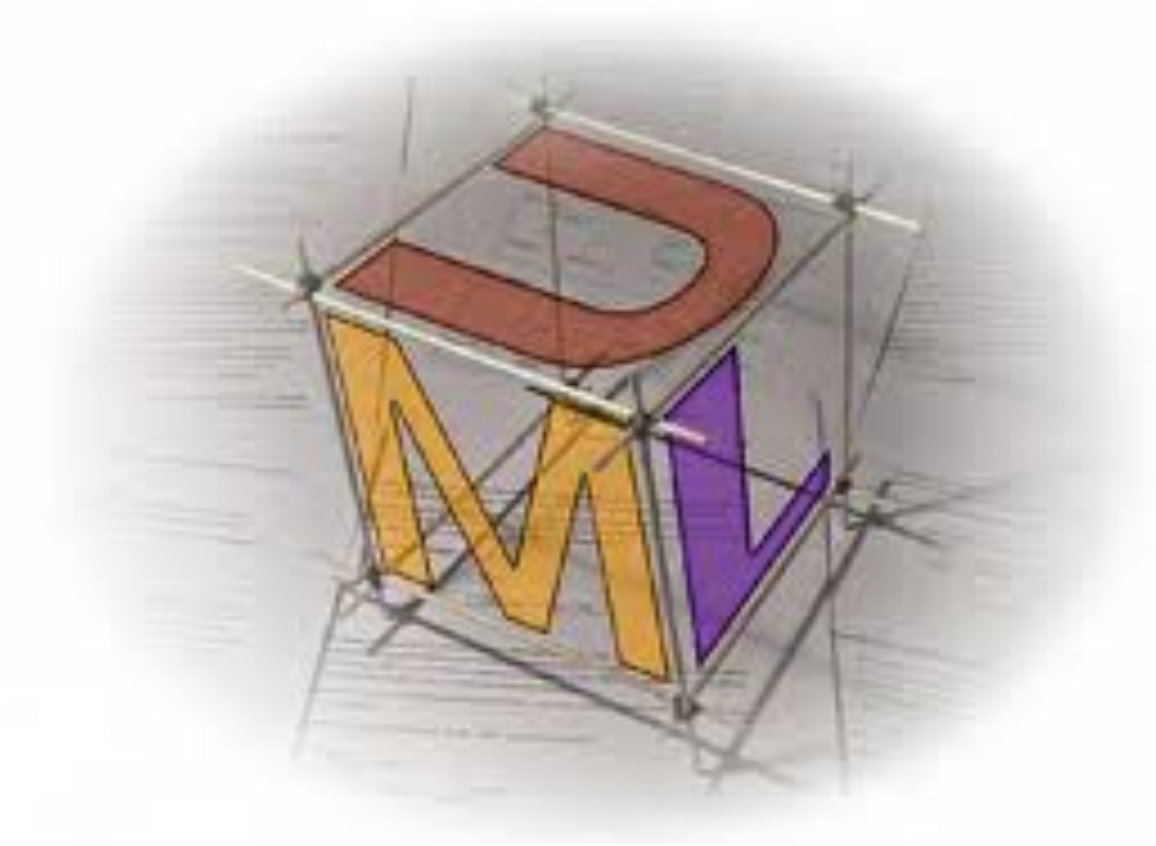


UNIFIED MODELLING LANGUAGE (UML)

An Overview of Diagram Types used in the Standard Object Modeling Language UML 2.0

What is UML?

- Standard general-purpose modeling language designed for OO software engineering
- Used to create visual models of object-oriented software-intensive systems
- Developed by the Object Management Group (OMG) and added to official technologies in 1997
- It fuses multiple methodologies into one common modeling language

Prior to UML:

Several popular techniques:

- The Booch method (Booch, 1993)
- The Object-Modeling Technique (OMT) (Rumbaugh et al, 1991)
- Object-Oriented Software Engineering (OOSE) (Jacobson, 1992)
- "The Three Amigos" led an international consortium, **UML Partners**, in development of UML 1.1 (November 1997)

UML 1.x

- The best ideas from the Three Amigos and the international consortium were combined into one standardized form
- Concepts from other OO methodologies were also incorporated to create a broad set of modeling methods and graphical tools
- UML is an international standard

