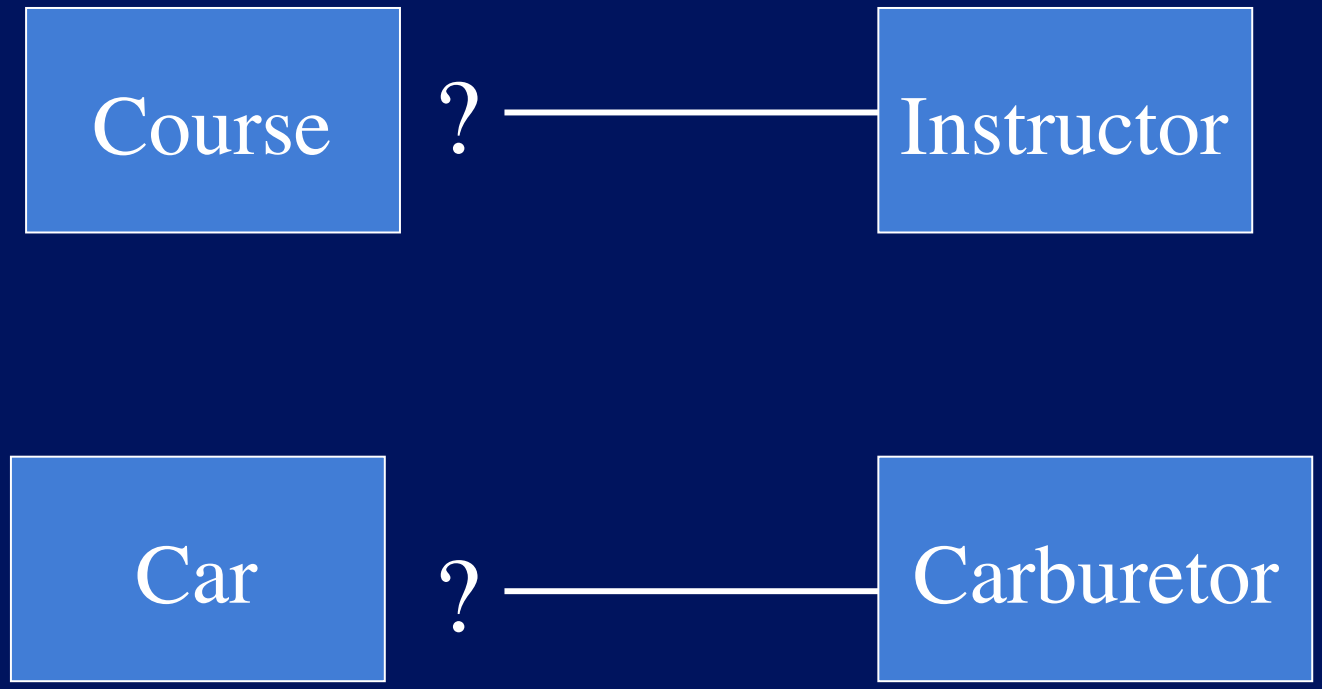
Two Special Associations

- Aggregation = grouping (e.g., “by reference”)

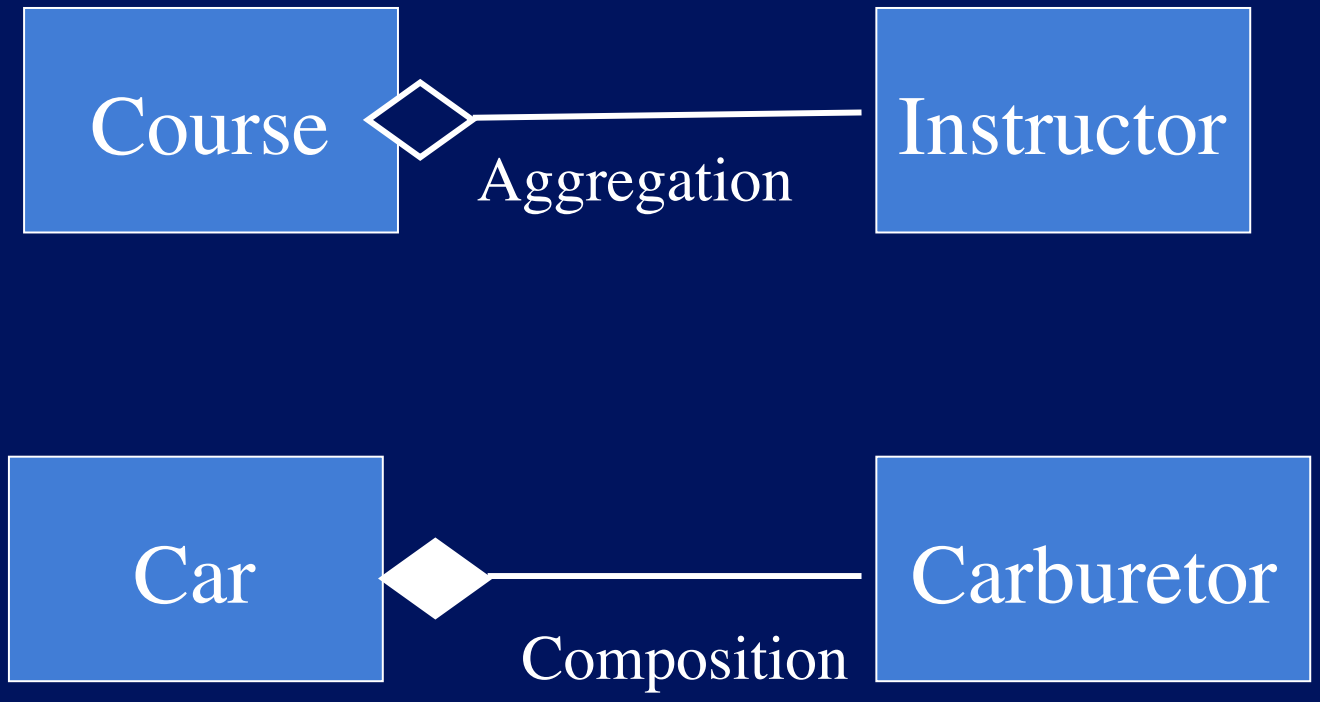


- Composition = is made of (e.g., “by value”)

Composition versus Aggregation



Composition versus Aggregation



Composition versus Aggregation

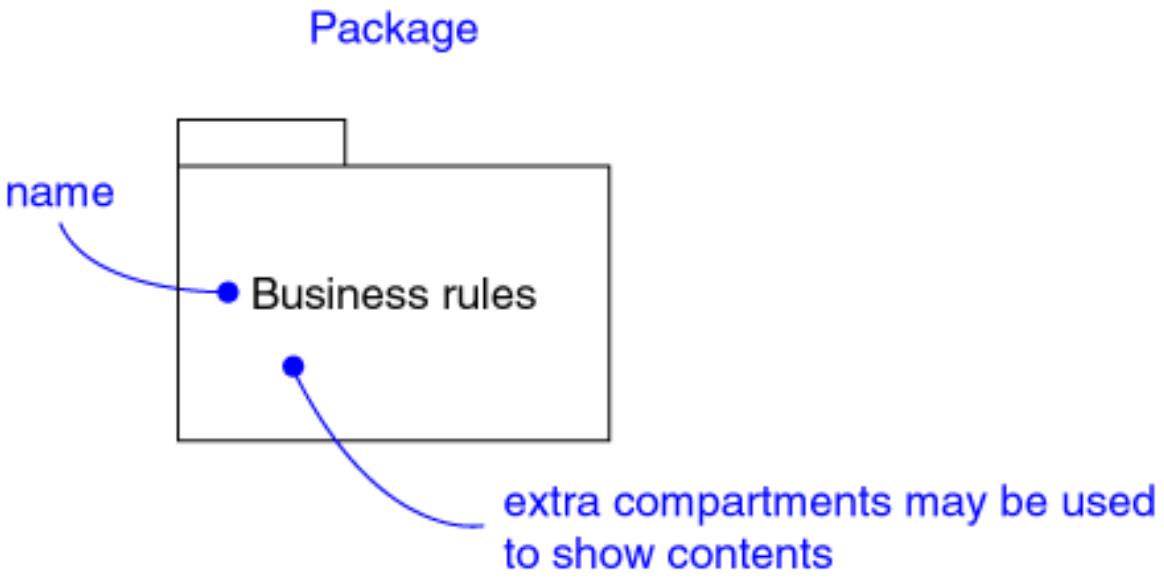
- Assume A associated with B
- If I destroy an object A, is the associated B also destroyed?
  - Yes? you probably have a composition
- If an object A1 is associated with object B1, can the same object B1 be also associated with another object A2
  - Yes? you probably have an aggregation

Composition vs. Aggregation

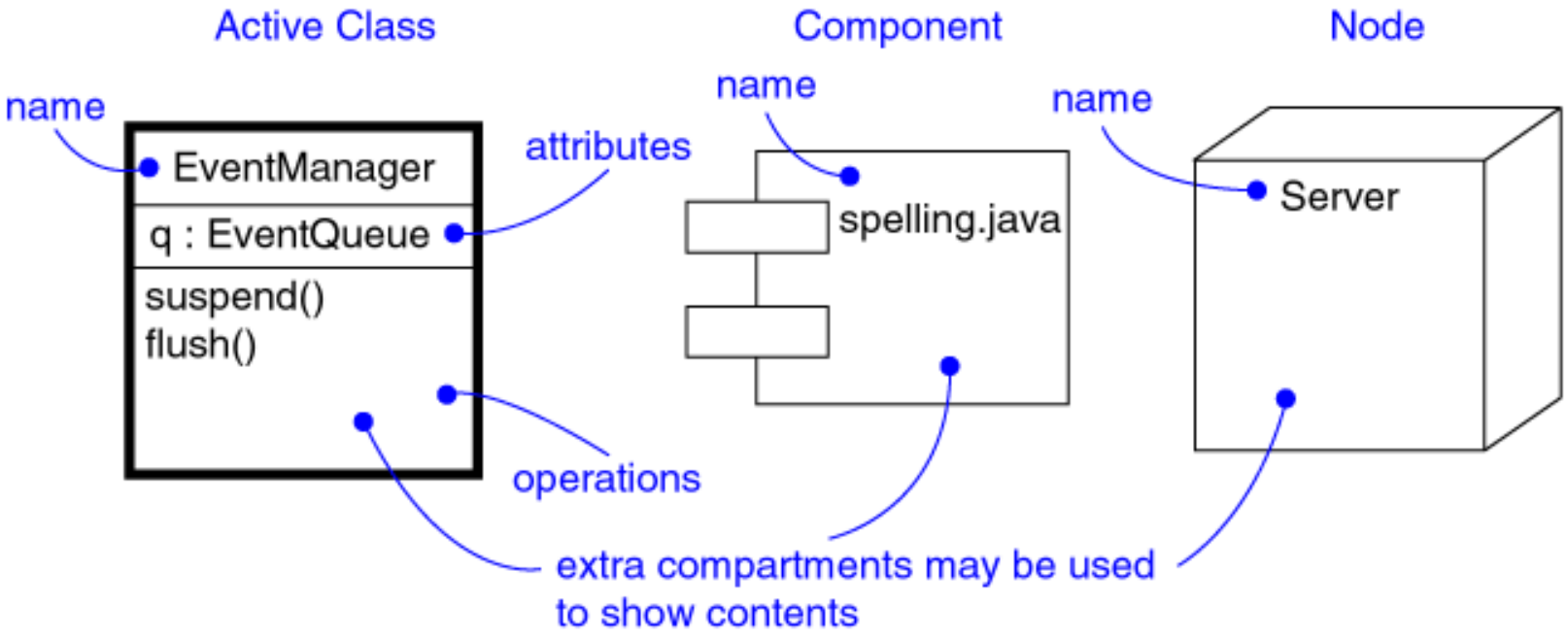
- The whole of a composition must have a multiplicity of 0..1 or 1, indicating that a part must be for only one whole.
  - The whole of an aggregation may have any multiplicity.

