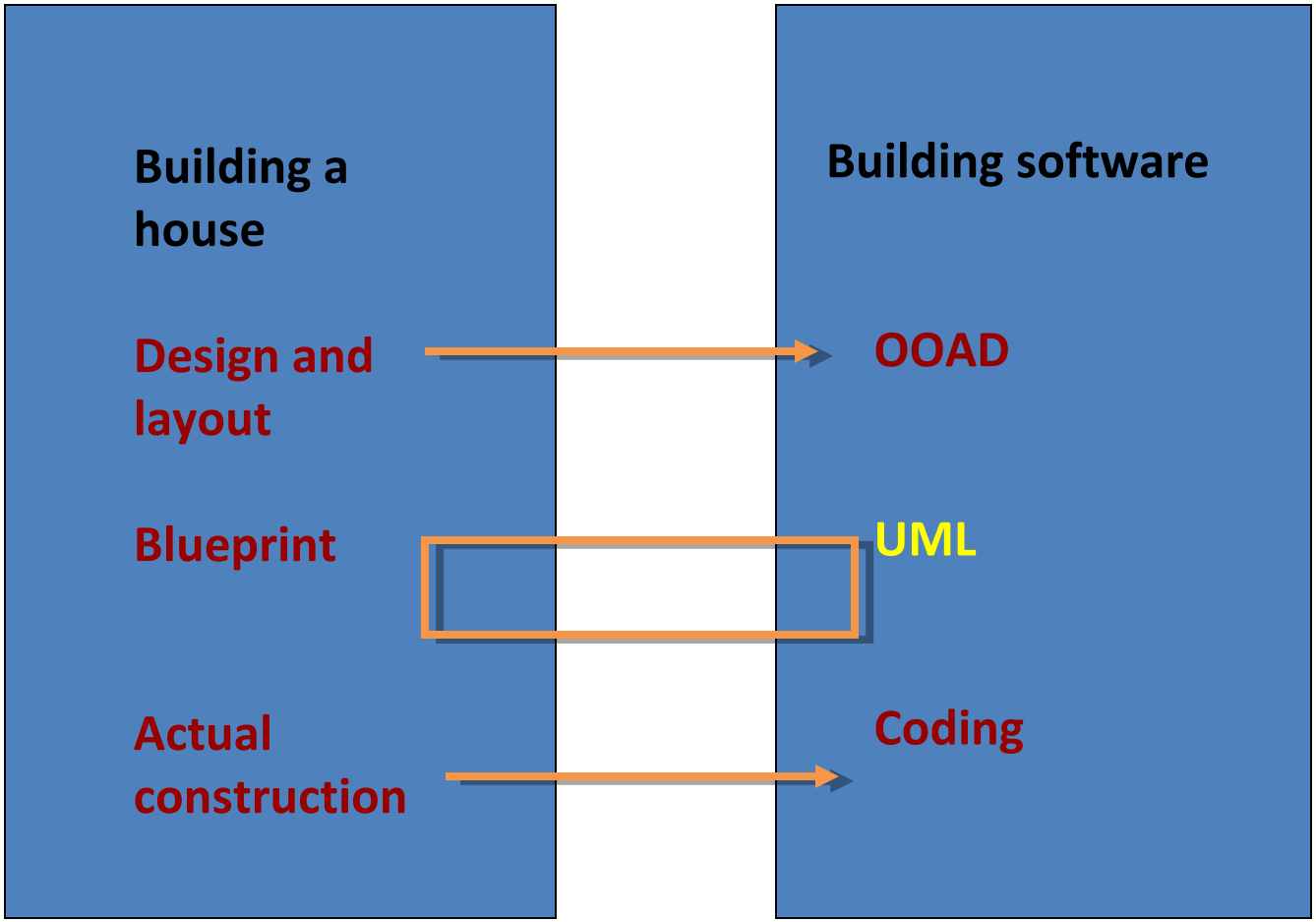
Building Blocks of UML

OOAD with UML



A conceptual model of the UML

To understand the UML, you need to form a conceptual model of the language, and this requires learning three major elements:

- The UML's basic building blocks.
- The rules that dictate how these building blocks put together.
- Some mechanisms that apply throughout the UML.

Building Blocks of UML

- Things – abstractions that are first class citizen in a model
- Relationships - tie things together
- Diagrams - group interesting collections of things

Things in the UML

There are four kinds of things in the UML:

- ✓ Structural things
- ✓ Behavioral things
- ✓ Grouping things
- ✓ Annotational things

Structural Things

- Static part of a model, representing elements that are either conceptual or physical
  - Classes
  - Interfaces
  - Collaborations
  - Use cases
  - Active classes
  - Components
  - Nodes

- ***class***
  - a *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics
  - A class implements one or more interfaces.
- ***interface***
  - a collection of operations that specify a service of a class or component
  - describes the externally visible behavior of that element
  - defines a set of operation specifications but never a set of operation implementations
  - attached to class or component that realizes the interface
- ***Collaboration***
  - defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior
  - Collaborations have structural, as well as behavioral dimensions
  - A given class may participate in several collaborations
  - therefore represent the implementation of patterns that make up a system