  ISO/IEC19501:2005

UML 2.x

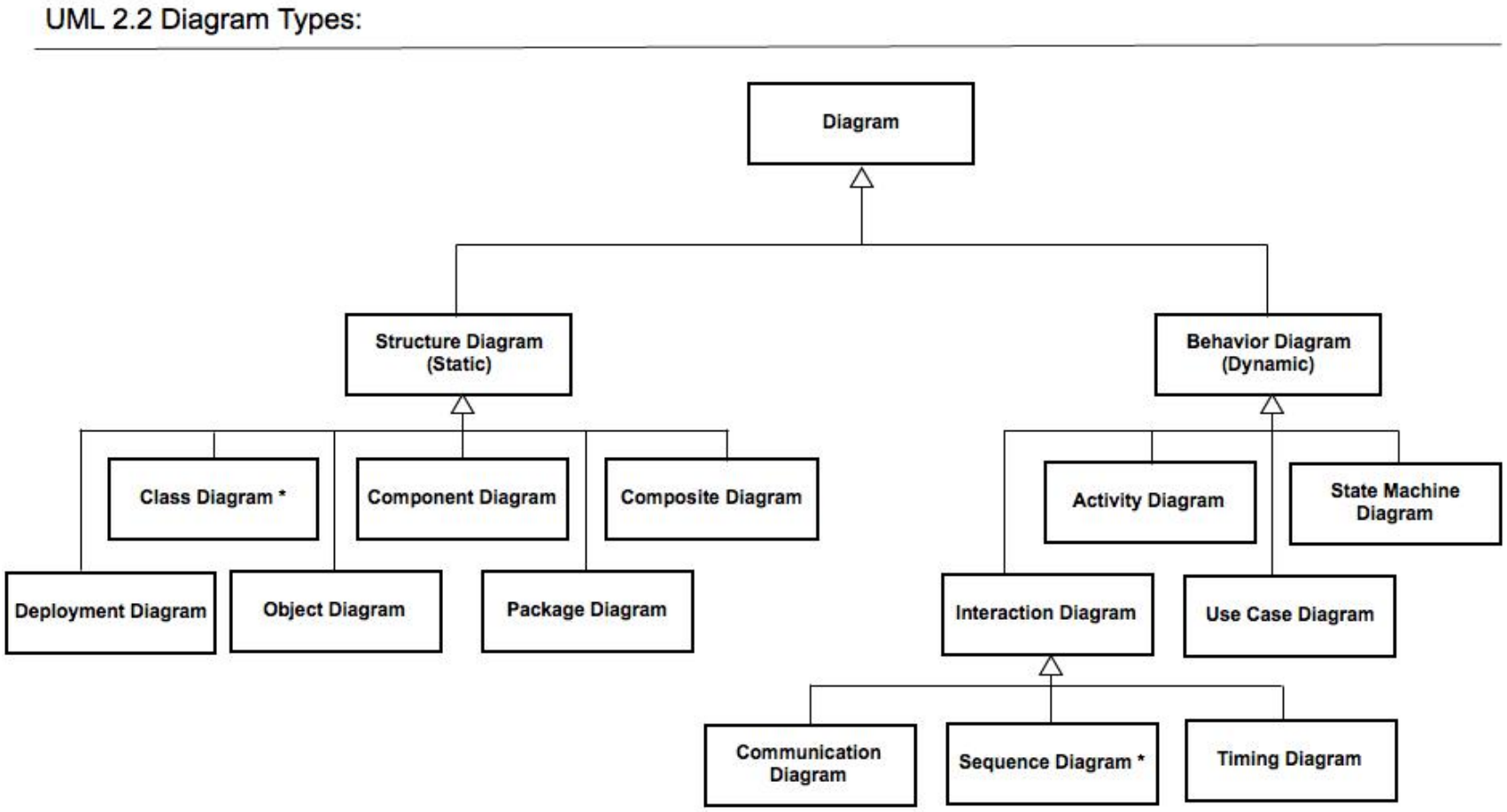
Four Parts to the UML 2.x Specification:

1. The Superstructure defining the notation and semantics for diagrams and their elements
2. Infrastructure defining the core meta-model (Meta-Object Facility – MOF used in model-driven engineering)
3. The Object Constraint Language (OCL)
4. UML Diagram Interchange – how diagram layouts are exchanged

Diagram Types

- There are 13 official diagram types
- Two Conceptual Views:
  - Structural (static): description of system architecture underlying the design
  - Behavioral (dynamic): description of how the pieces of the system interact with each other
- Most Commonly Used
  - The Class Diagram (static/structural)
  - The Sequence Diagram (dynamic/behavioral)

Diagram Types



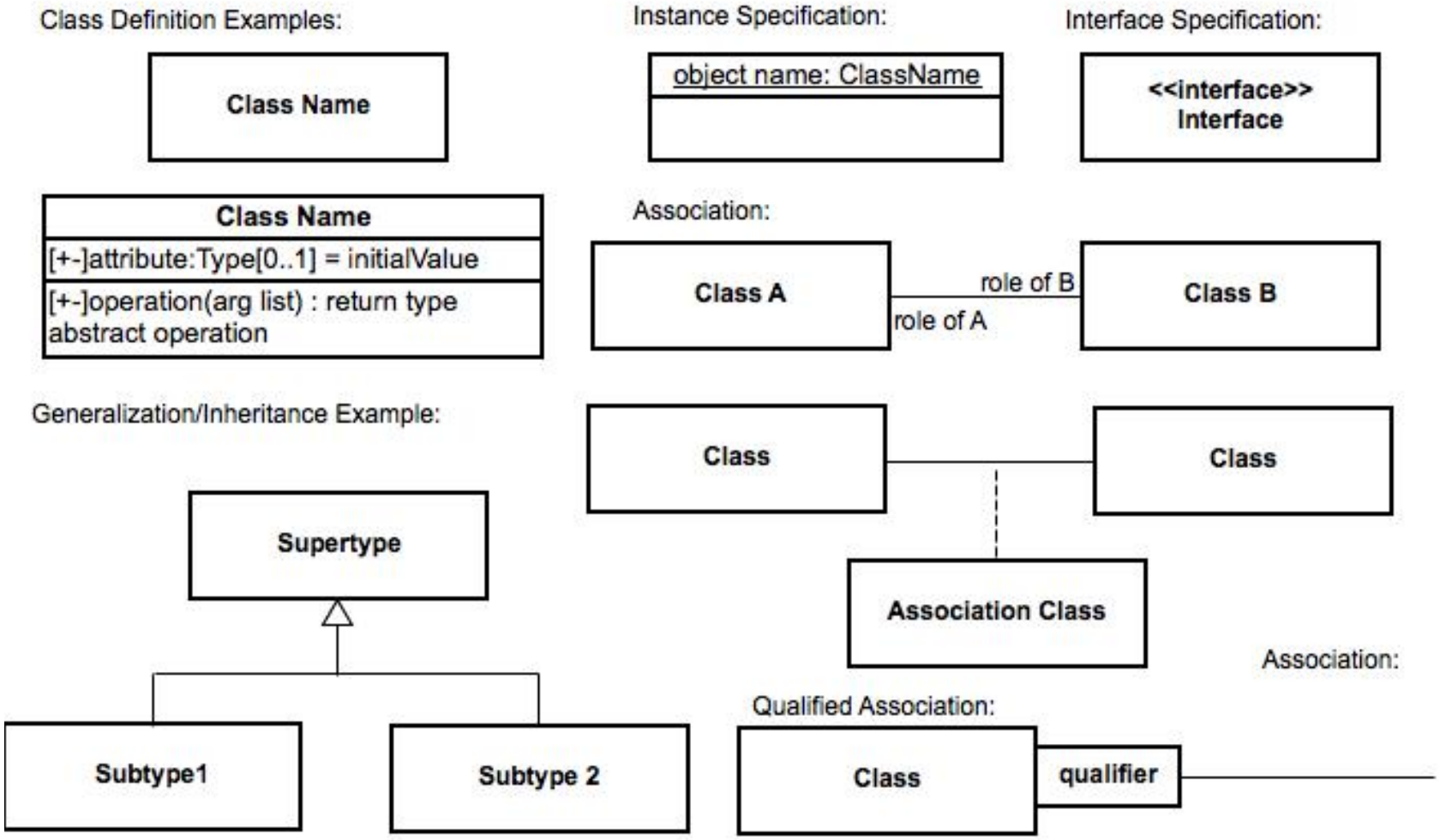
Class Diagram (structural)