- Example:
  - represent real-world whole-part relationships,
    - e.g., an engine is part of a car,
    - → the composition relationship is most appropriate.
  - represent database relationship,
    - e.g., car model engine ENG01 is part of a car model CM01,
    - → an aggregation relationship is best, (as the engine ENG01 may be also part of a different car model)

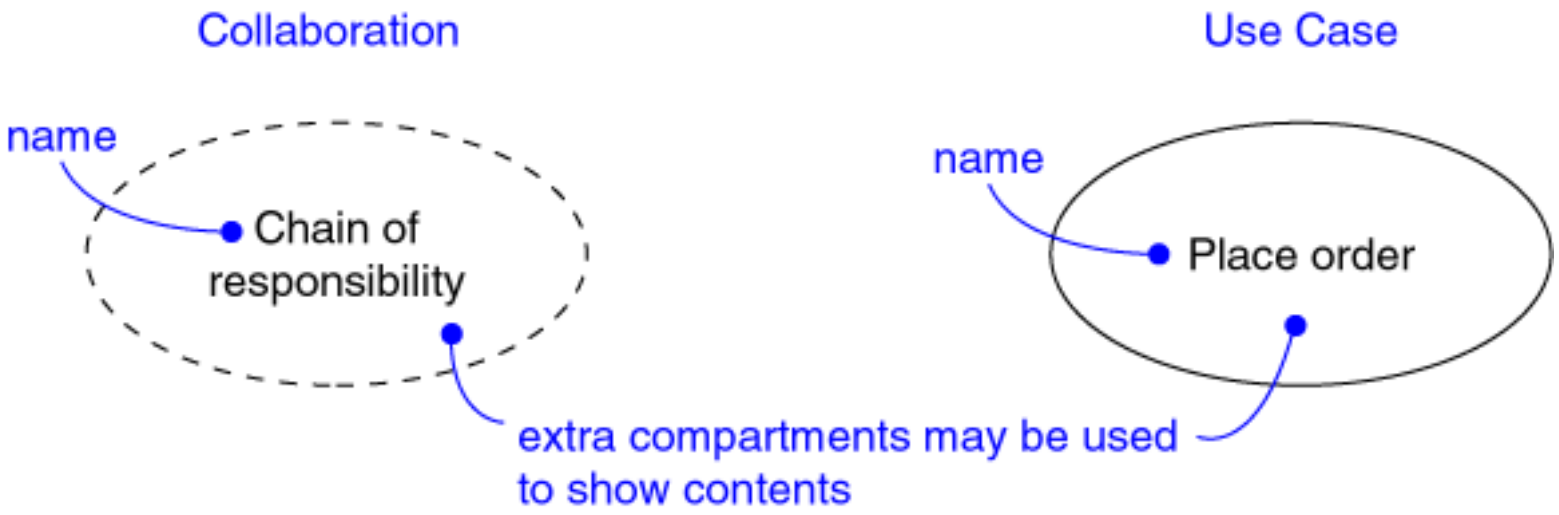
Packages


Package
name
Business rules
extra compartments may be used
to show contents

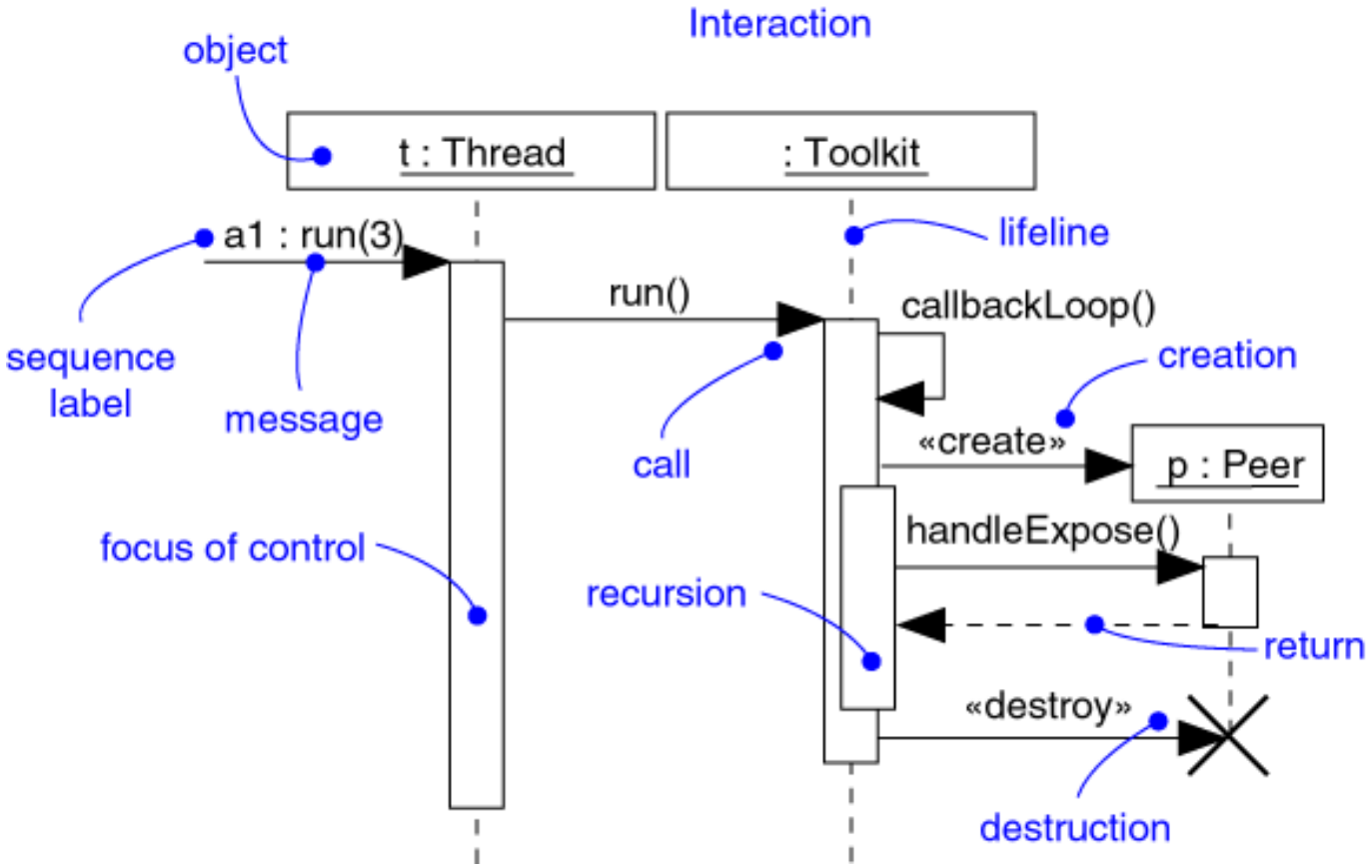
Other major modeling elements



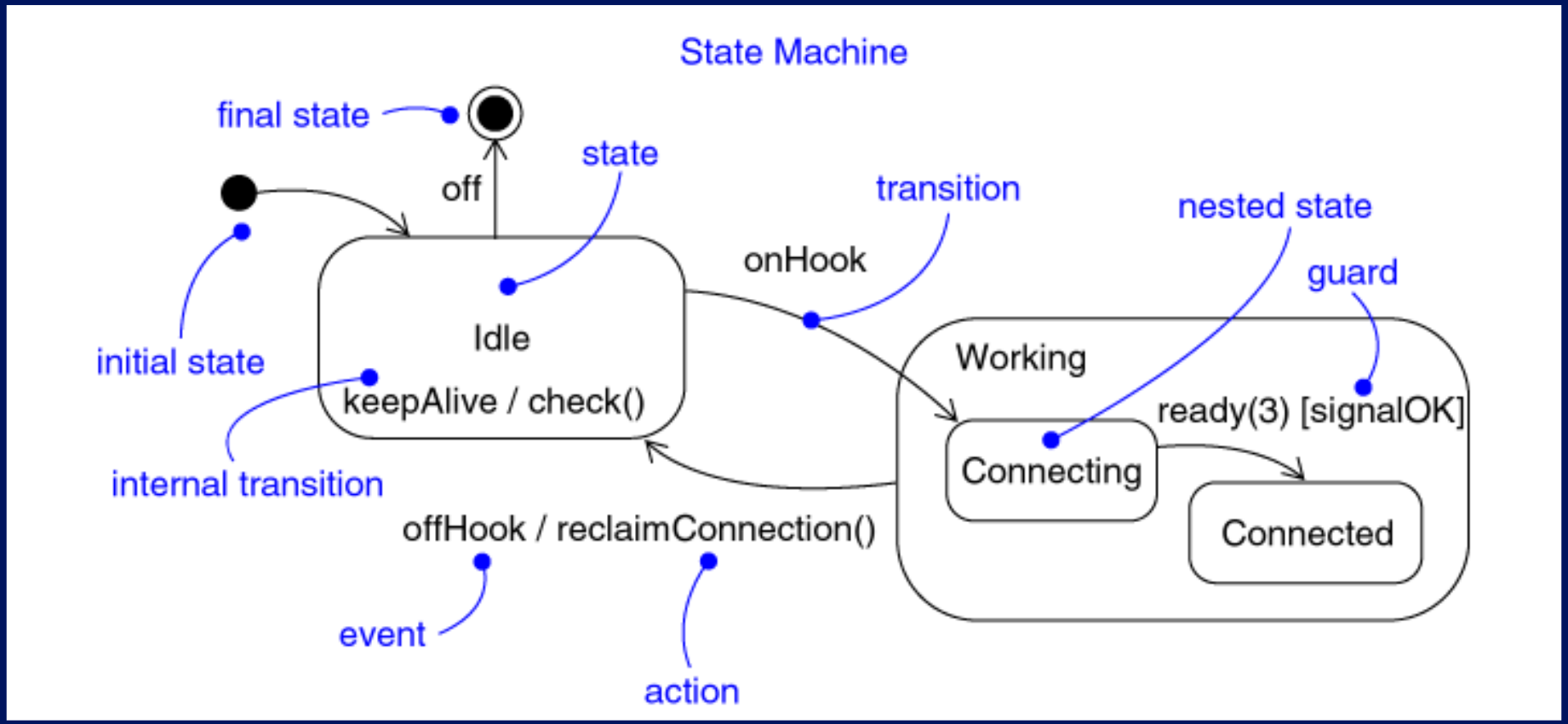
Collaborations and Use cases



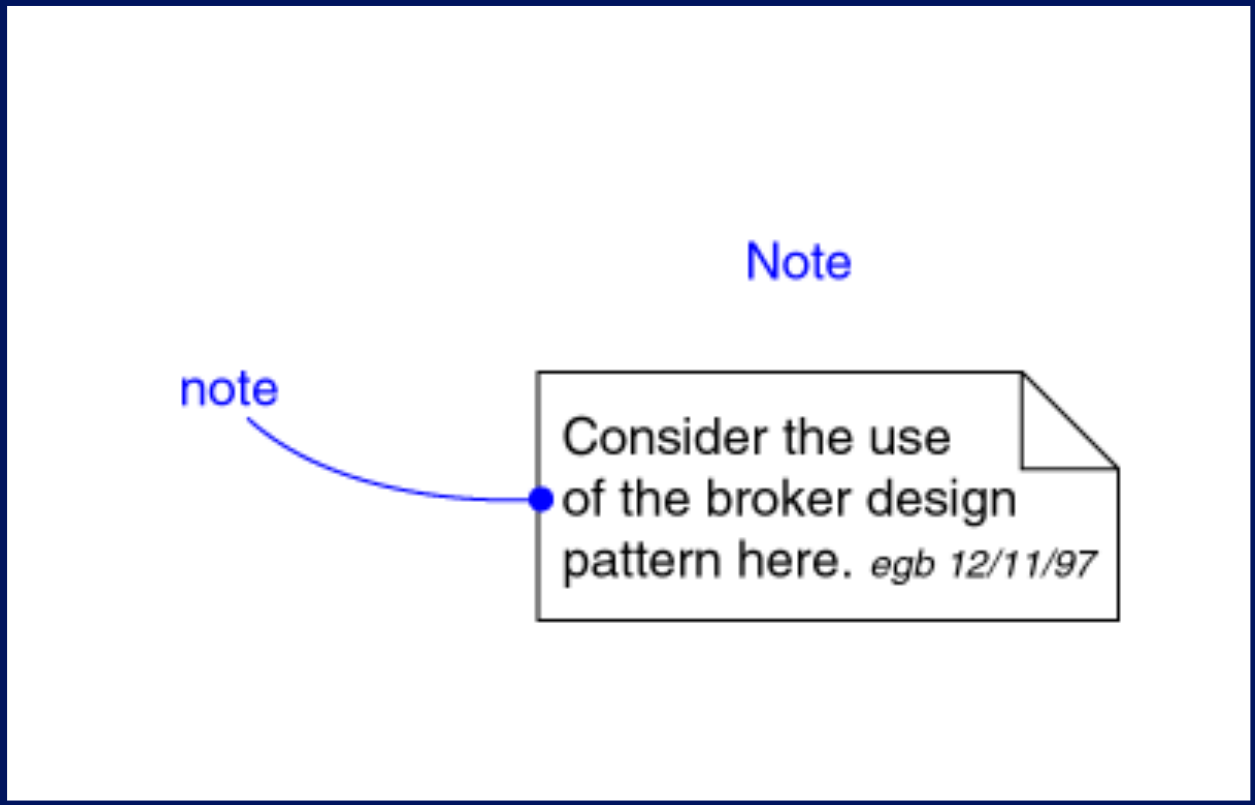
Sequences


Interaction
object
t : Thread
: Toolkit
lifeline
a1 : run(3)
run()
callbackLoop()
sequence label
message
call
creation
«create»
p : Peer
handleExpose()
focus of control
recursion
return
«destroy»
destruction