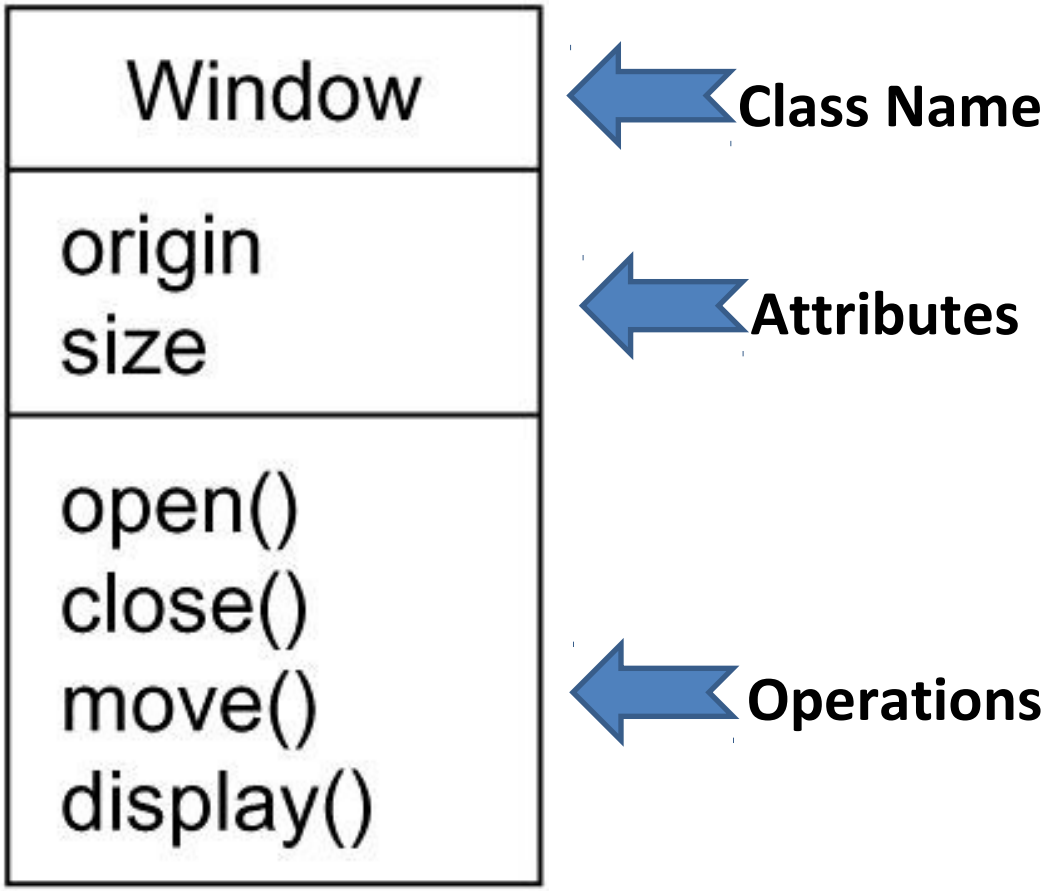


Figure 1: Classes

Figure 2: Interfaces

Figure 3: Collaborations

- ***use case*** - a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor. A use case is used to structure the behavioral things in a model. A use case is realized by a collaboration.
- ***active class*** - a class whose objects own one or more processes or threads and therefore can initiate control activity. An active class is just like a class except that its objects represent elements whose behavior is concurrent with other elements.
- ***component*** - a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. Represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations.
- ***node*** - a physical element that exists at run time and represents a computational resource, having at least some memory and, often, processing capability. A set of components may reside on a node and may also migrate from node to node.

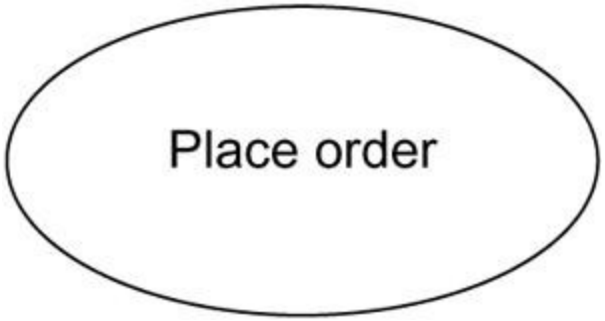


Figure 3: Use Cases

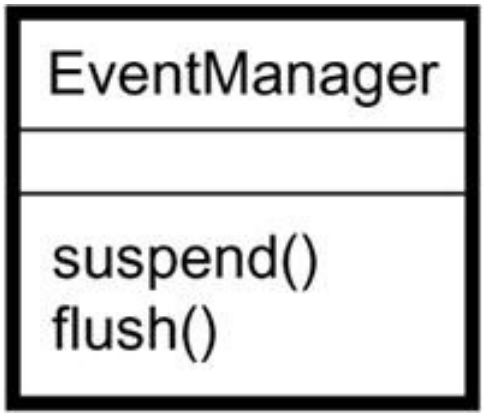


Figure 4: Active Classes

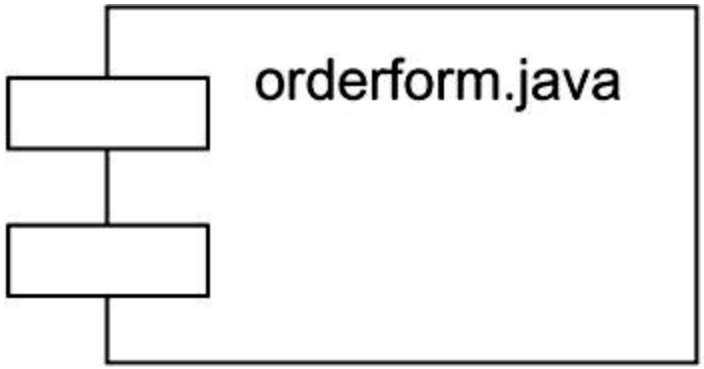


Figure 5: Components

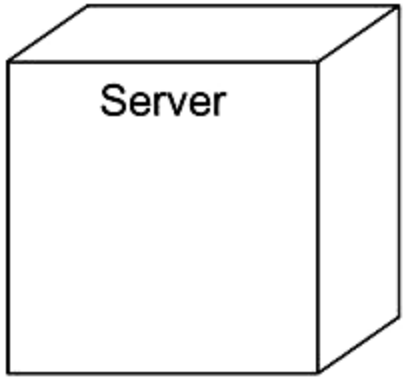


Figure 6: Nodes

Behavioral Things

- Behavioral things are the ***dynamic parts*** of UML models.
- These are the verbs of a model, representing behavior over time and space.
- **Two primary kinds** of behavioral things.
  - ***Interaction*** *(exchange of set of messages among a set of objects)(includes message, links, action sequence)*
  - ***State Machine*** *(specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events.)(includes states, transitions, events, activities)*

**** Shown in the figure 7 and 8***

Grouping Things

- Grouping things are the organizational parts of UML models. These are the boxes into which a model can be decomposed.
- There is one primary kind of grouping.
  - **Packages**(mechanism for organizing elements into groups) (a package is purely conceptual meaning that it exists only at development time)

    **** Shown in the figure 9***

Annotational Things

- Annotational things are the explanatory parts of UML models. These are the comments you may apply to describe, illuminate, and remark about any element in a model.
- There is one primary kind of annotational things.
  - **Note** (a symbol for rendering constraints and comments attached to an element or a collection of elements)