Describes the types of objects in a system and the relationships between them

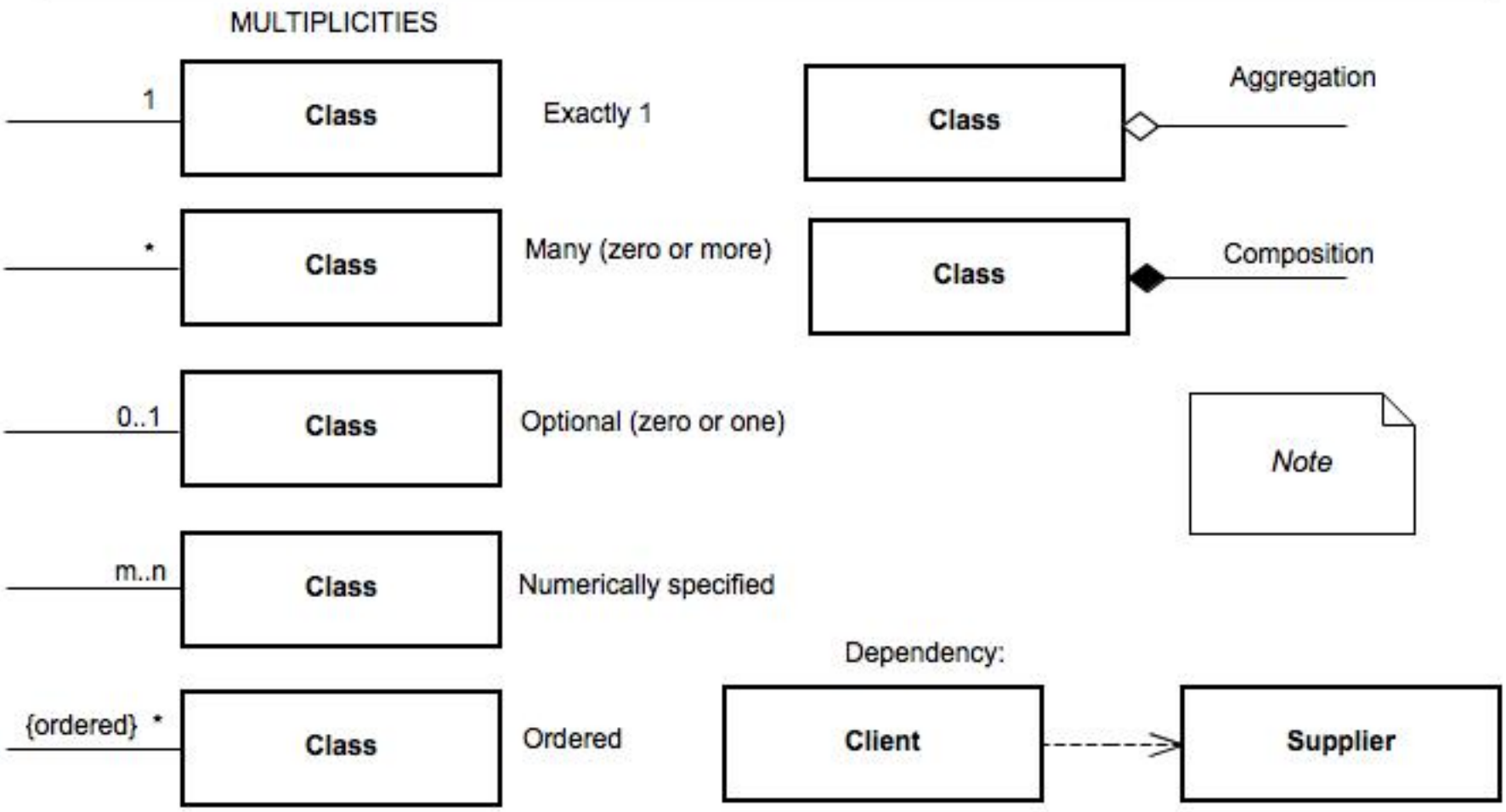
Class Diagram

- Most common diagram with the widest range of variation from simple to complex
- It can be used as simple high level overview to a detailed definition including attributes/methods
- Because we have already covered this in lecture, I am only going to provide a summary of notation
- Much of the class notation is used in other diagram types

Class Diagram Notation:



Class Diagram Notation (continued)