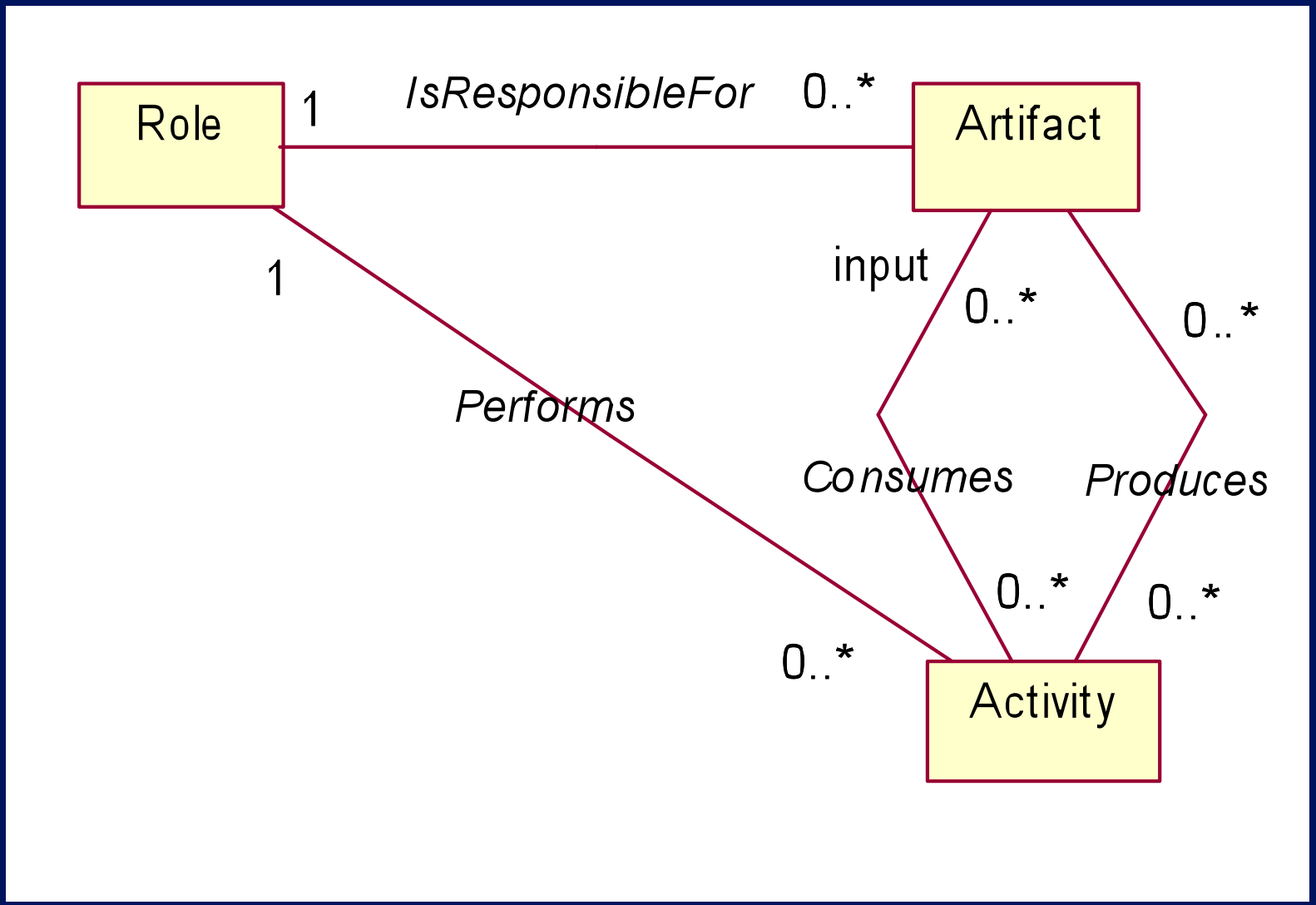
States

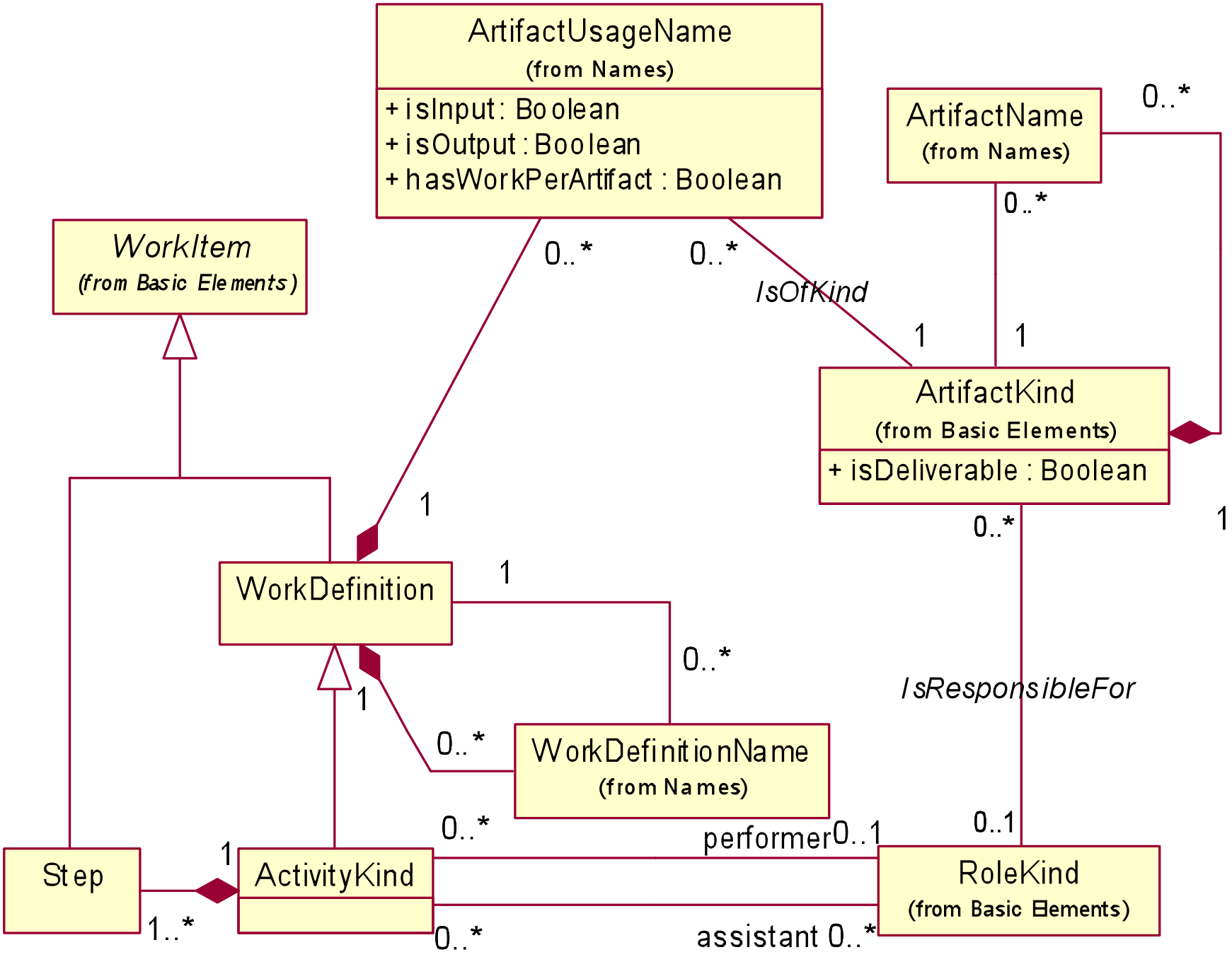

State Machine
final state
off
state
transition
nested state
guard
onHook
initial state
Idle
Working
keepAlive / check()
ready(3) [signalOK]
internal transition
Connecting
Connected
offHook / reclaimConnection()
event
action