    **** Shown in the figure 10***

Figure 7: Messages

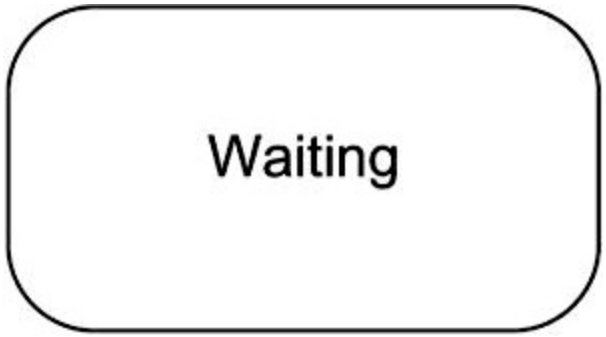


Figure 8: States

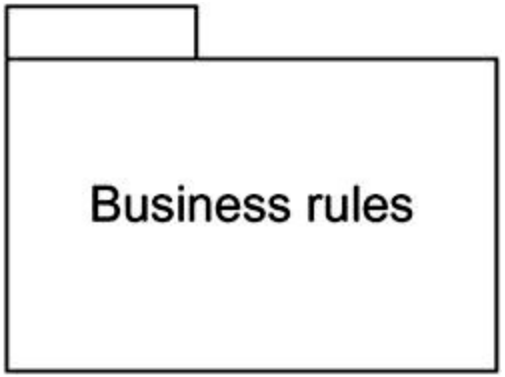


Figure 9: Packages

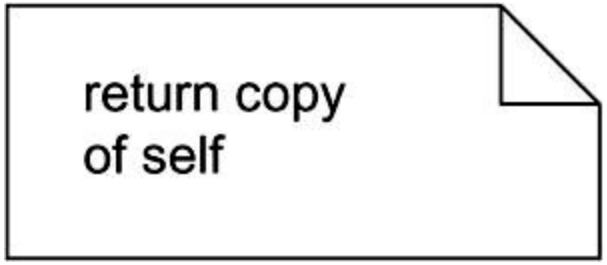


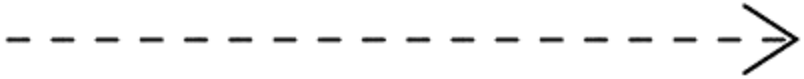
Figure 10: Notes

Relationships in the UML

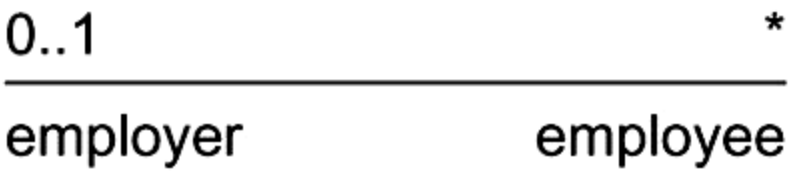
Relationships in the UML

- There are four kinds of relationships in the UML:
  1. Dependency *(a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing)*
  2. Association *(a structural relationship that describes a set of links, a link being a connection among objects)*
  3. Generalization *(a specialization / generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent))*
  4. Realization (a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out) (between interfaces and the classes or components and between use cases and the collaborations)

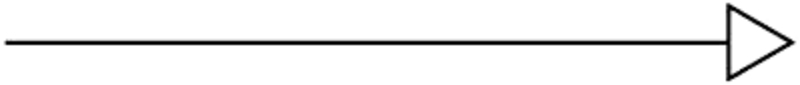
Relationships in the UML



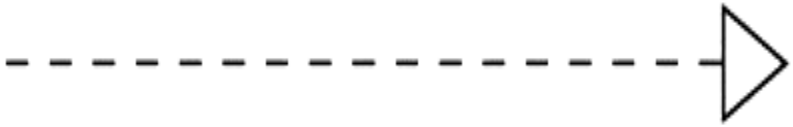
1. Dependencies: rendered as a dashed line, possibly directed, and occasionally including a label



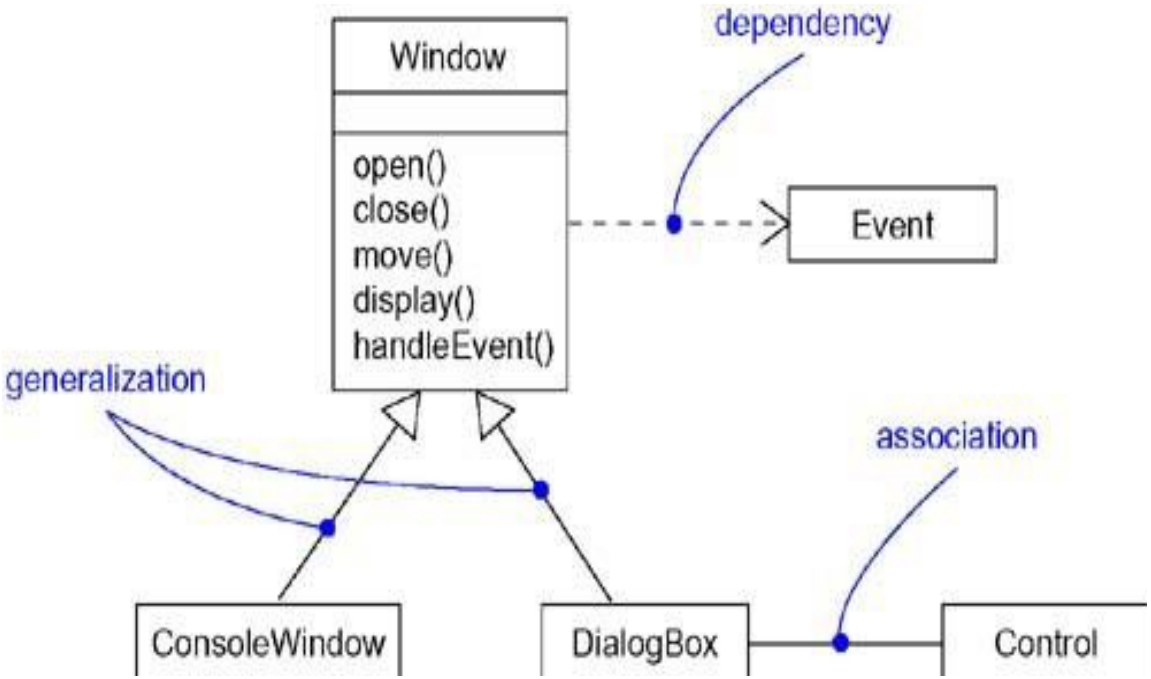
2. Associations: rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names

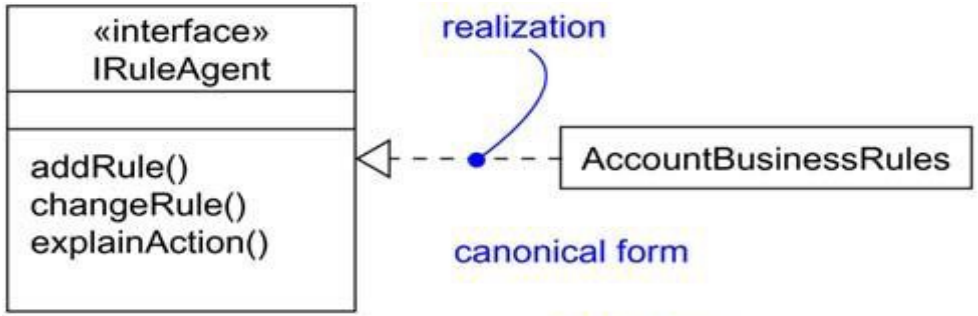


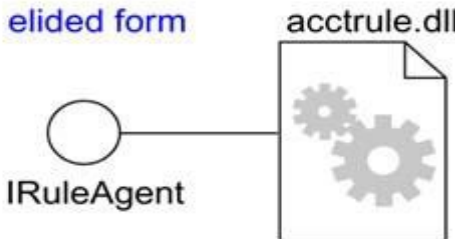
3. Generalizations: rendered as a solid line with a hollow arrowhead pointing to the parent