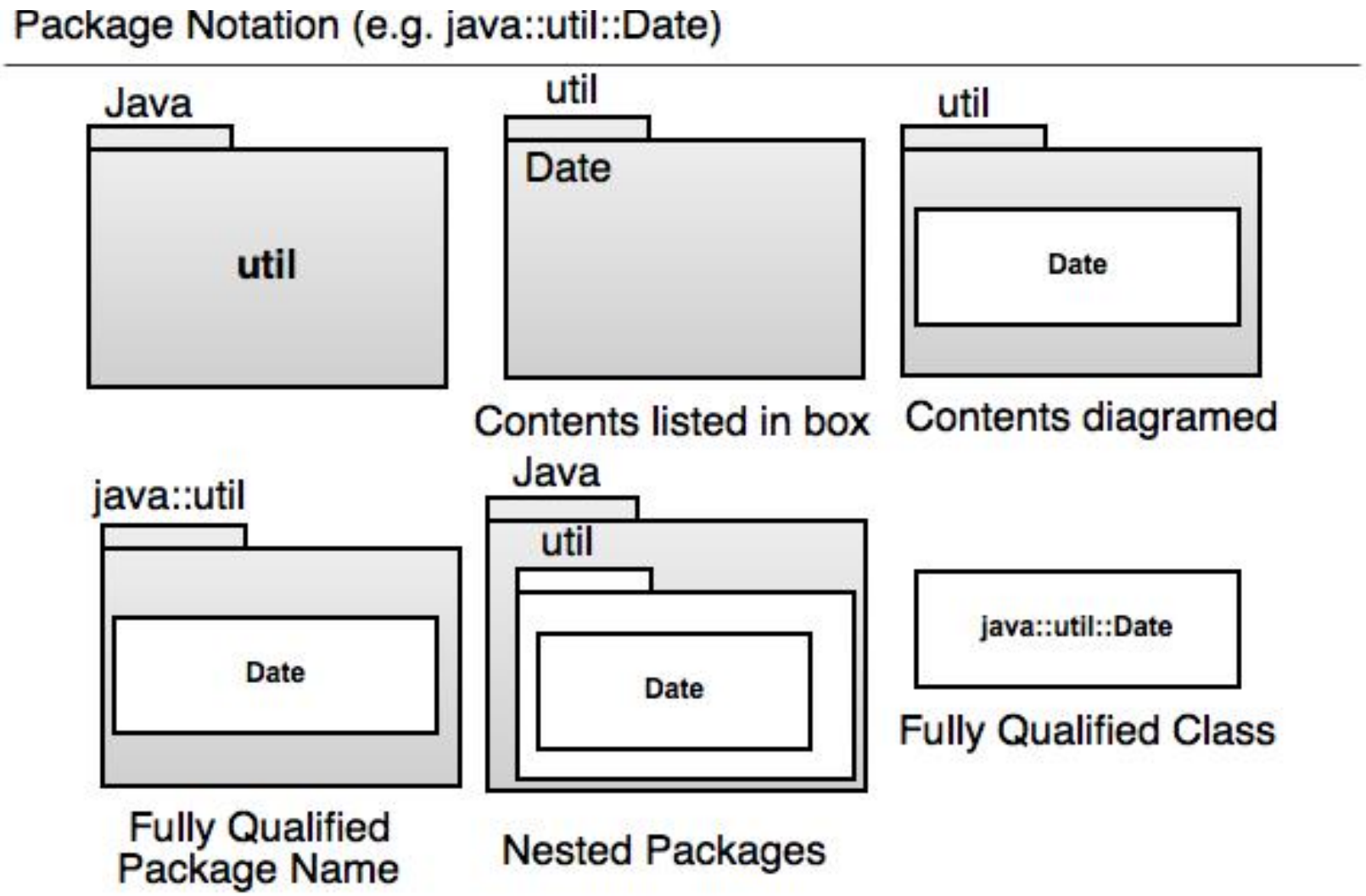
Package Diagram (structural)

Structural overview of how the components of a system are organized into higher-level units

Package Diagrams

- Allows you to group any set of UML components into higher-level units
- Most commonly used to show how classes are organized into packages
- Hierarchical structure allows packages to be broken down into sub-packages and classes
- They are denoted using "::" to separate levels of the hierarchy (e.g. java::util::Date