Notes

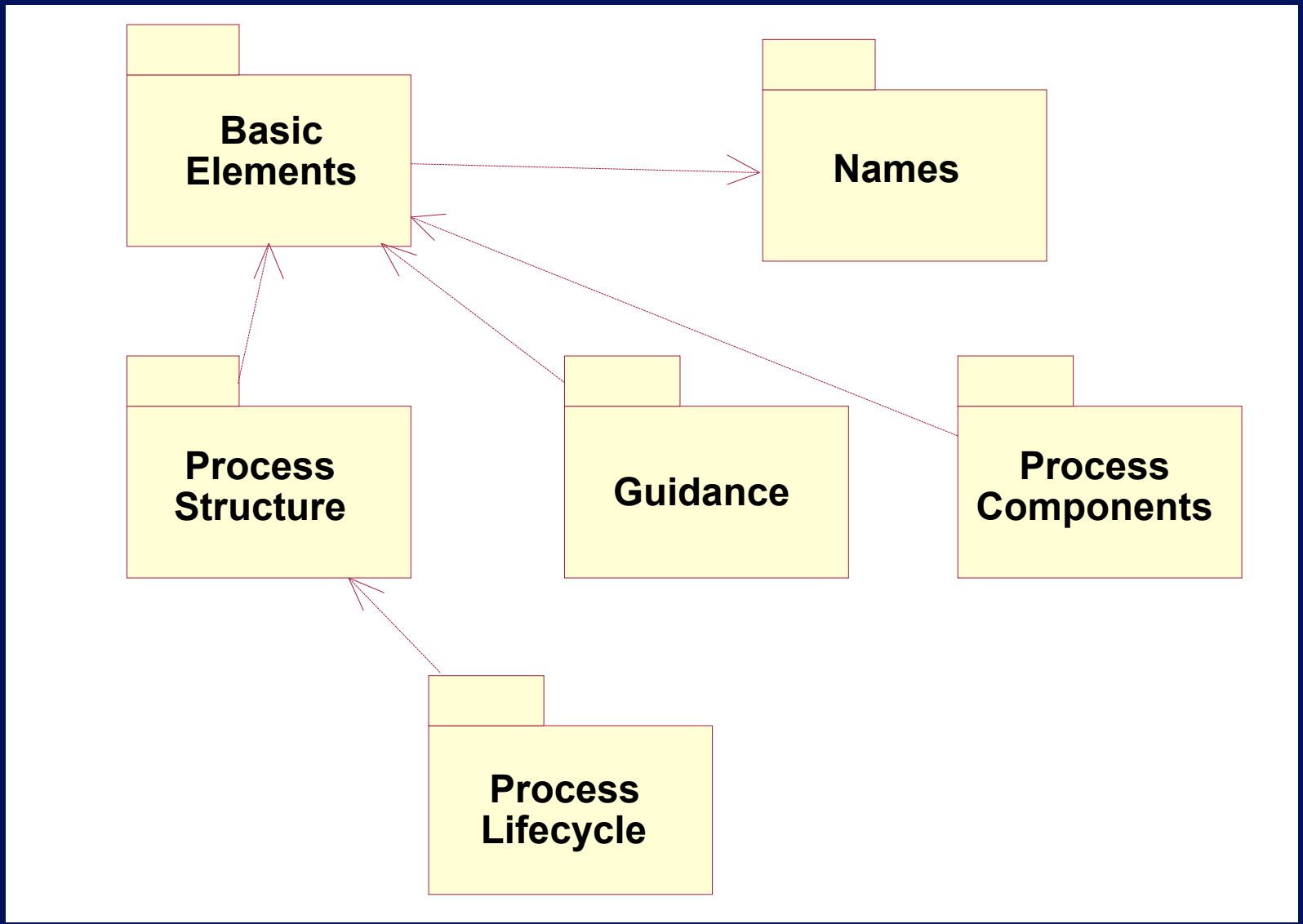

Note
note
Consider the use of the broker design pattern here. egb 12/11/97

Class Diagram

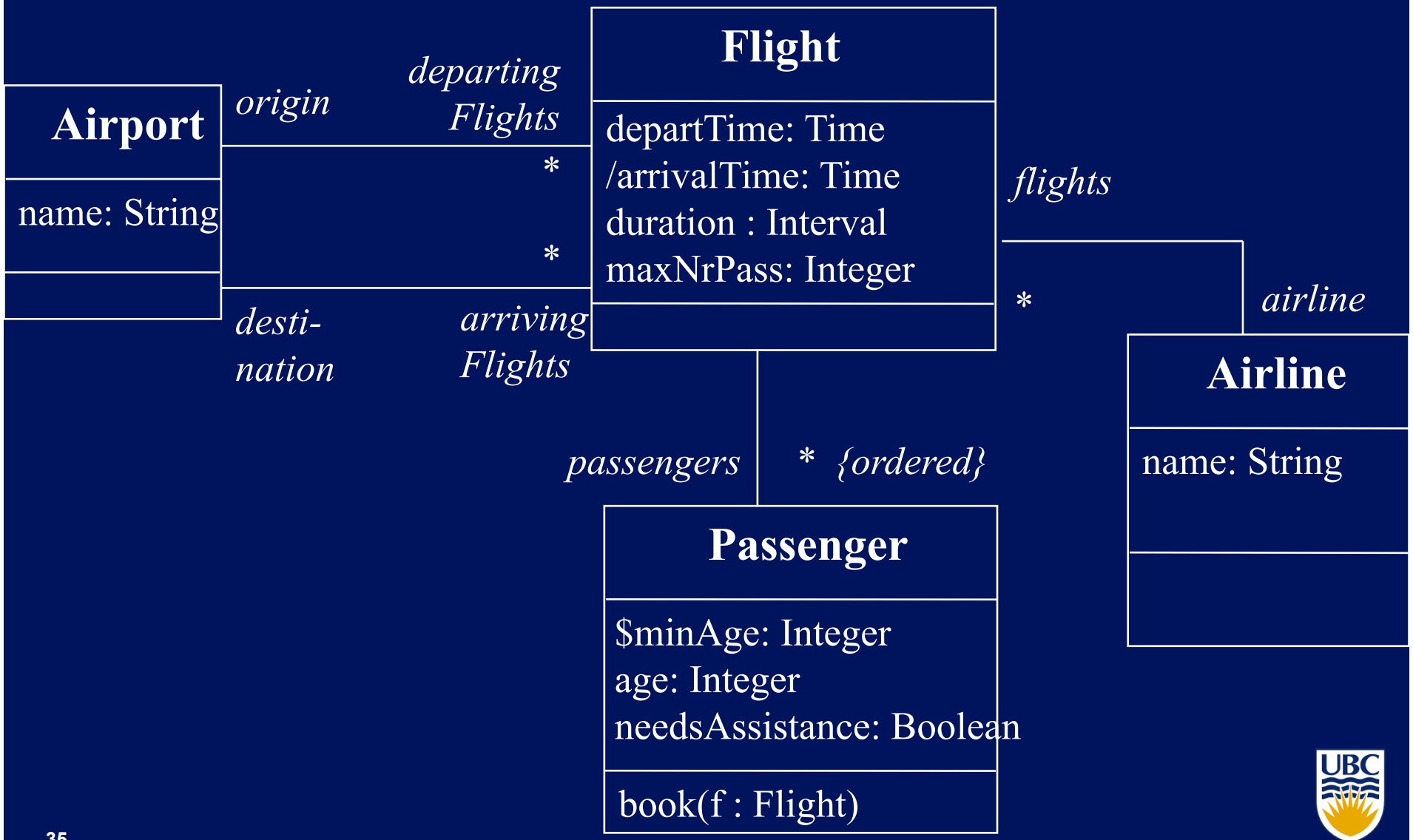


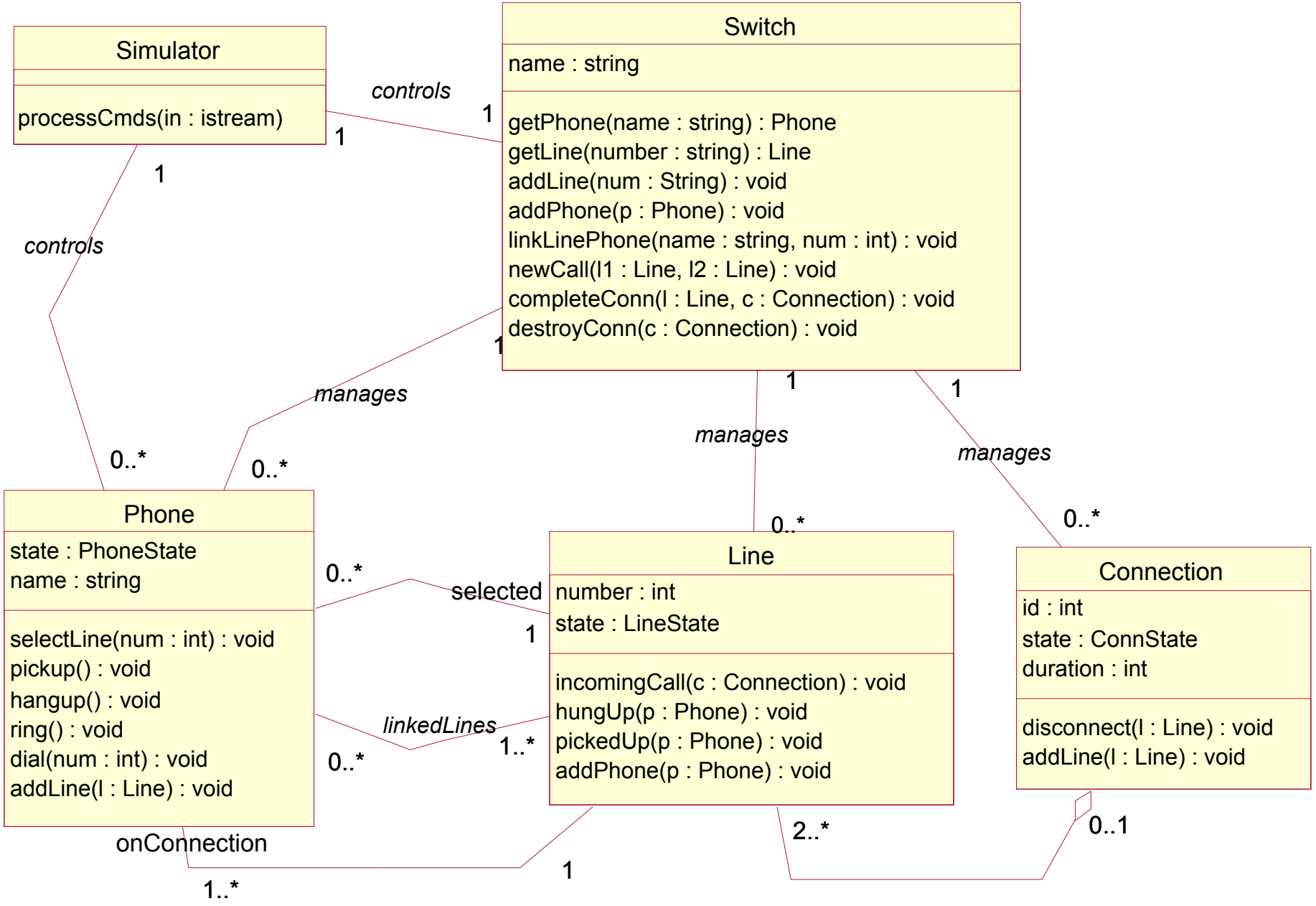

ArtifactUsageName
(from Names)
+ isInput: Boolean
+ isOutput: Boolean
+ hasWorkPerArtifact: Boolean
ArtifactName
(from Names)
0..*
WorkItem
(from Basic Elements)
0..*
0..*
IsOfKind
1
1
ArtifactKind
(from Basic Elements)
+ isDeliverable: Boolean
1
0..*
1
WorkDefinition
1
0..*
IsResponsibleFor
1
0..*
WorkDefinitionName
(from Names)
0..*
performer
0..1
0..1
Step
1
ActivityKind
RoleKind
(from Basic Elements)
1..*
0..*
assistant
0..*

Class Diagram (3)