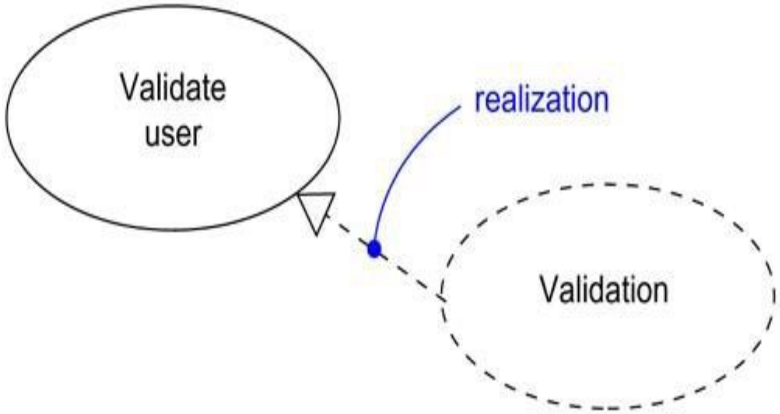


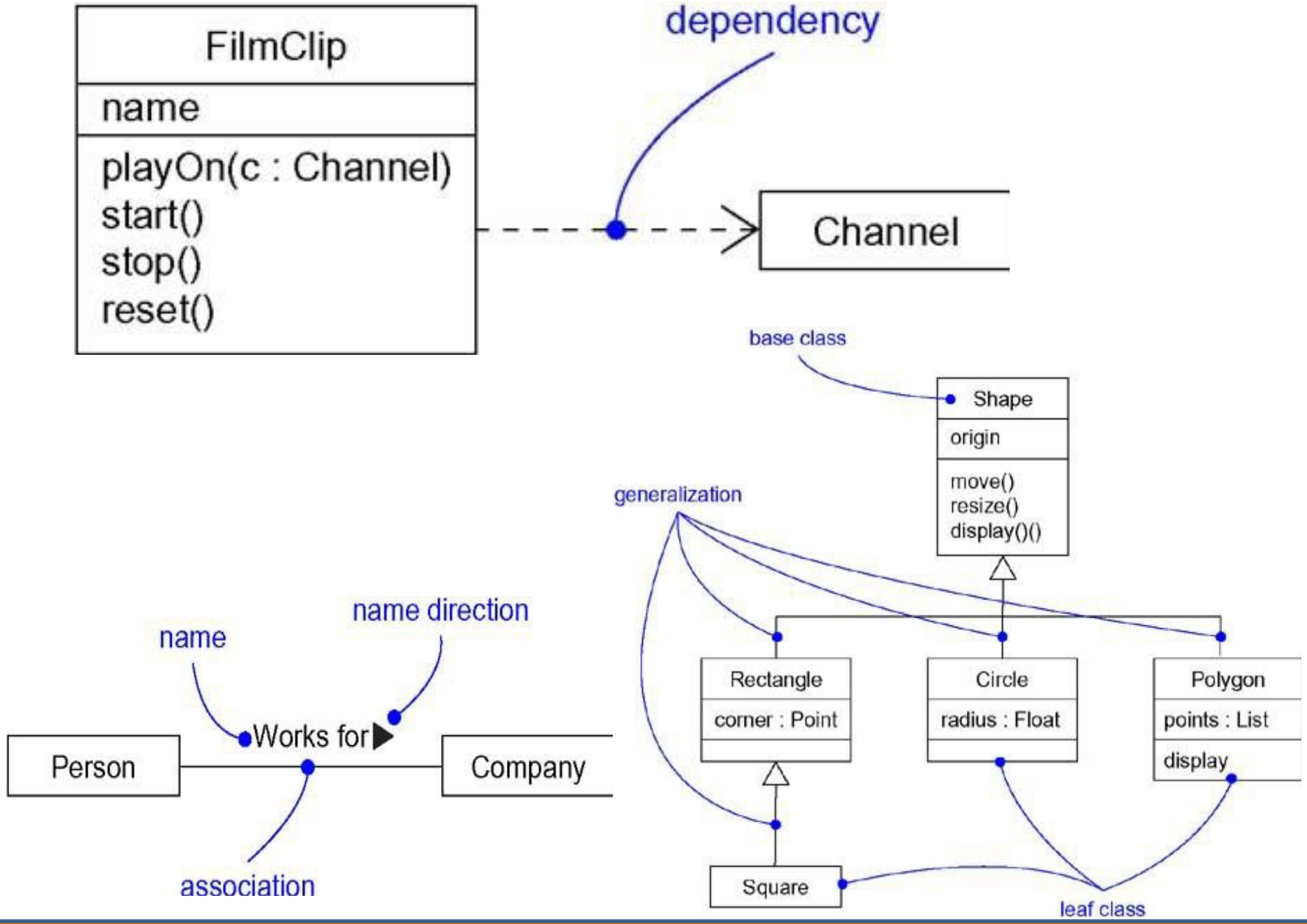
4. Realization: rendered as a cross between a generalization and a dependency relationship


Window
open()
close()
move()
display()
handleEvent()
dependency
Event
generalization
association
ConsoleWindow
DialogBox
Control


«interface»
IRuleAgent
addRule()
changeRule()
explainAction()
realization
AccountBusinessRules
canonical form


elided form
acctrule.dll
IRuleAgent


Validate
user
realization
Validation

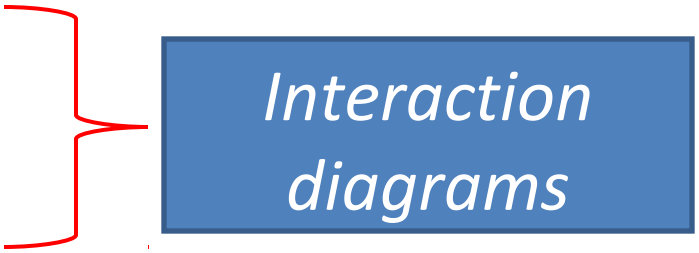

FilmClip
name
playOn(c : Channel)
start()
stop()
reset()
dependency
Channel
base class
Shape
origin
move()
resize()
display()()
generalization
Rectangle
corner : Point
Circle
radius : Float
Polygon
points : List
display
Square
leaf class
name
name direction
Works for
Person
Company
association

Diagrams in the UML

- graphical presentation of a set of elements, rendered as a connected graph of vertices (things) and arcs (relationships).
- a projection into a system
- The same element may appear in all diagrams, only a few diagrams (the most common case), or in no diagrams at all (a very rare case)