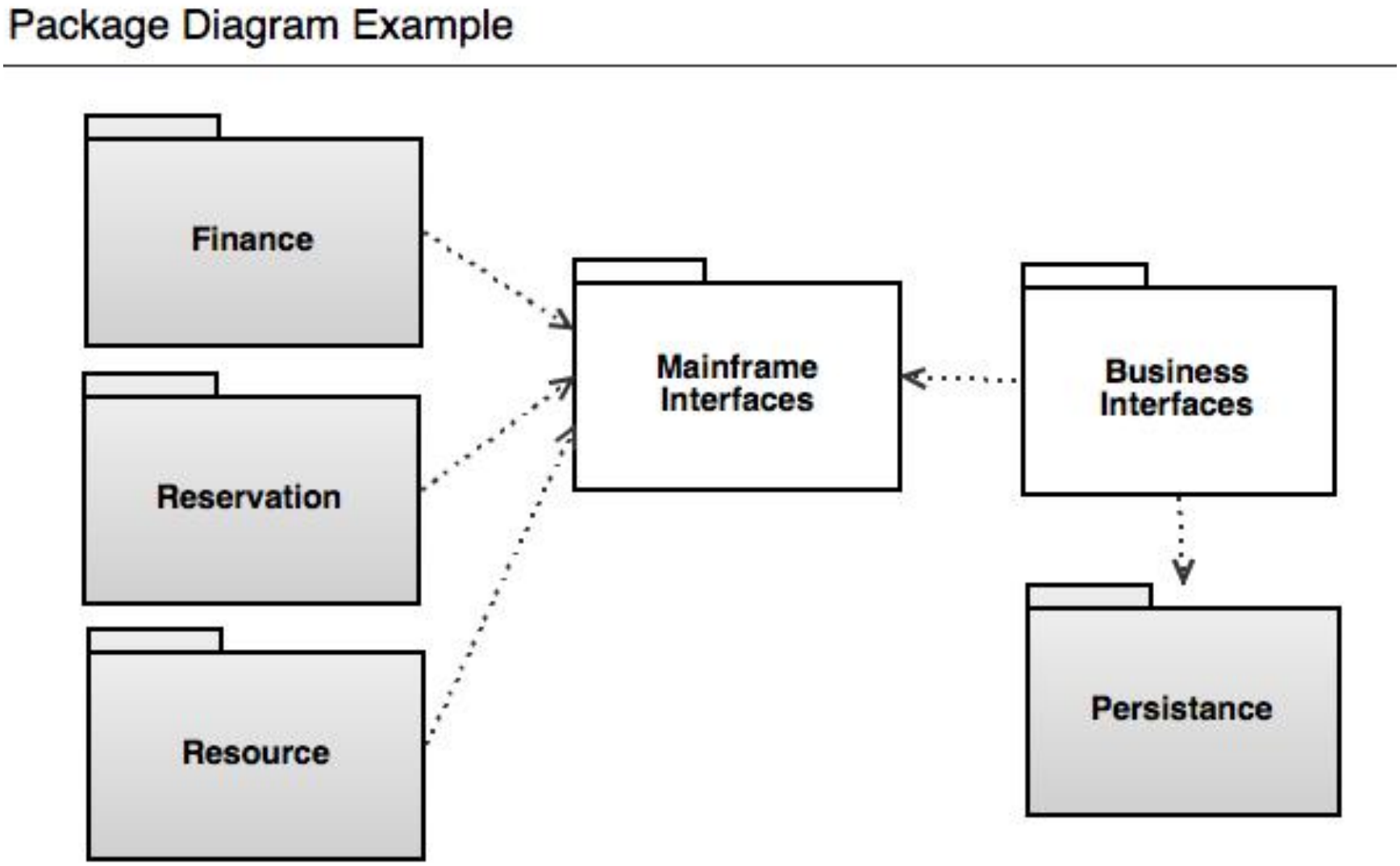
Package Notation



Package Diagrams

- Primarily used to show packages and their dependencies
- Ideally, this package dependency diagram is generated by the system itself so that you can view an accurate representation of dependencies
- Good package structure has a clear flow of dependencies
- Circular dependencies are not recommended and should be minimized
- The more dependencies coming into a package, the more stable the package interface should be

Package Diagram



Composite Structure Diagram (structural)

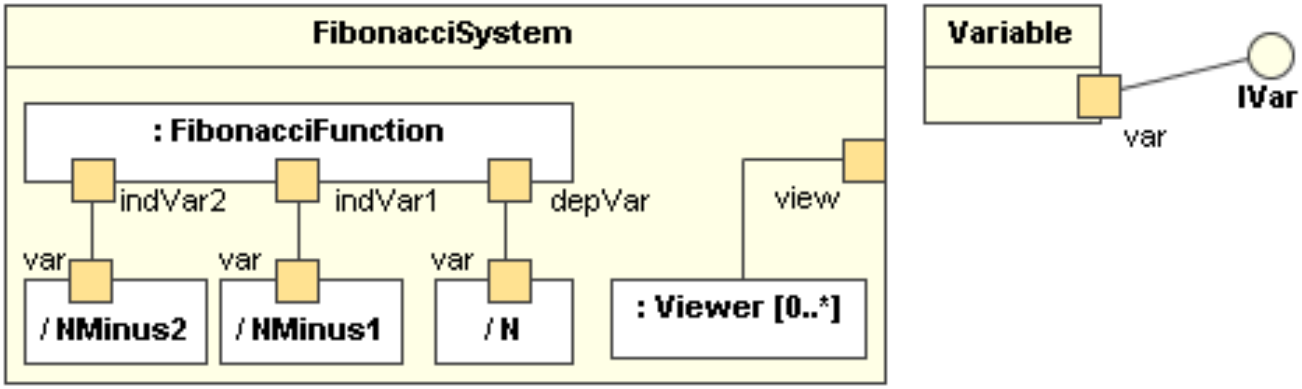
Allows complex objects to be broken down into run-time groupings

Composite Structures

- New feature for UML 2.0
- Allows you to take a complex object and break it into parts
- The difference between packages and composite structures
  - Packages are compile time groupings
  - Composite structures are run-time groupings
- This feature is new and so there isn't much information yet about how useful this is

Composite Structure Example

This example shows run-time components of a classic Fibonacci sequence:



Component Diagram (structural)

Diagram showing the structural relationships between the logical components of a system

Component Diagrams

- Main purpose is to show structural relationships between logical components of a system
- It has more to do with analyzing customer expectations & marketing decisions than the actual technology
- Customers often want to purchase or upgrade logical entities in a system separately
- Component: a logical unit connected through implemented and required interfaces
- They are integrated into the standard structure of the class diagram and often utilize notation from composite structure diagrams