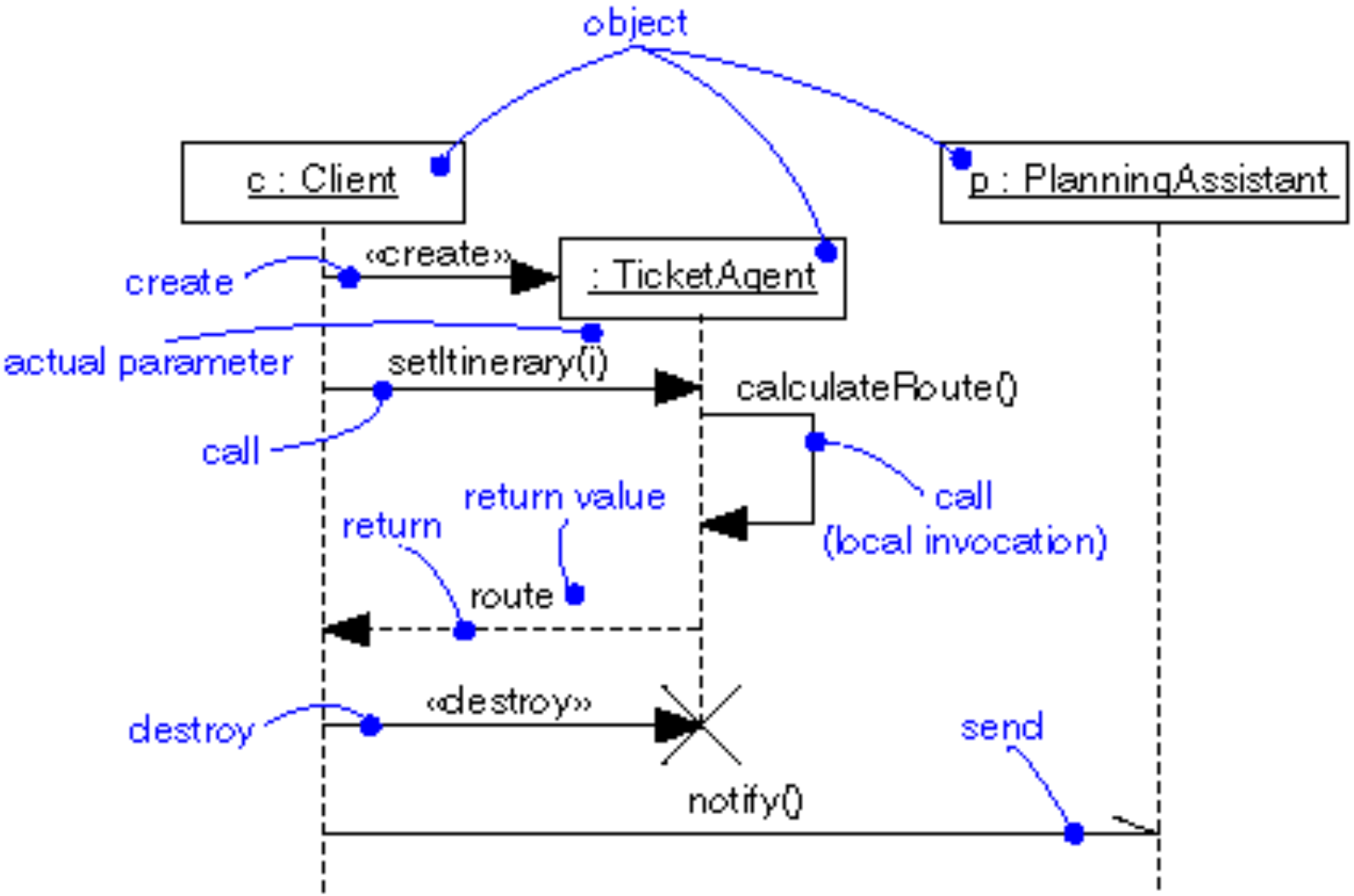


More class diagrams

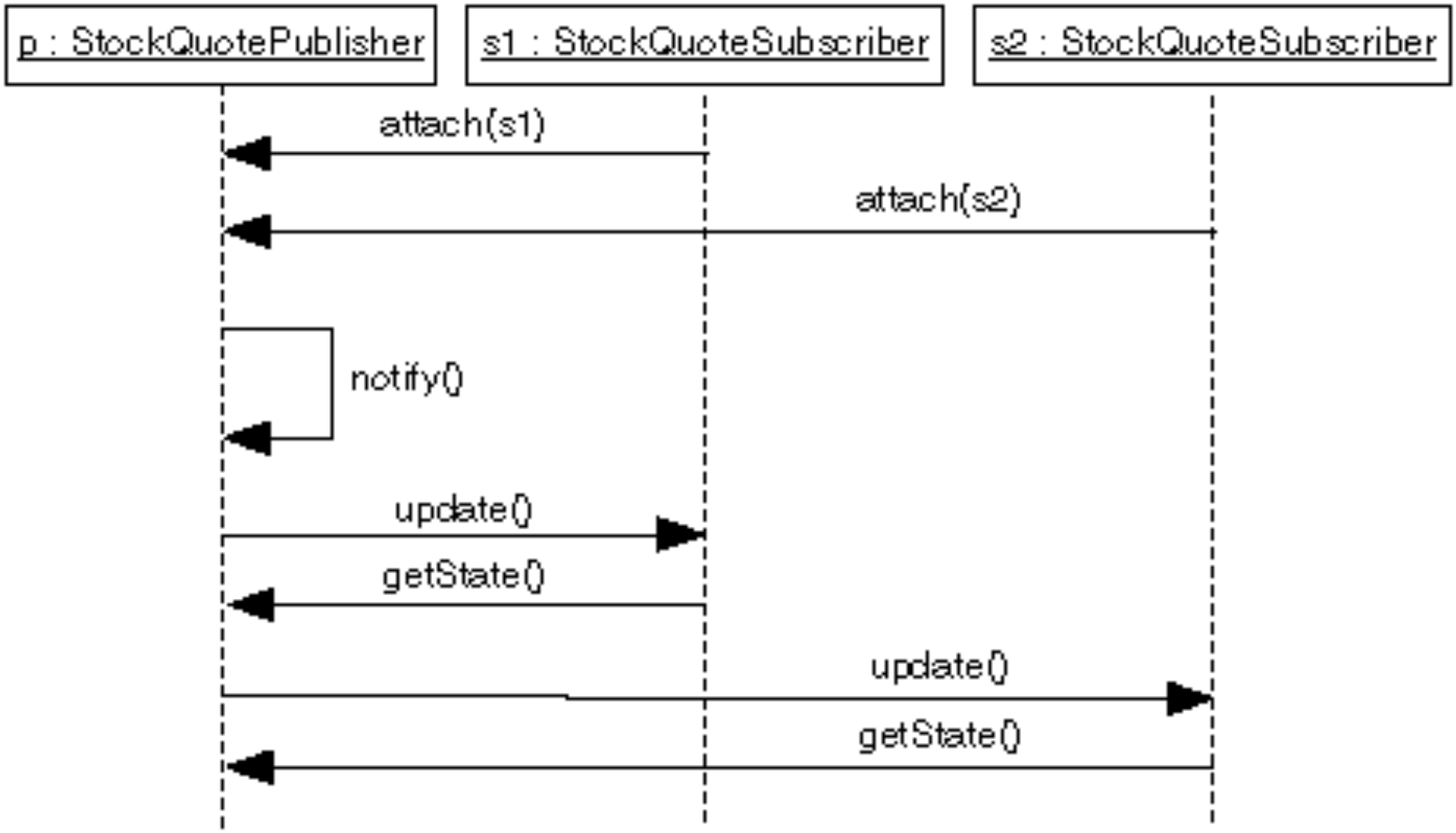



Simulator
processCmds(in : istream)
Switch
name : string
getPhone(name : string) : Phone
getLine(number : string) : Line
addLine(num : String) : void
addPhone(p : Phone) : void
linkLinePhone(name : string, num : int) : void
newCall(l1 : Line, l2 : Line) : void
completeConn(l : Line, c : Connection) : void
destroyConn(c : Connection) : void
controls
1
1
1
controls
1
manages
0..*
0..*
1
manages
1
0..*
manages
1
0..*
Phone
state : PhoneState
name : string
selectLine(num : int) : void
pickup() : void
hangup() : void
ring() : void
dial(num : int) : void
addLine(l : Line) : void
0..*
selected
1
linkedLines
0..*
1..*
Line
number : int
state : LineState
incomingCall(c : Connection) : void
hungUp(p : Phone) : void
pickedUp(p : Phone) : void
addPhone(p : Phone) : void
Connection
id : int
state : ConnState
duration : int
disconnect(l : Line) : void
addLine(l : Line) : void
onConnection
1..*
1
2..*
0..1