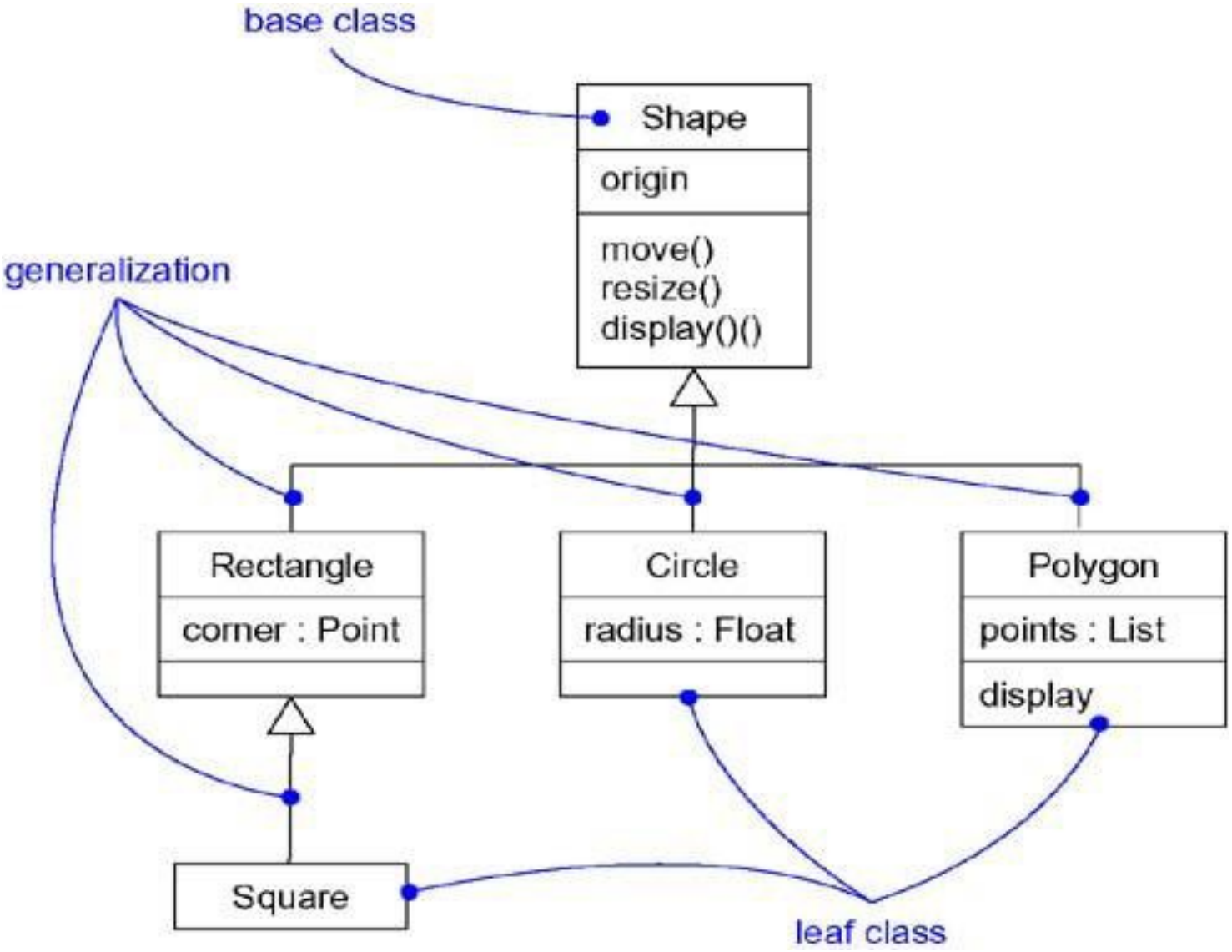
Diagrams in the UML

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Statechart diagram
7. Activity diagram
8. Component diagram
9. Deployment diagram



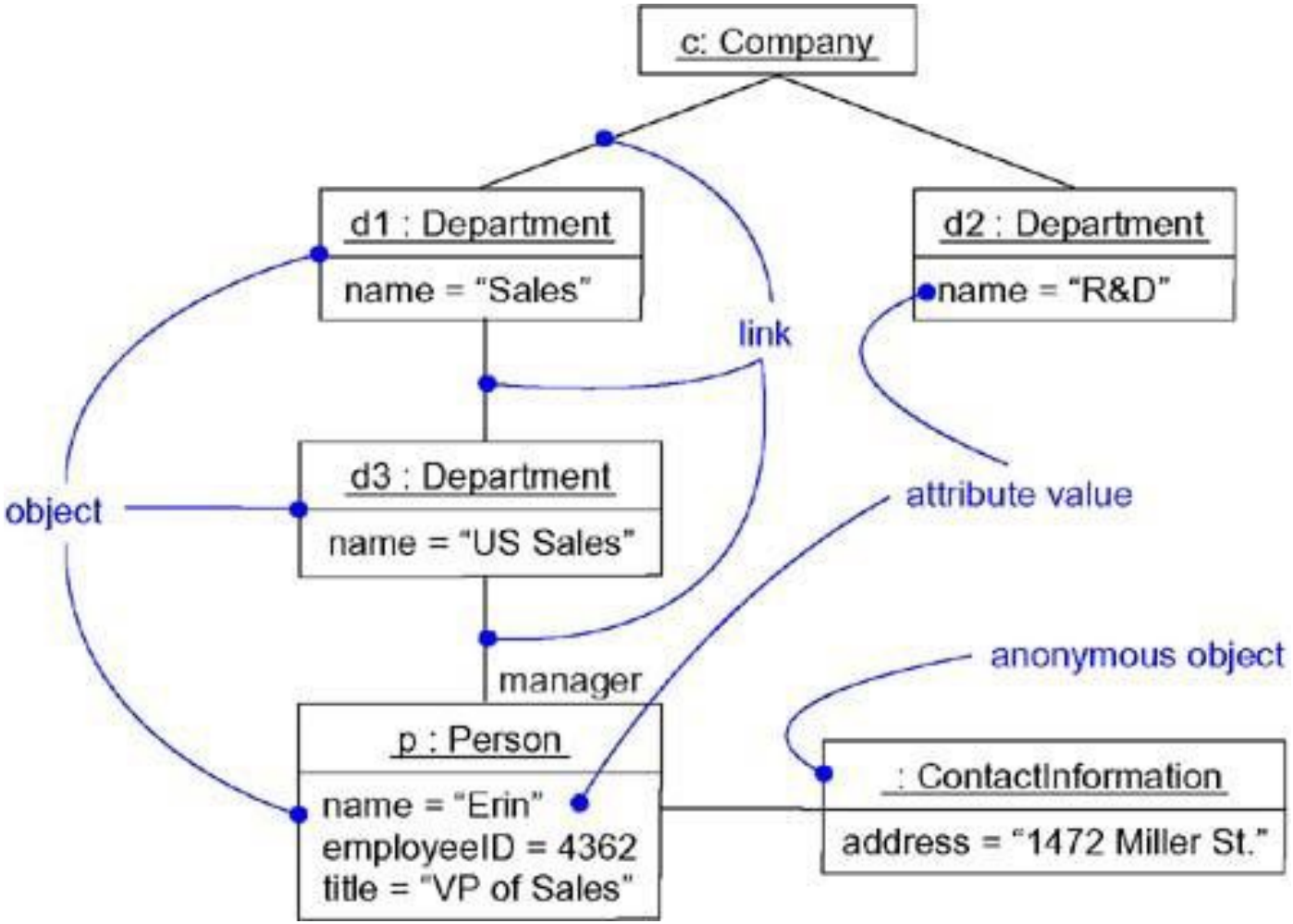
1. Class Diagram

- A class diagram shows a set of classes, interfaces, and collaborations and their relationships.
- most common diagram found in modeling object-oriented systems.
- address the **static design** view of a system.
- Class diagrams that include active classes address the static process view of a system.


base class
Shape
origin
move()
resize()
display()()
generalization
Rectangle
corner : Point
Circle
radius : Float
Polygon
points : List
display
Square
leaf class

2. Object Diagram

- An object diagram shows a set of objects and their relationships.
- Object diagrams represent static snapshots of instances of the things found in class diagrams.
- address the static design view or static process view of a system as do class diagrams, but from the perspective of real or prototypical cases.