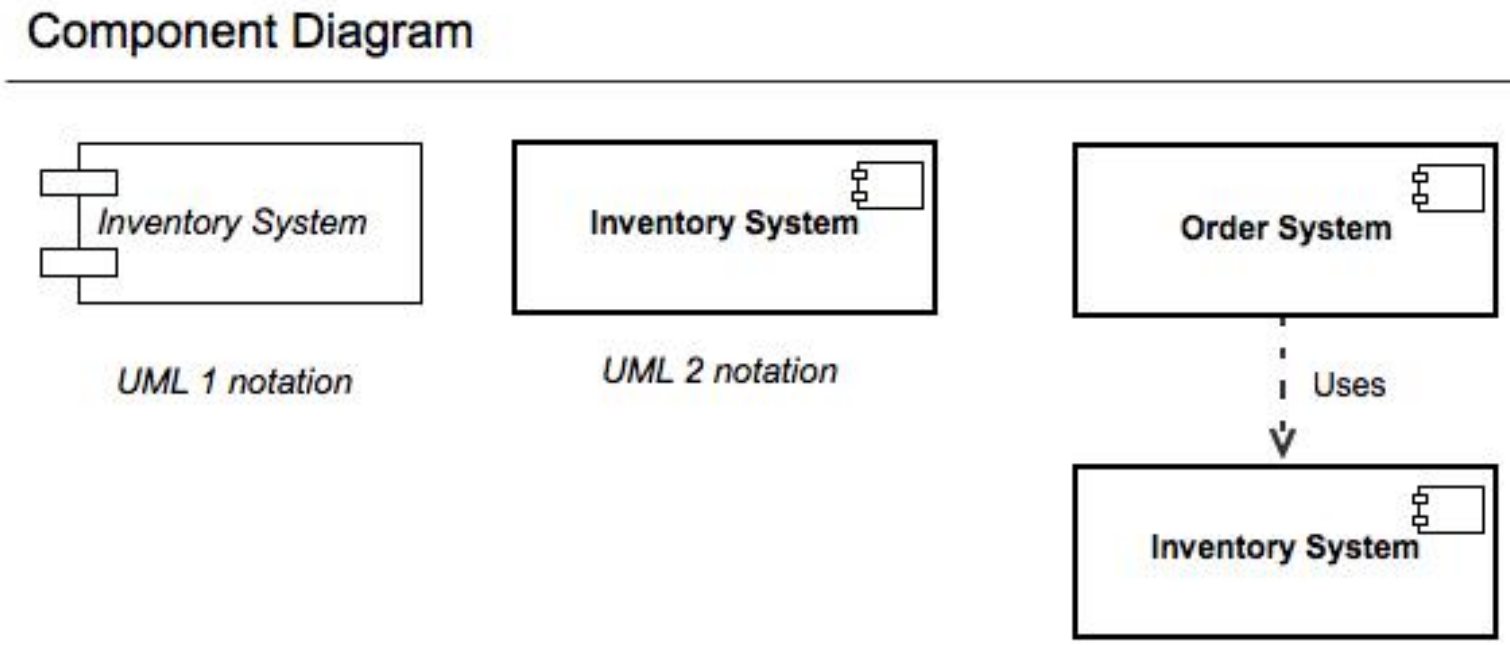
Do Components Exist?

This quote sums up confusion among technologists about what components are and suggests how component diagrams are used as part of the design:

Components are not a technology. Technology people seem to find this hard to understand. Components are about how customers want to relate to software. They want to be able to buy their software a piece at a time, and to be able to upgrade it just like they can upgrade their stereo. They want new pieces to work seamlessly with their old pieces, and to be able to upgrade on their own schedule, not the manufacturer's schedule. They want to be able to mix and match pieces from various manufacturers. This is a very reasonable requirement. It is just hard to satisfy.

Ralph Johnson http://www.cs.com/cgi/wiki?DoComponentsExist

Component Notation/Example



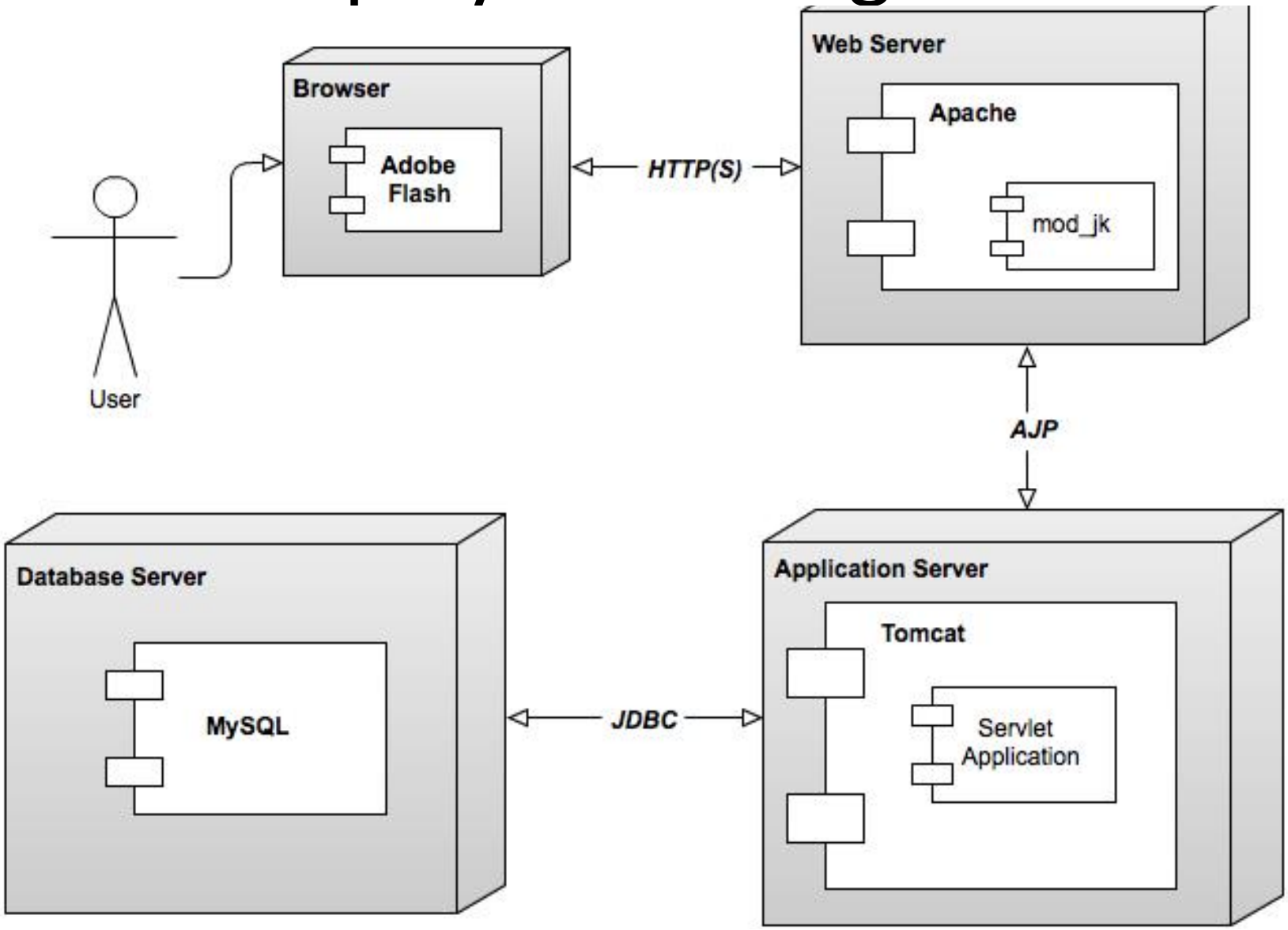
Deployment Diagram (structural)