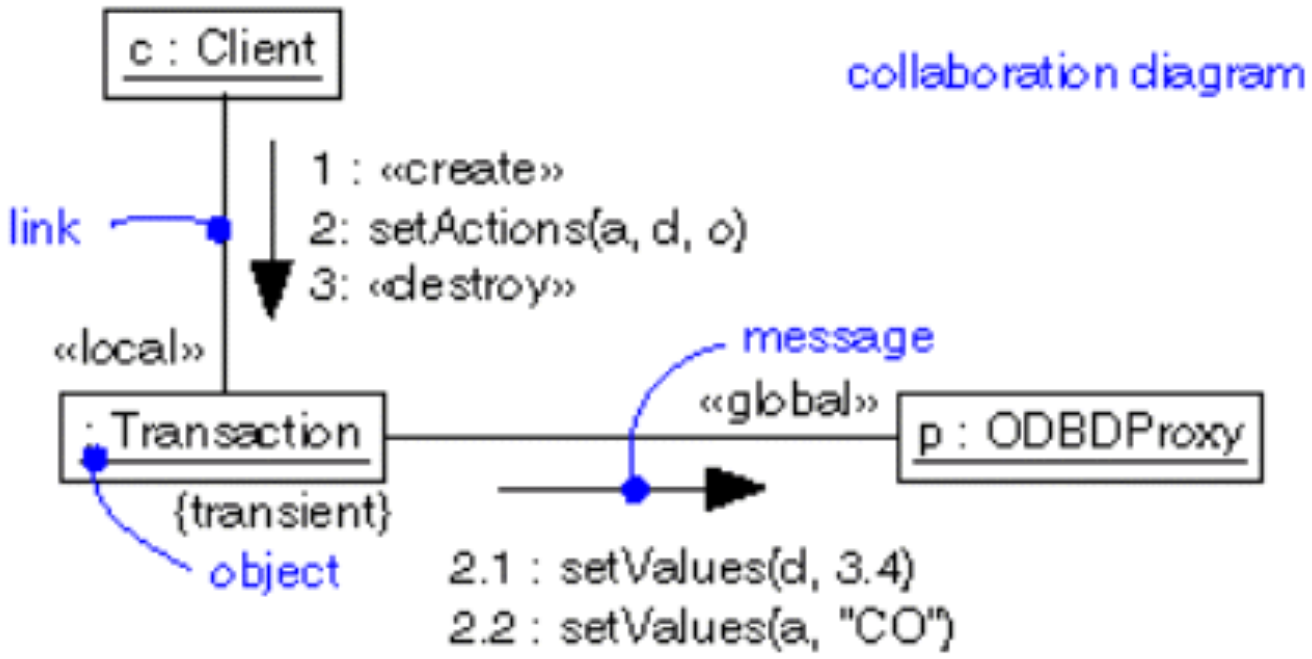
Sequence diagram



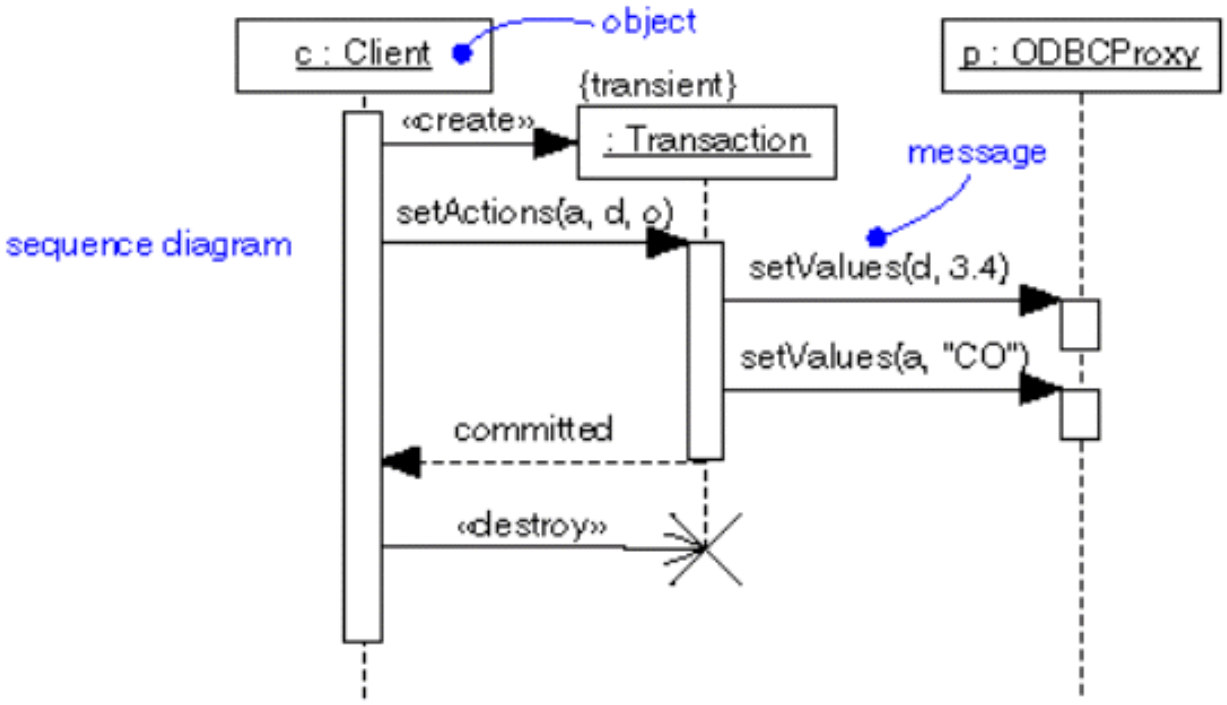
Sequence diagram



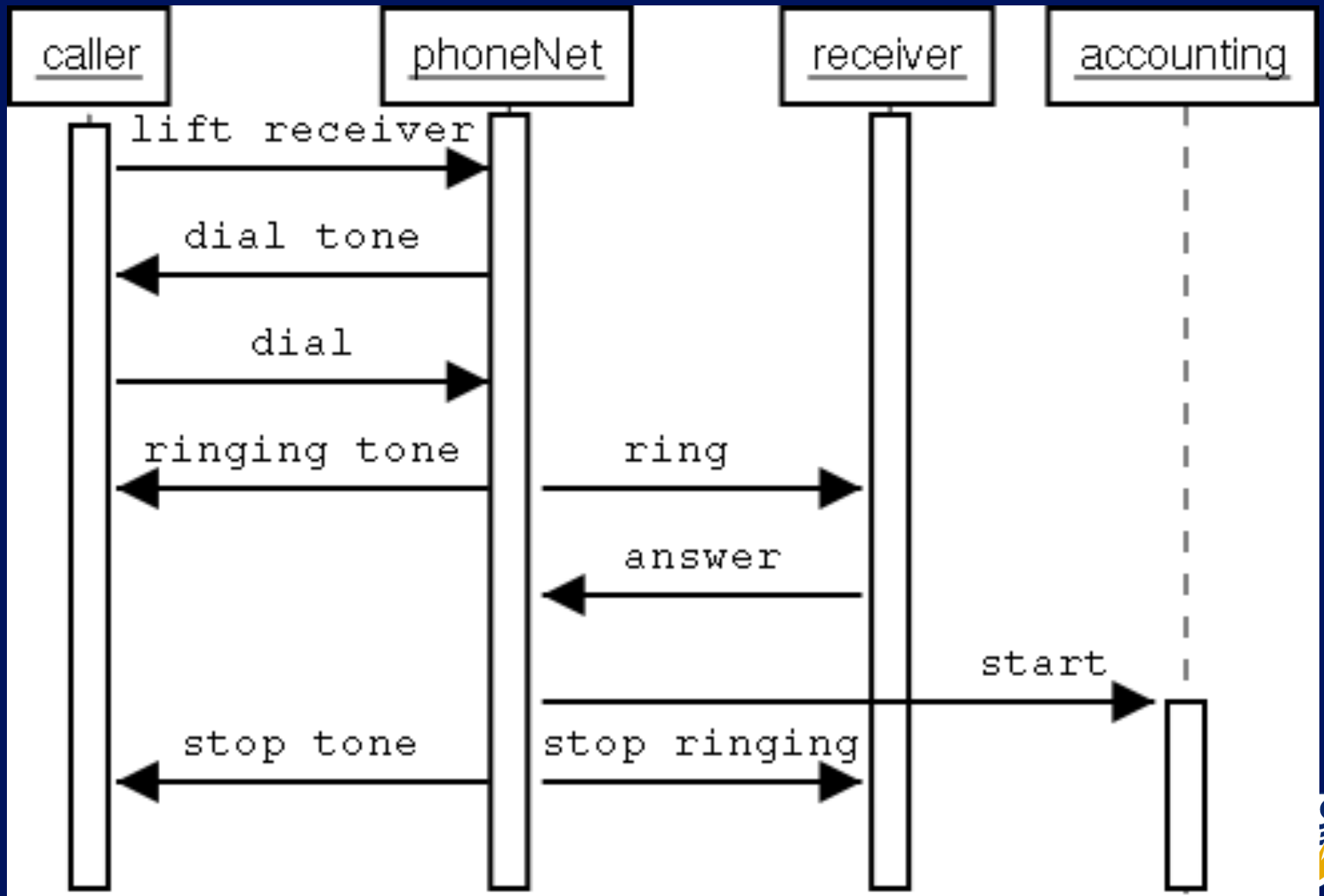
Collaboration diagram



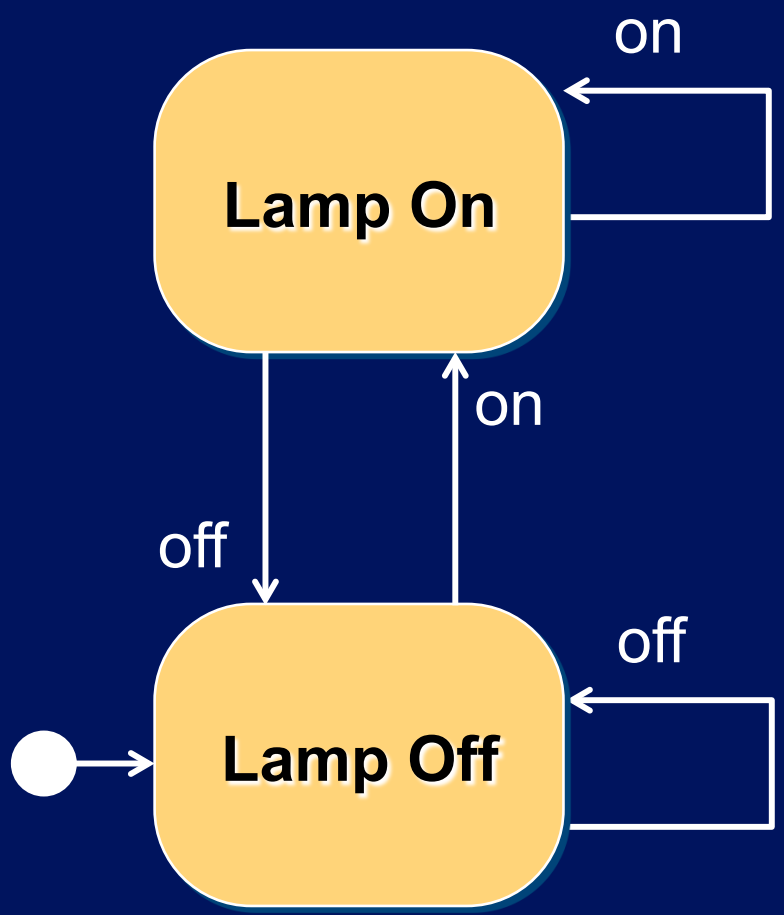
Sequence diagram



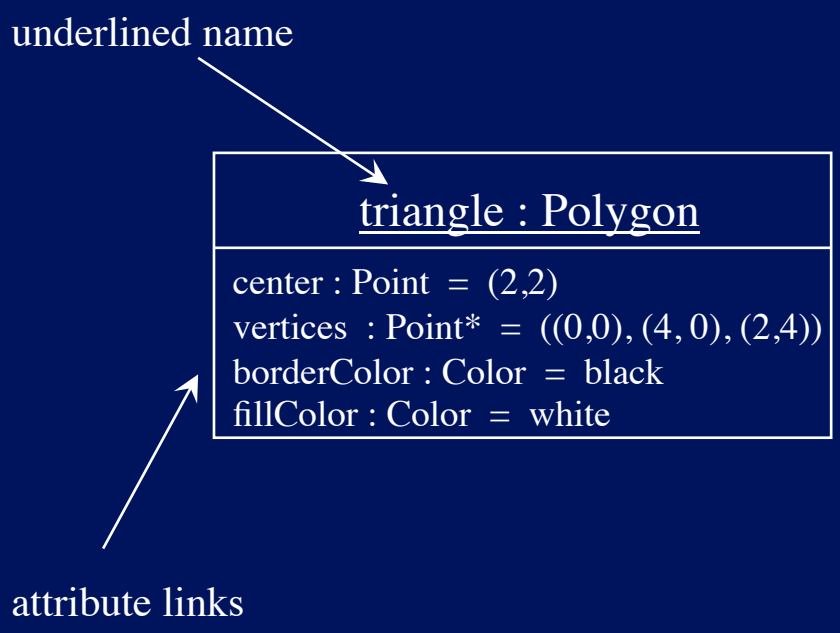
Sequence Diagram for a Phone Call



State diagram



Instances of a Class (Objects)