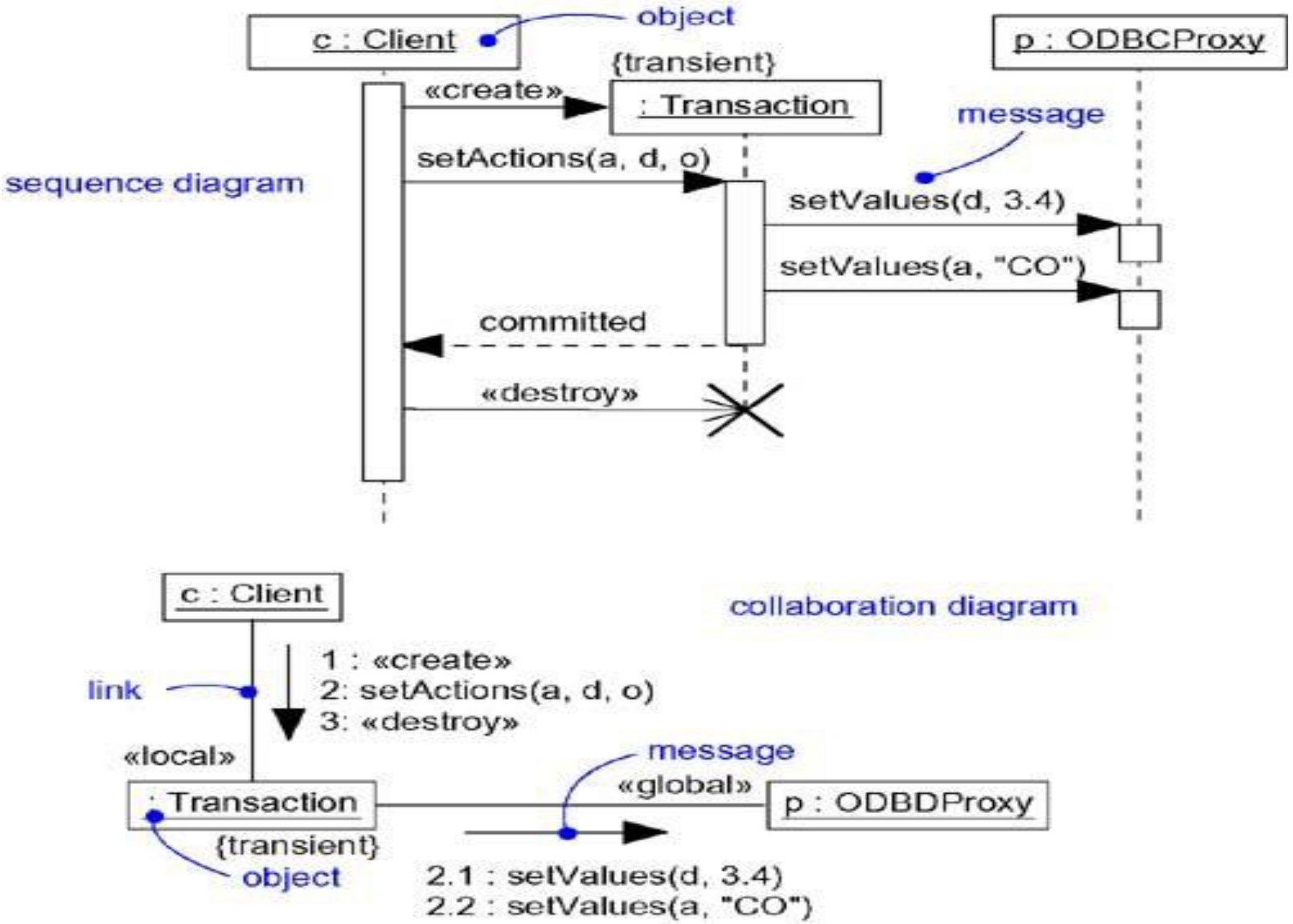

c: Company
d1 : Department
name = "Sales"
d2 : Department
name = "R&D"
link
object
d3 : Department
name = "US Sales"
attribute value
manager
anonymous object
p : Person
name = "Erin"
employeeID = 4362
title = "VP of Sales"
: ContactInformation
address = "1472 Miller St."

3. Use Case Diagram

- A use case diagram shows a set of use cases and actors (a special kind of class) and their relationships.
- address the static use case view of a system.
- especially important in organizing and modeling the behaviors of a system.

Interaction Diagrams

- Both **sequence** diagrams and **collaboration** diagrams are kinds of interaction diagrams.
- Arc shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them.
- **Interaction diagrams address the dynamic view of a system**


c : Client
object
{transient}
«create»
: Transaction
p : ODBCProxy
message
sequence diagram
setActions(a, d, o)
setValues(d, 3.4)
setValues(a, "CO")
committed
«destroy»
c : Client
collaboration diagram
1 : «create»
2: setActions(a, d, o)
3: «destroy»
link
«local»
message
«global»
: Transaction
p : ODBDProxy
{transient}
object
2.1 : setValues(d, 3.4)
2.2 : setValues(a, "CO")

4. Sequence Diagram

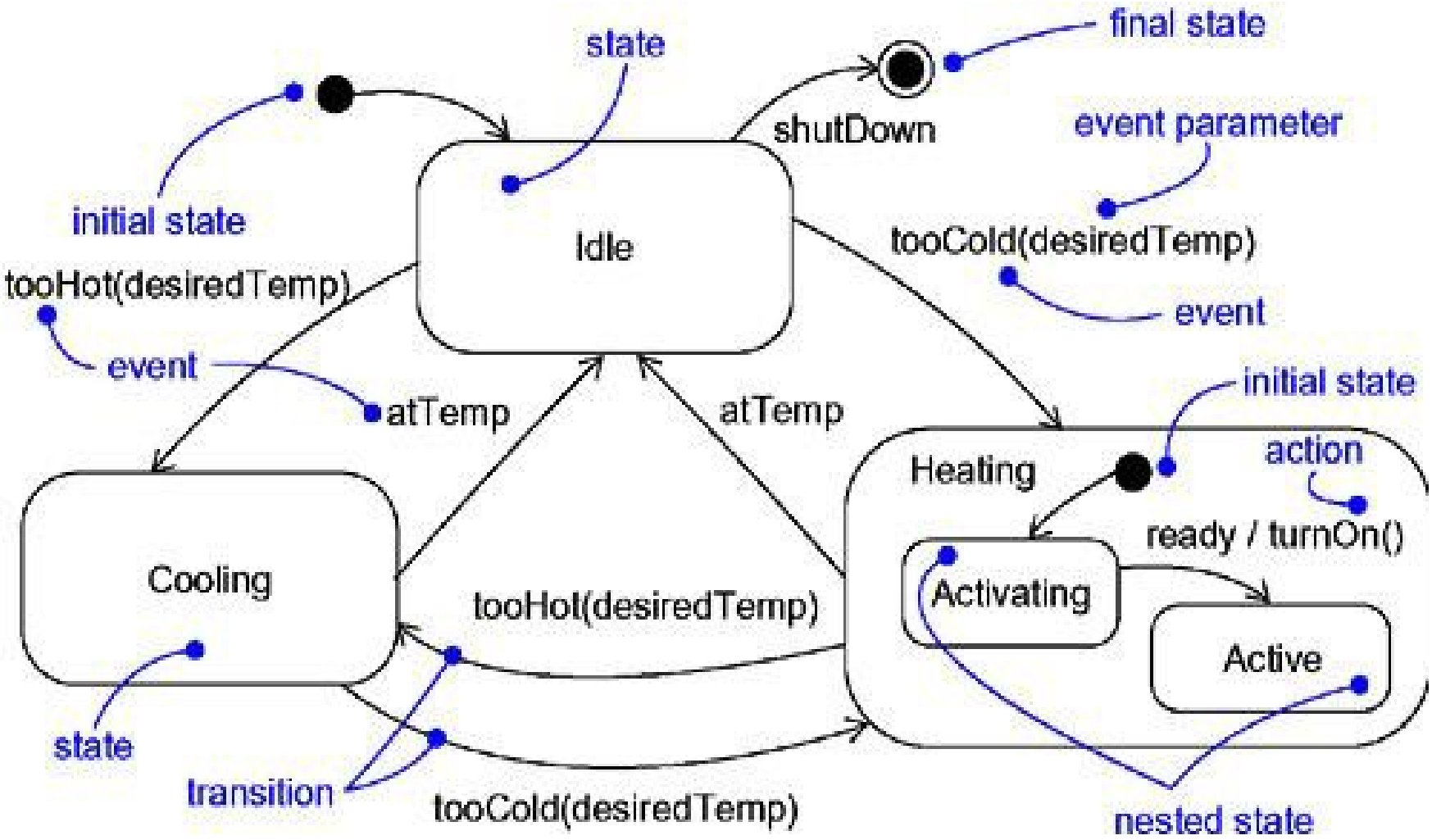
- A sequence diagram is an interaction diagram that emphasizes the **time-ordering** of messages.