Shows the physical layout of the hardware and software in a running system

Deployment Diagrams

- It contains nodes connected by communication paths
- Nodes are either specific hardware or execution environments that host the software
- Artifacts contained on nodes include: executables, libraries, data files, configuration files, HTML Documents…
- Including an artifact in the node indicates that it is part of the deployment in a working system

Deployment Diagram



Object Diagram (structural)

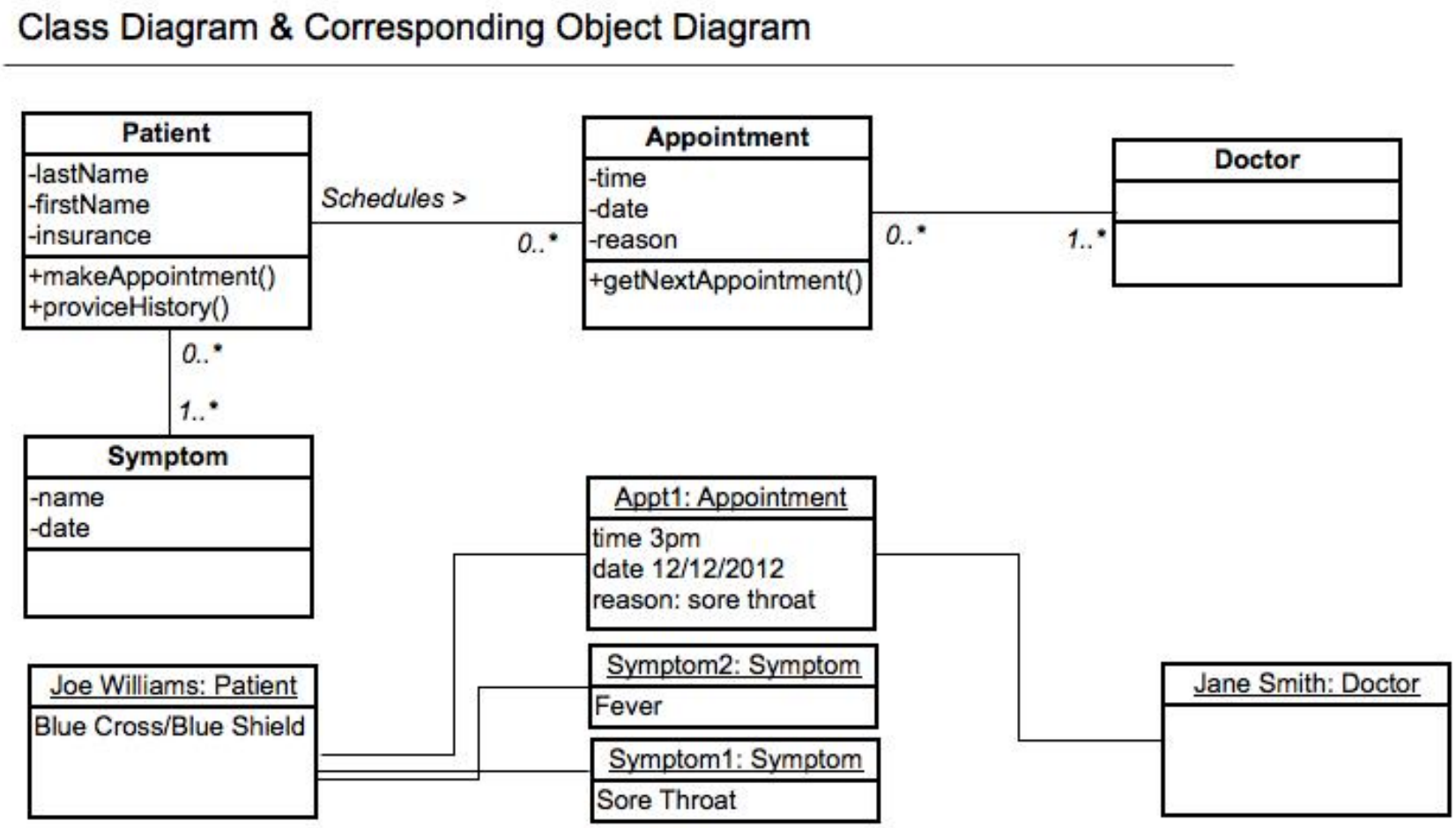
Diagram showing an overview of objects at a particular point in time – it is an example instantiation derived from the class diagram