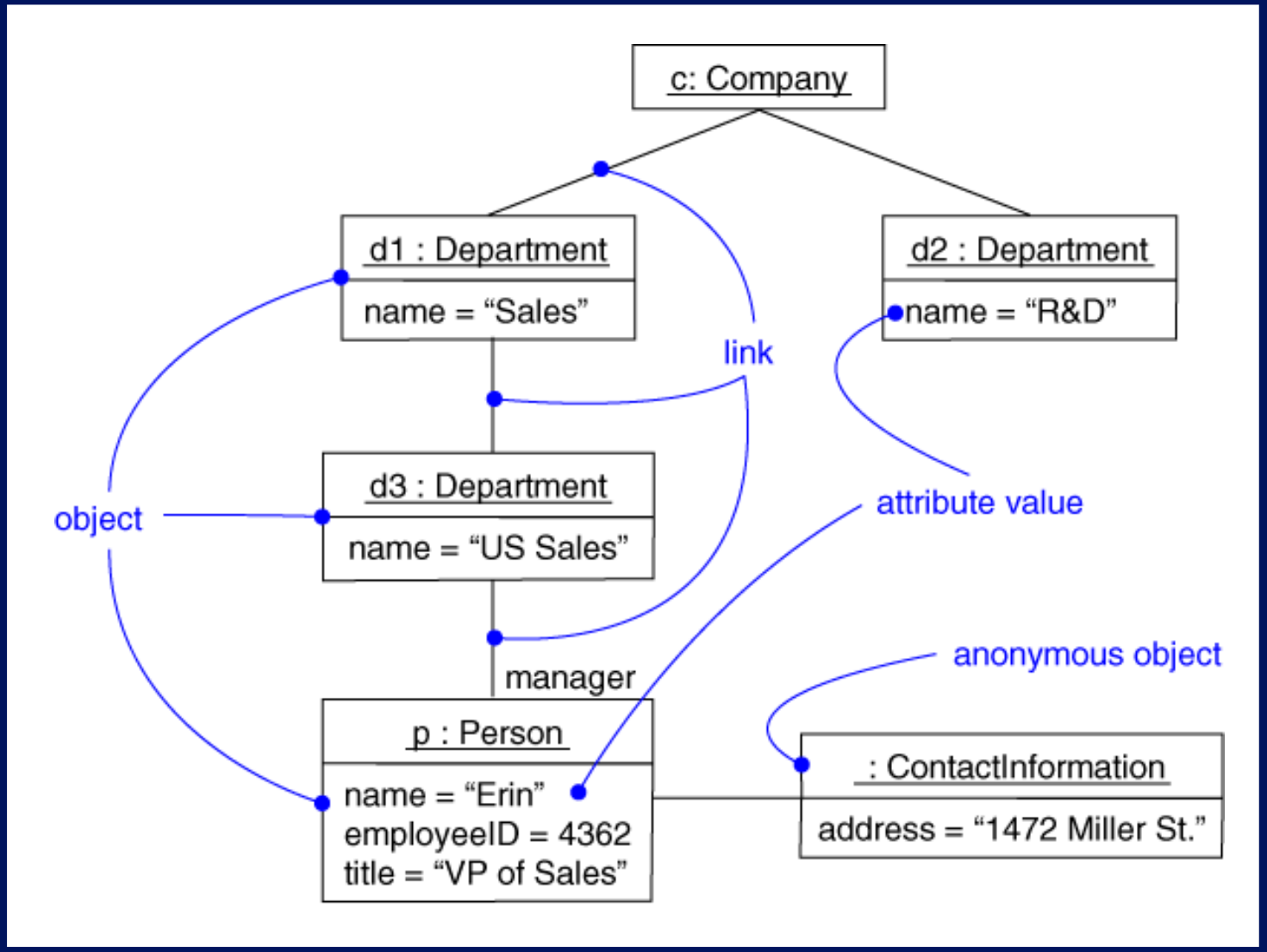


triangle : Polygon

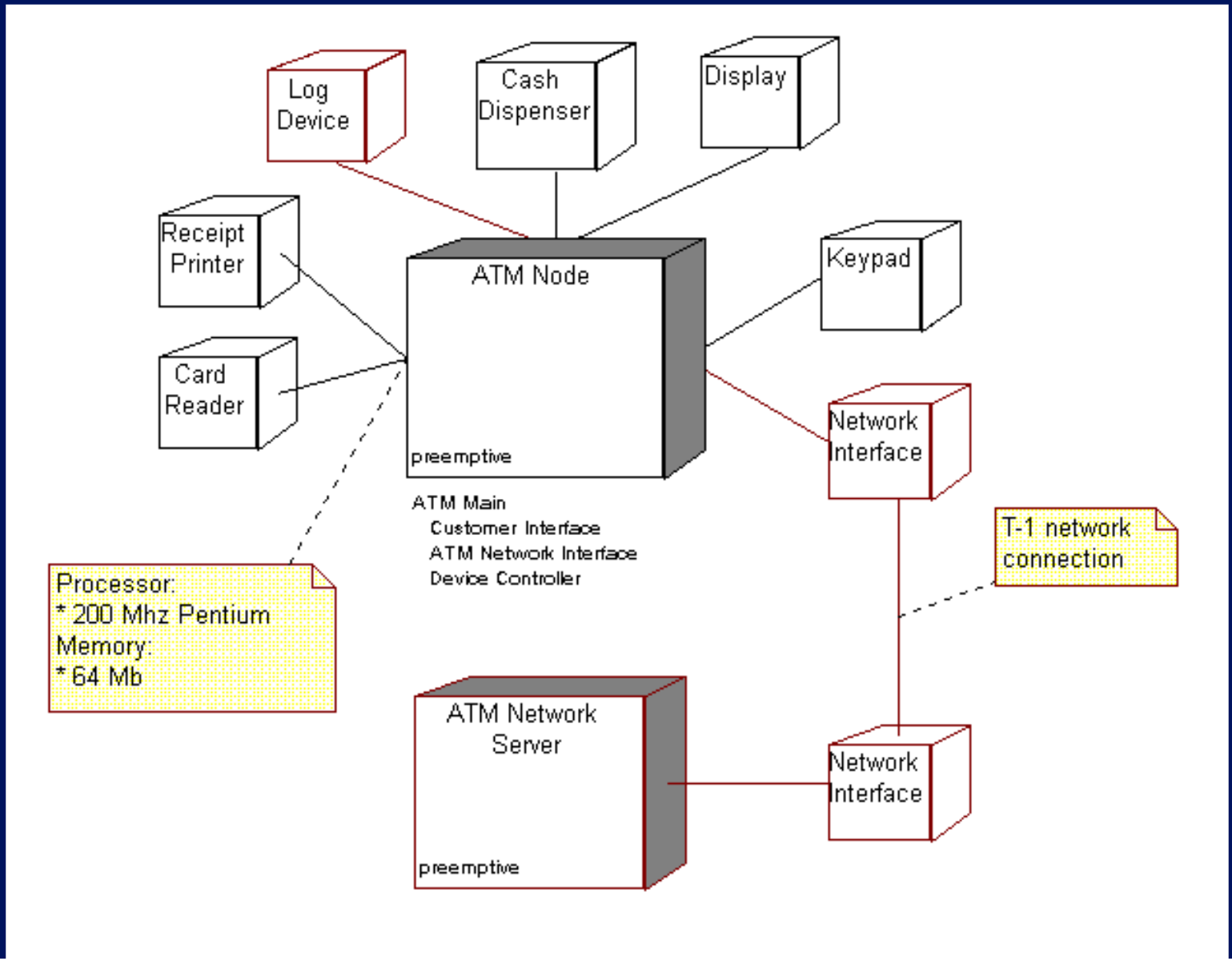
triangle

: Polygon

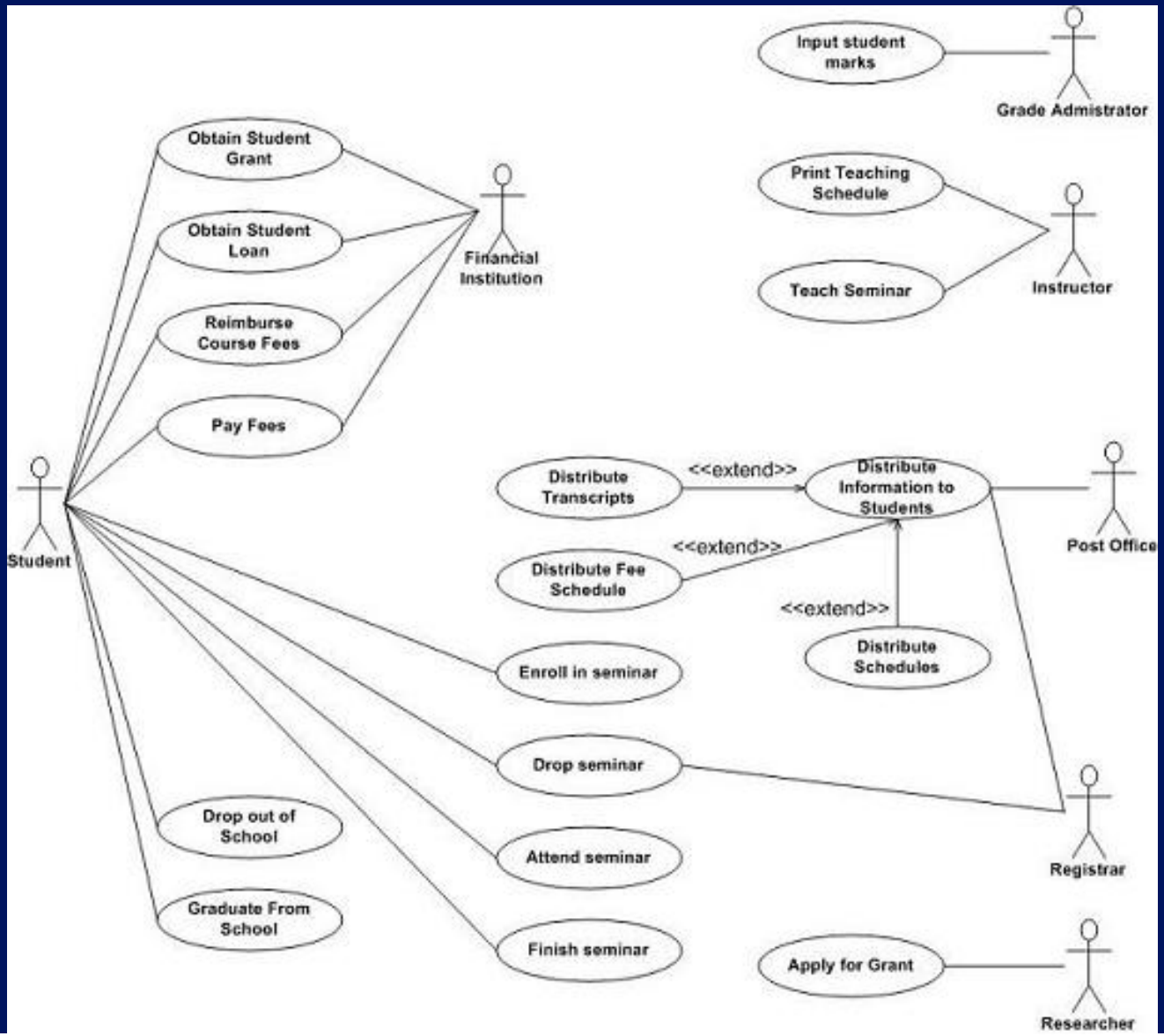
Object Diagram



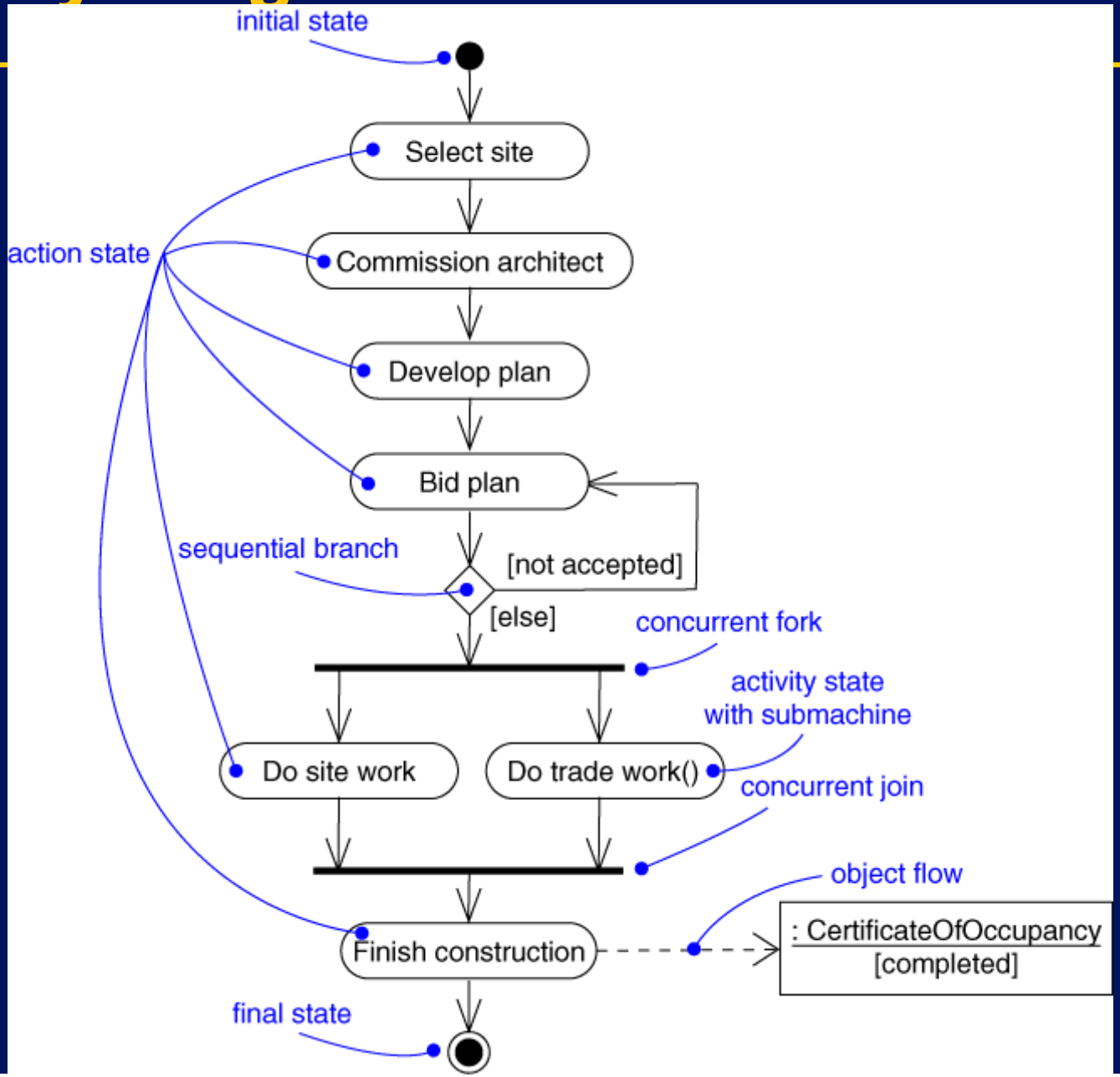
Deployment diagram



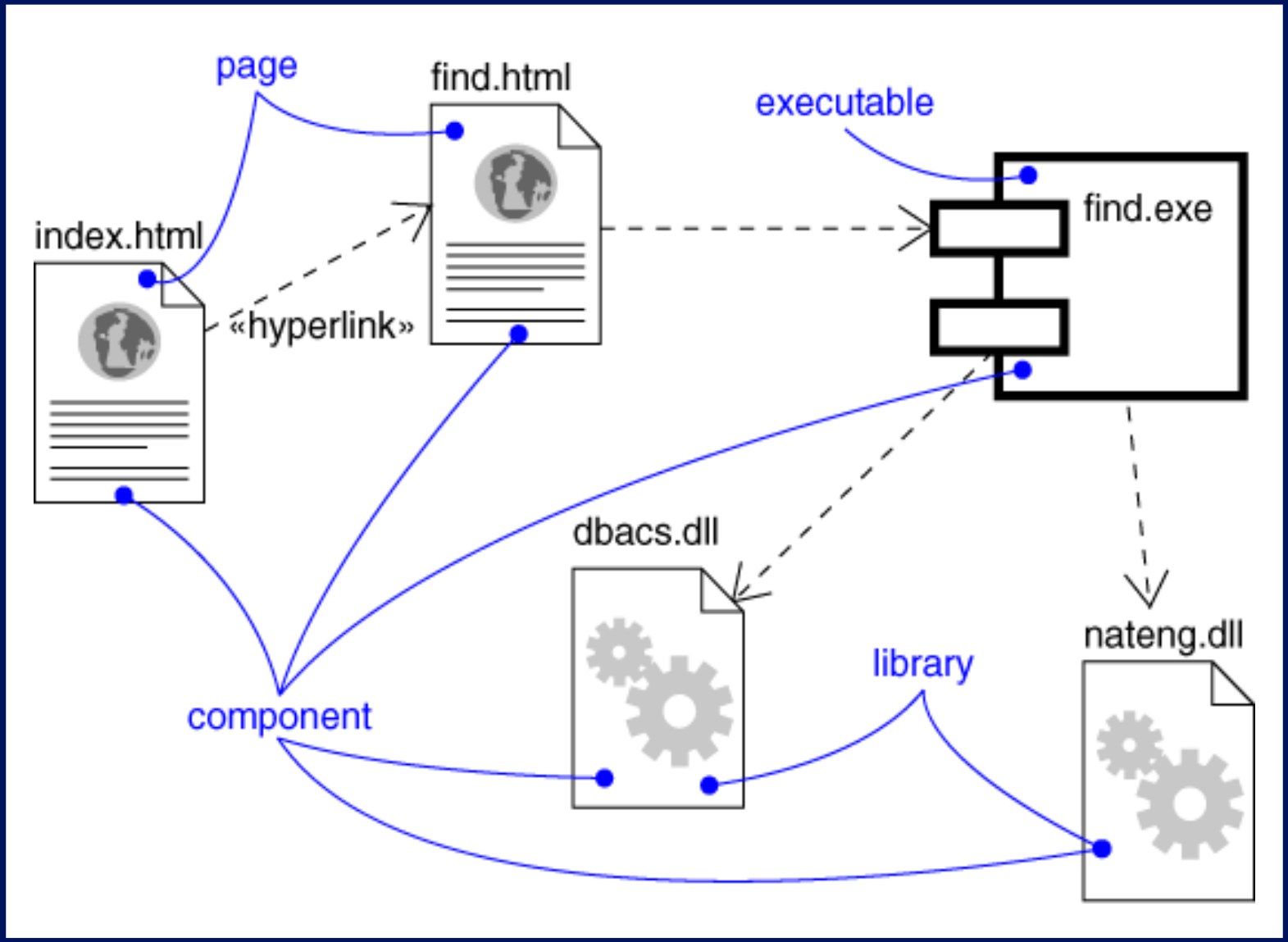
Use-case diagram



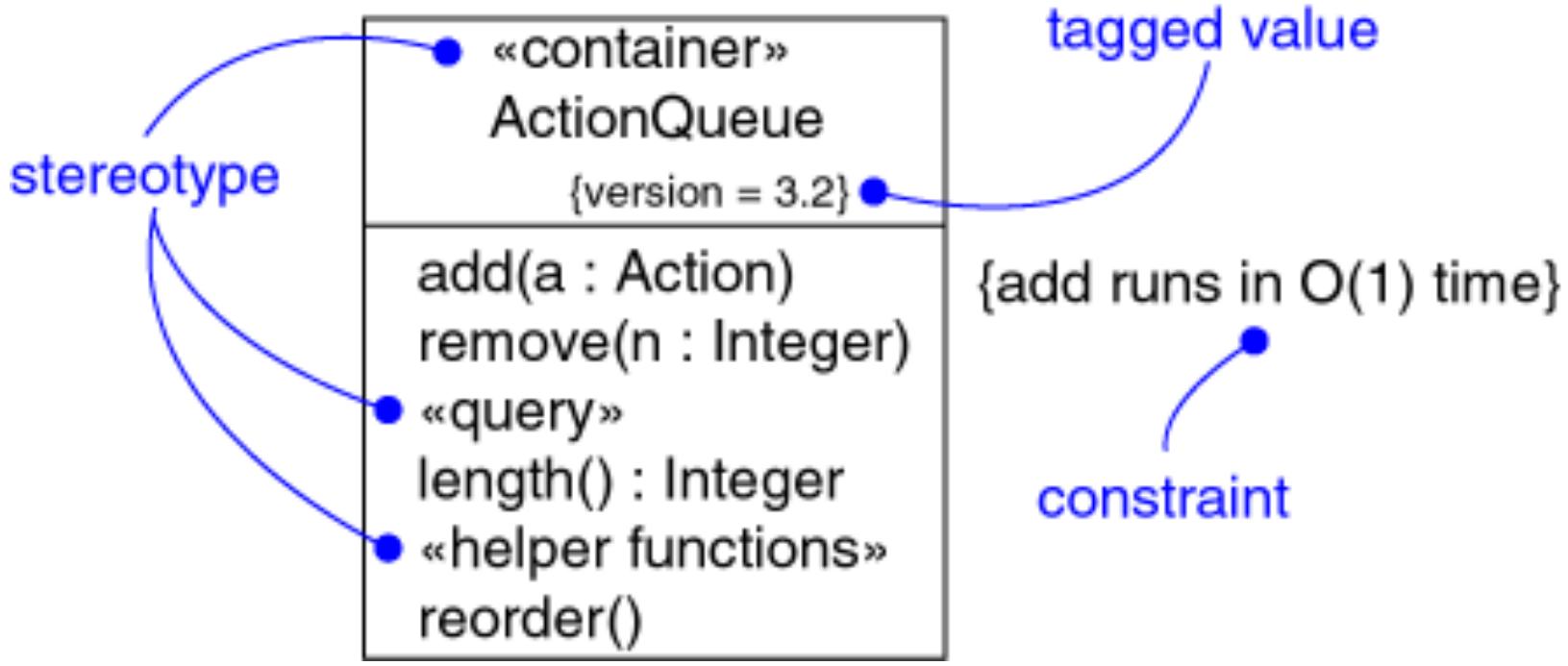
Activity Diagram

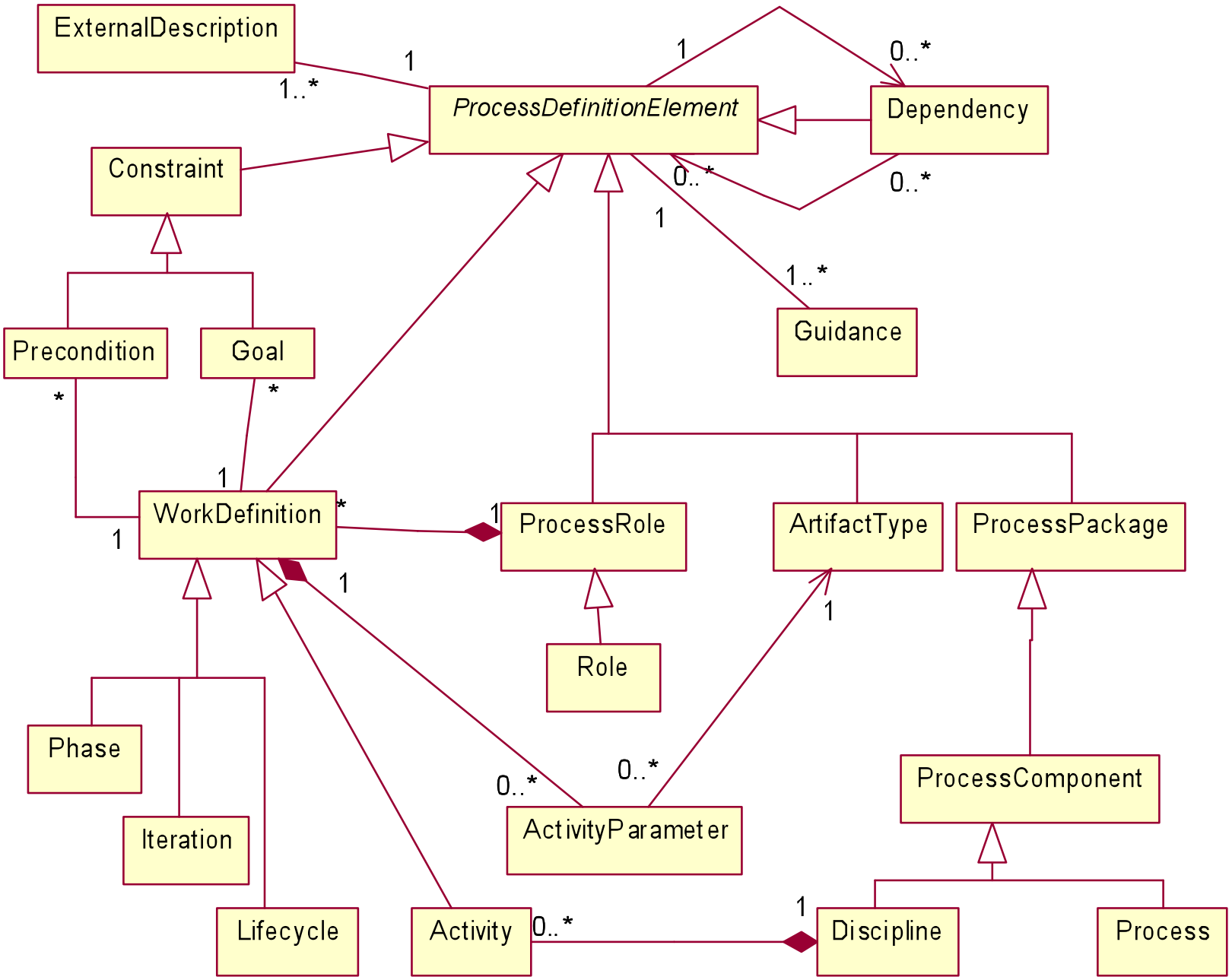


Component Diagram



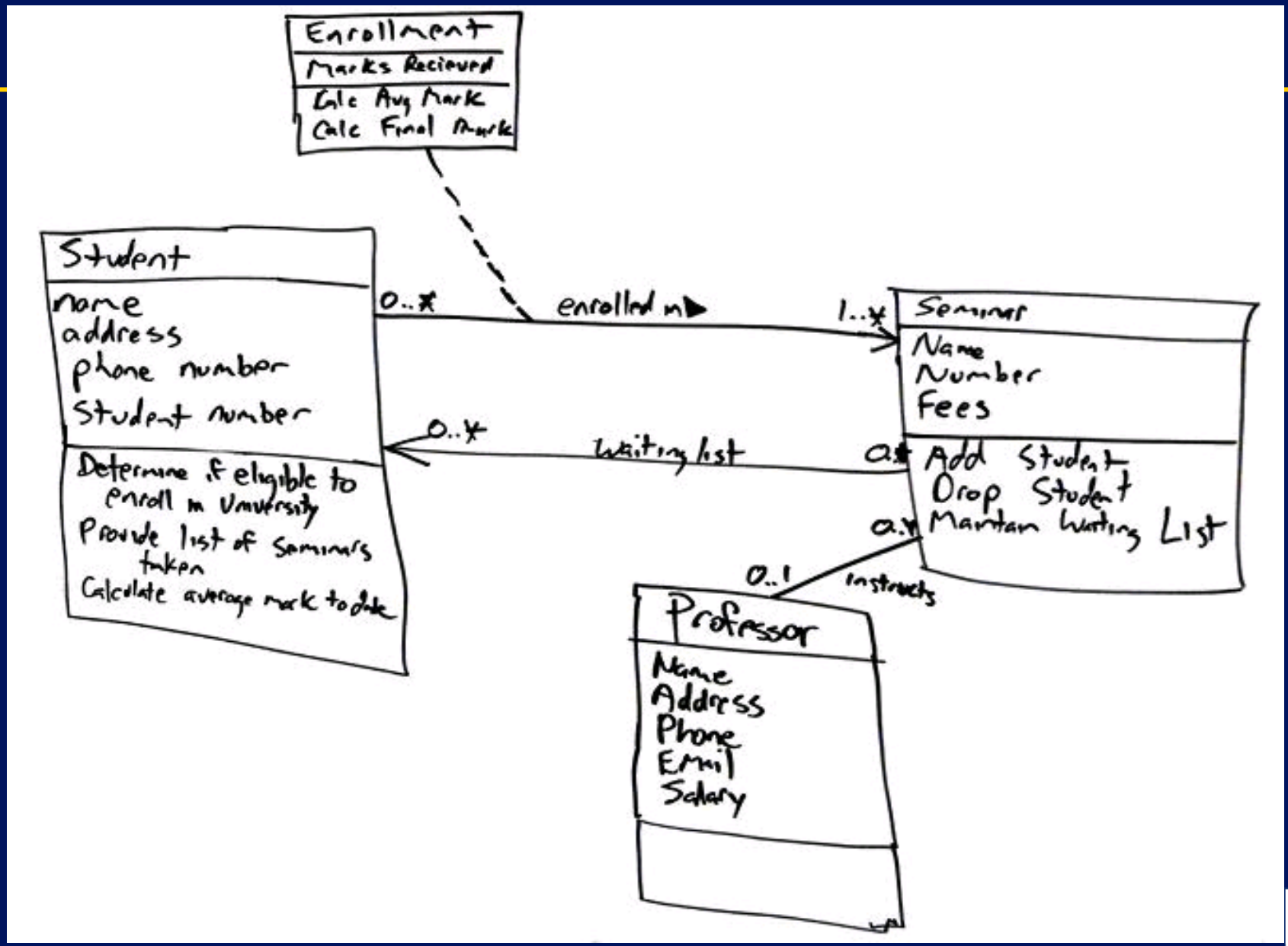
UML Extensions

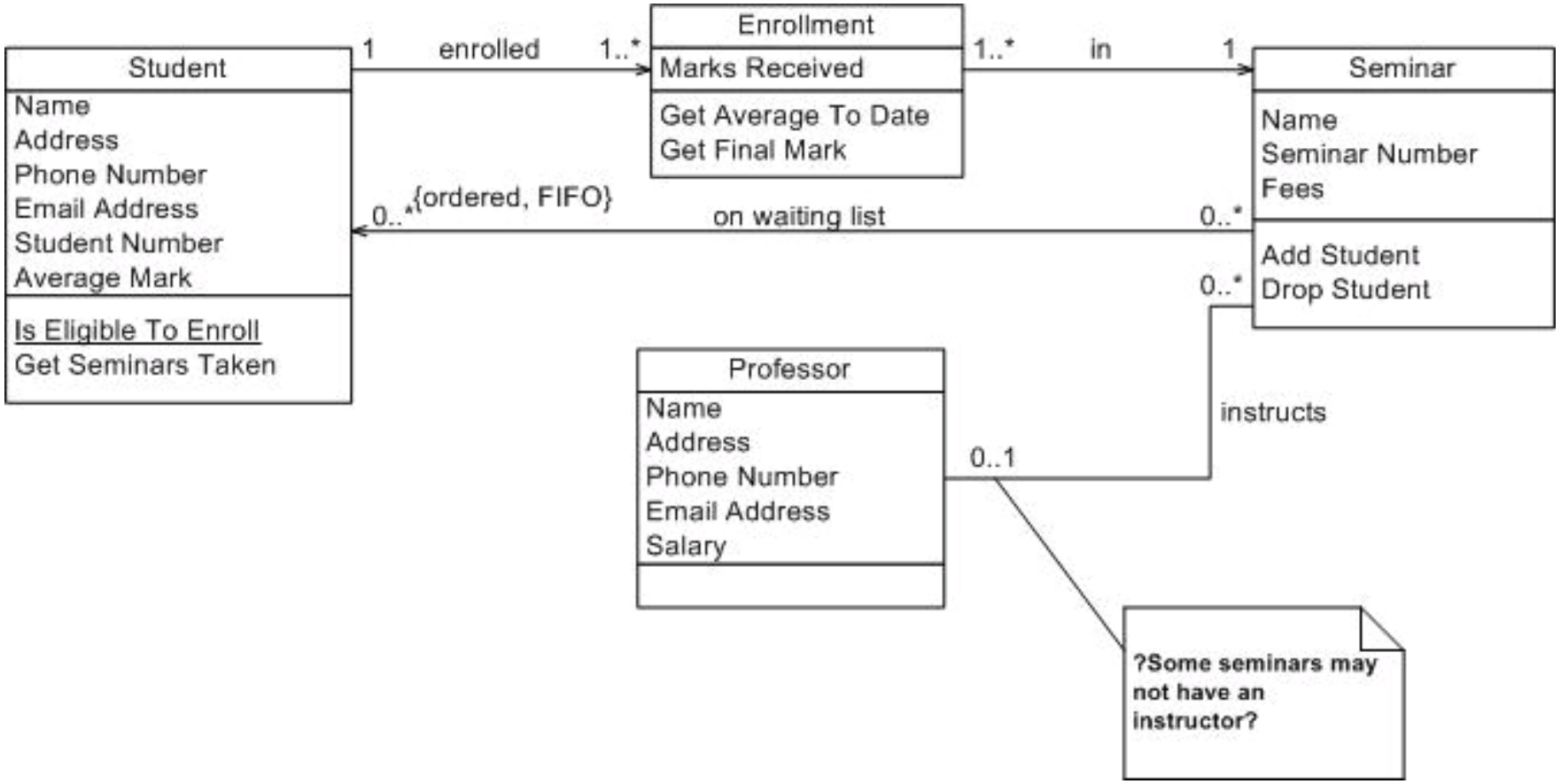



ExternalDescription
1
1..*
ProcessDefinitionElement
1
0..*
Dependency
Constraint
0..*
0..*
1
1..*
Guidance
Precondition
Goal
*
*
1
WorkDefinition
*
1
ProcessRole
ArtifactType
ProcessPackage
1
1
1
Role
Phase
Iteration
Lifecycle
0..*
0..*
ActivityParameter
ProcessComponent
Activity
0..*
1
Discipline
Process

Tools to “do UML”

- *Pen and pencil (and eraser) + scanner*
- *White board and digital camera*
- IBM Rational Rose & Rose XDE ($$)
- IBM Rational Software Architect ($$$$$)
- Microsoft Visio ($)
  - use the free Pavel Hruby stencil: http://www.softwarestencils.com/uml/index.html
- Eclipse UML Plug-in (free)
- Visual Paradigm (free)
- Together Designer (Borland) (free)
- ArgoUML (free)