5. Collaboration Diagram

- collaboration diagram is an interaction diagram that emphasizes the **structural organization** of the objects that send and receive messages.
- **Note:** Sequence diagrams and collaboration diagrams are isomorphic, meaning that you can take one and transform it into the other.

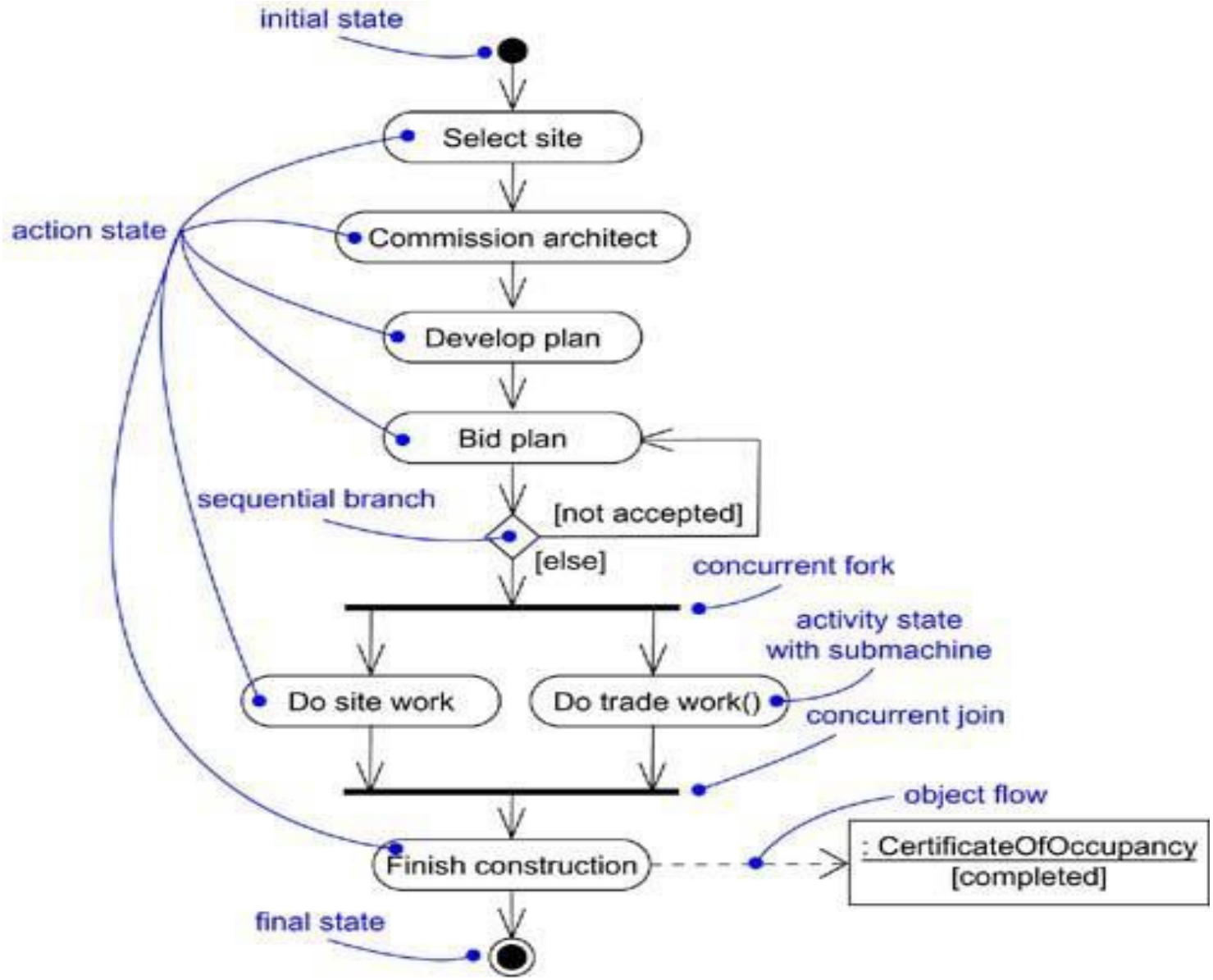
6. Statechart Diagram

- A **statechart diagram** shows a state machine, consisting of states, transitions, events and activities.
- Statechart diagrams address the dynamic view of a system.
- especially important in modeling the behavior of an interface, class, or collaboration and emphasize the event-ordered behavior of an object, which is especially useful in modeling reactive systems.


state
final state
shutDown
event parameter
initial state
Idle
tooCold(desiredTemp)
tooHot(desiredTemp)
event
event
atTemp
atTemp
initial state
Heating
action
ready / turnOn()
Cooling
Activating
tooHot(desiredTemp)
Active
state
transition
tooCold(desiredTemp)
nested state