- …. and many more, mostly not free


Enrollment
Marks Recieved
Calc Avg Mark
Calc Final Mark
Student
name
address
phone number
Student number
Determine if eligible to enroll in University
Provide list of Seminars taken
Calculate average mark to date
0..*
enrolled in
1..*
Seminar
Name
Number
Fees
Add Student
Drop Student
Maintain Waiting List
0..*
Waiting list
a:
a:
0..1
instructs
Professor
Name
Address
Phone
Email
Salary


Student
Name
Address
Phone Number
Email Address
Student Number
Average Mark
Is Eligible To Enroll
Get Seminars Taken
1
enrolled
1..*
Enrollment
Marks Received
Get Average To Date
Get Final Mark
1..*
in
1
Seminar
Name
Seminar Number
Fees
Add Student
Drop Student
0..*
{ordered, FIFO}
on waiting list
0..*
0..*
instructs
Professor
Name
Address
Phone Number
Email Address
Salary
0..1
?Some seminars may not have an instructor?

References

- Books:
  - Martin Fowler: *UML Distilled,* 3rd ed., AWL
  - Grady Booch: *UML User's Guide,* AWL
- On-line Tutorial & Resources
  - Tutorial David Braun *et al.* (Kennesaw St. U) at: http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/
  - Scott Ambler's http://www.agilemodeling.com/
  - http://www.uml.org/
  - http://www-306.ibm.com/software/rational/uml/

Deployment Diagram