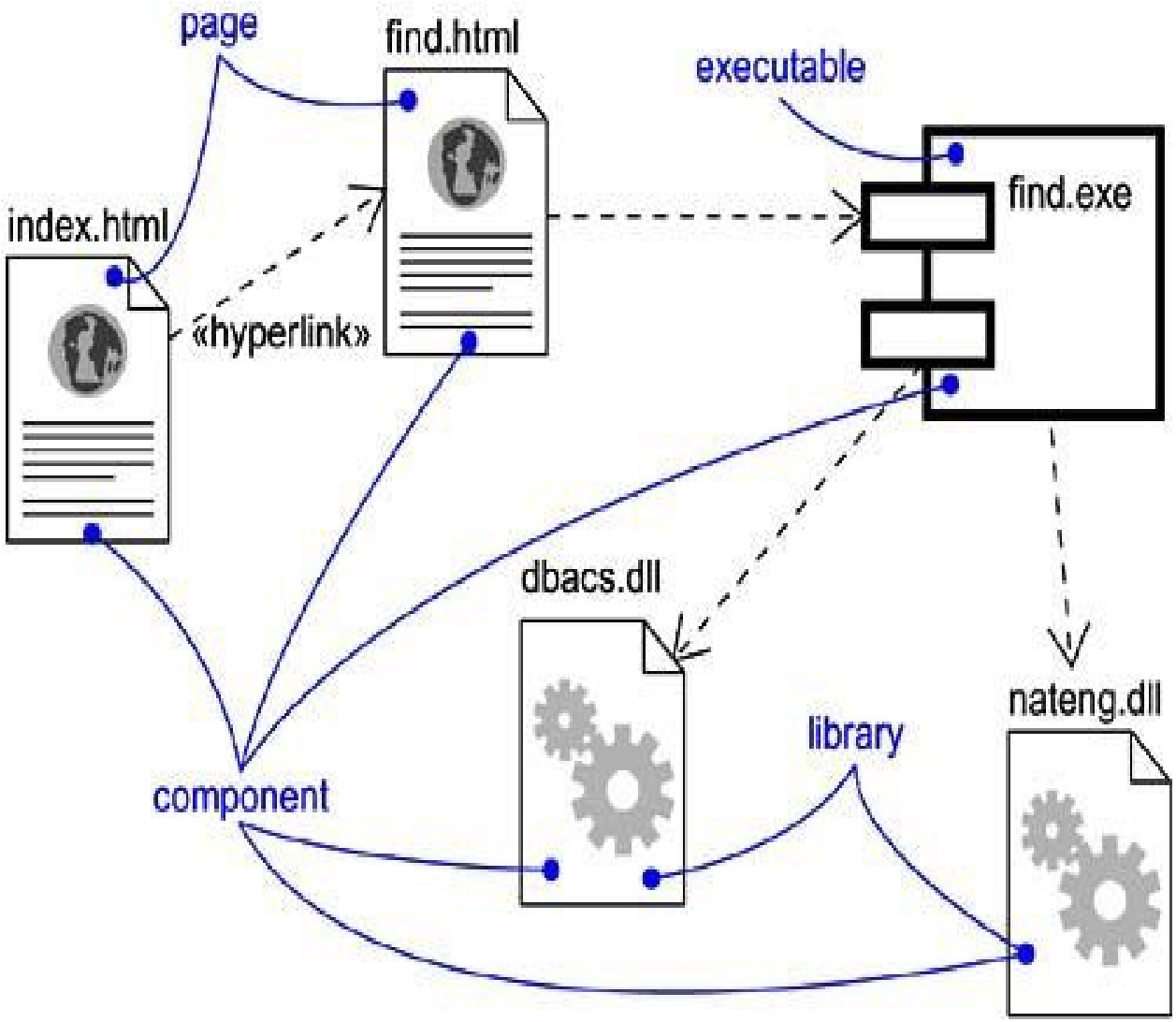
7. Activity Diagram

- An activity diagram is a special kind of a statechart diagram that shows the **flow from activity to activity** within a system.
- Activity diagrams address the dynamic view of a system.
- They are especially important in modeling the function of a system and emphasize the flow of control among objects


initial state
Select site
action state
Commission architect
Develop plan
Bid plan
sequential branch
[not accepted]
[else]
concurrent fork
activity state
with submachine
Do site work
Do trade work()
concurrent join
object flow
Finish construction
: CertificateOfOccupancy
[completed]
final state

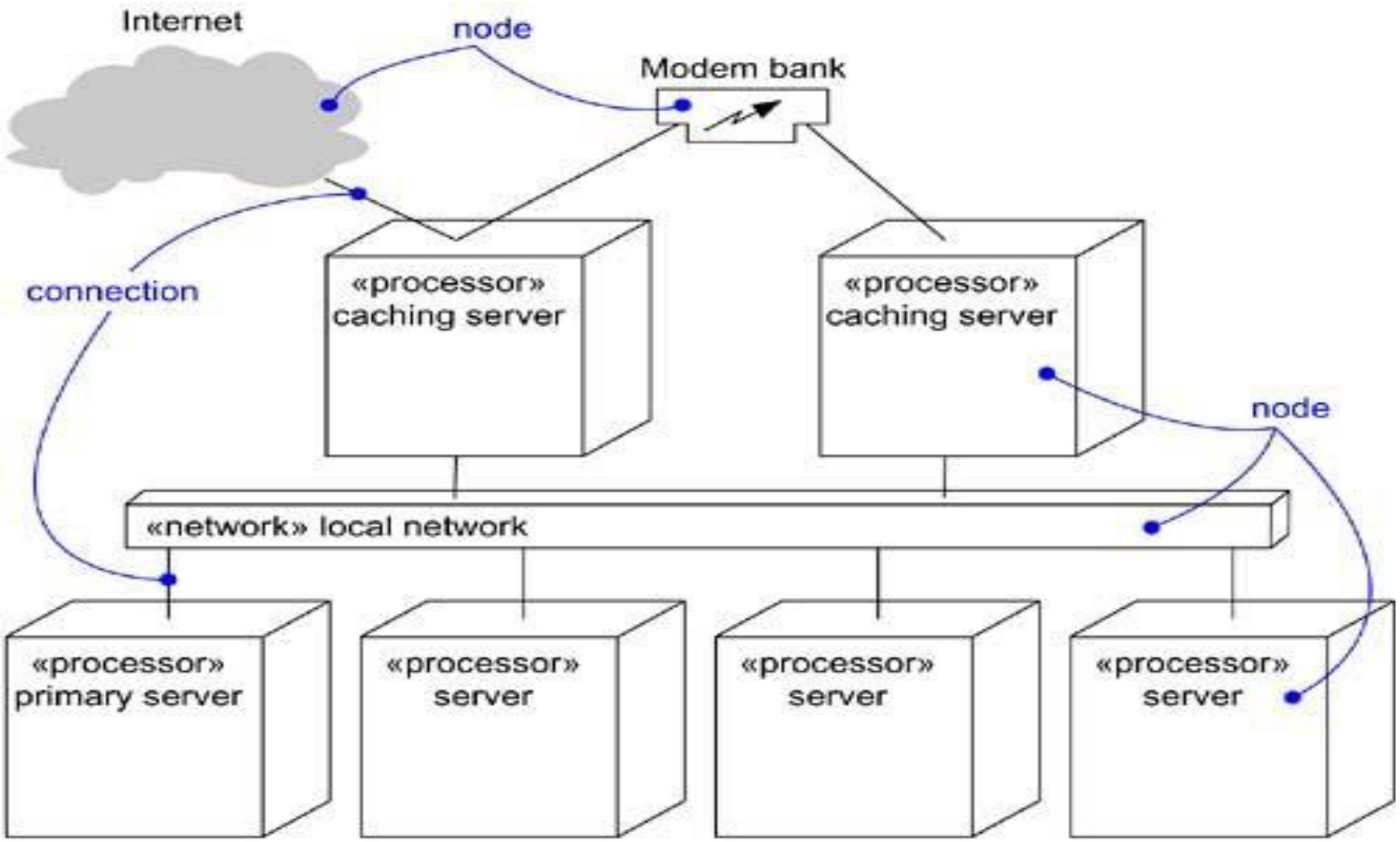
8. Component Diagram

- A component diagram shows the organizations and dependencies among a set of components.
- Component diagrams address the static implementation view of a system.
- They are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations.


page
find.html
executable
index.html
find.exe
«hyperlink»
dbacs.dll
nateng.dll
library
component

9. Deployment Diagram

- A deployment diagram shows the configuration of run-time processing nodes and the components that live on them.
- Deployment diagrams address the static deployment view of an architecture.
- They are related to component diagrams in that a node typically encloses one or more components.


Internet
node
Modem bank
connection
«processor»
caching server
«processor»
caching server
node
«network» local network
«processor»
primary server
«processor»
server
«processor»
server
«processor»
server