Object Diagram

- Commonly referred to as an “instance diagram”
- Useful to show sample configurations of objects in the working system and clarifying the class diagram
- Uses instance notation:

  Instance name : class name

Object Diagram



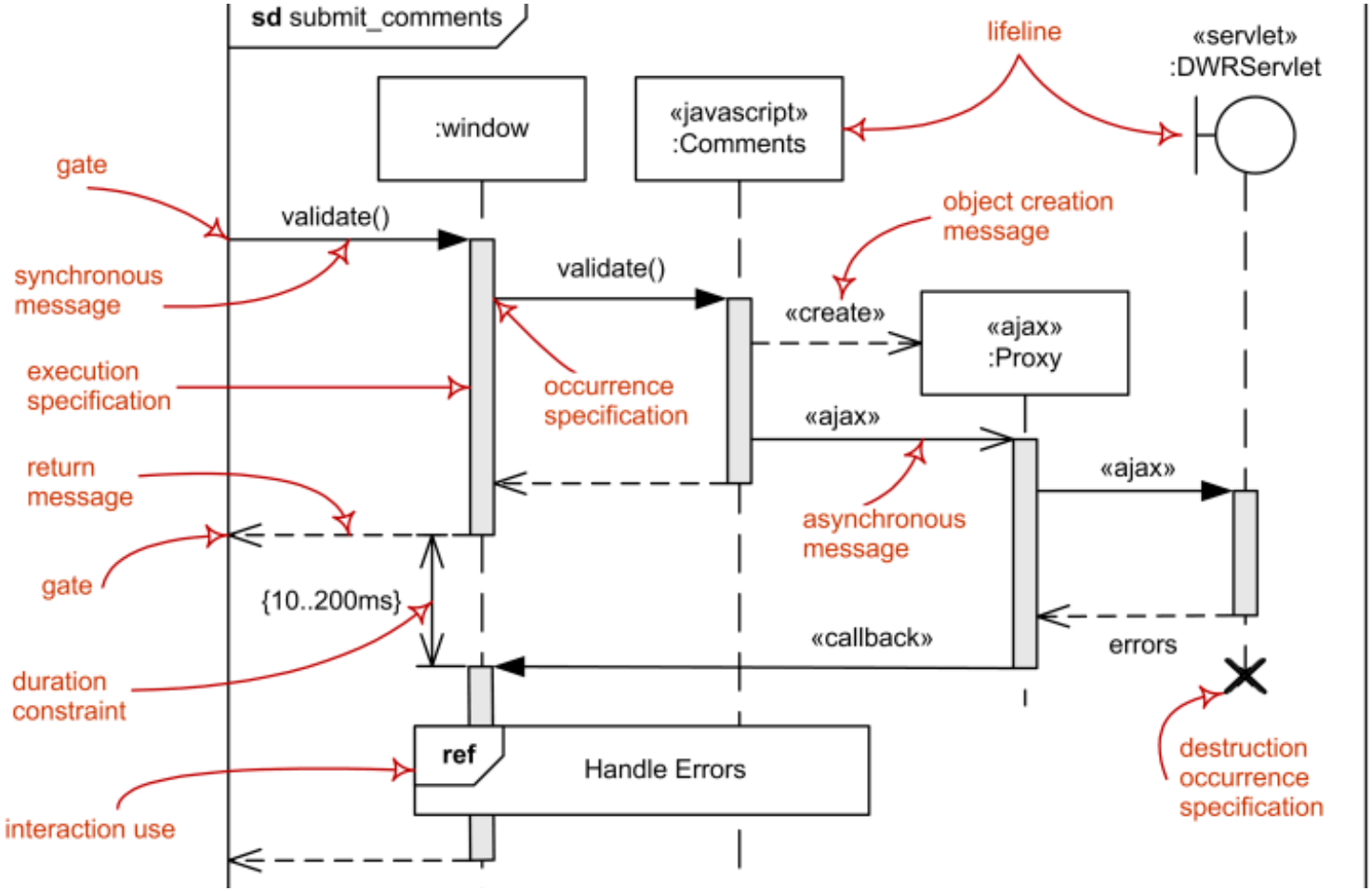
Sequence Diagram (behavioral)

A graphical description of the main success scenario described in the use cases for the system

Sequence Diagram

- Captures how objects interact with one another in a particular system scenario
- Interactions are shown along a lifeline that runs vertically from top to bottom
- It shows the instantiation of objects used in the scenario and the messages that are exchanged between them
- Creation of the object is indicated with an arrow along the timeline pointing to the box
- Deletion is indicated by an “X” crossbar when the object gets deleted

Overview of Basic Notation



When to Use Sequence Diagrams

- Useful for looking at how objects interact in a single scenario
- It provides a broad picture of interactions, but does not provide detailed definition of specific behavior within the interaction
- If you want to look at details of complex behavior or interactions across multiple scenarios, sequence diagrams will NOT work well.

Use Case Diagram (behavioral)

A graphical description of the main success scenario described in the use cases for the system

What is a Use Case?

- Use cases depict a set of scenarios organized around a common user goal
- Scenario: a sequence of steps describing an interaction between the user and the system
- Each step in the use case is a simple statement that clearly defines who is carrying out the step
- There can be multiple actors participating in a single use case (human or technological)
- Too little information is preferable to too much detail
- Overly detailed use cases don't get read whereas simple use cases & use case diagrams are useful for discussion

Example Use Case

Content:

Submit Work Report

Goal Level: Sea Level

Main Success Scenario:

1. Create a new account (if new volunteer)
2. Login
3. Create Report
    1. Enter detail fields
    2. Upload photo (optional)
4. Submit Report
5. Forward Copy to Team mates

Extensions:

1a: Review and consent to volunteer agreement

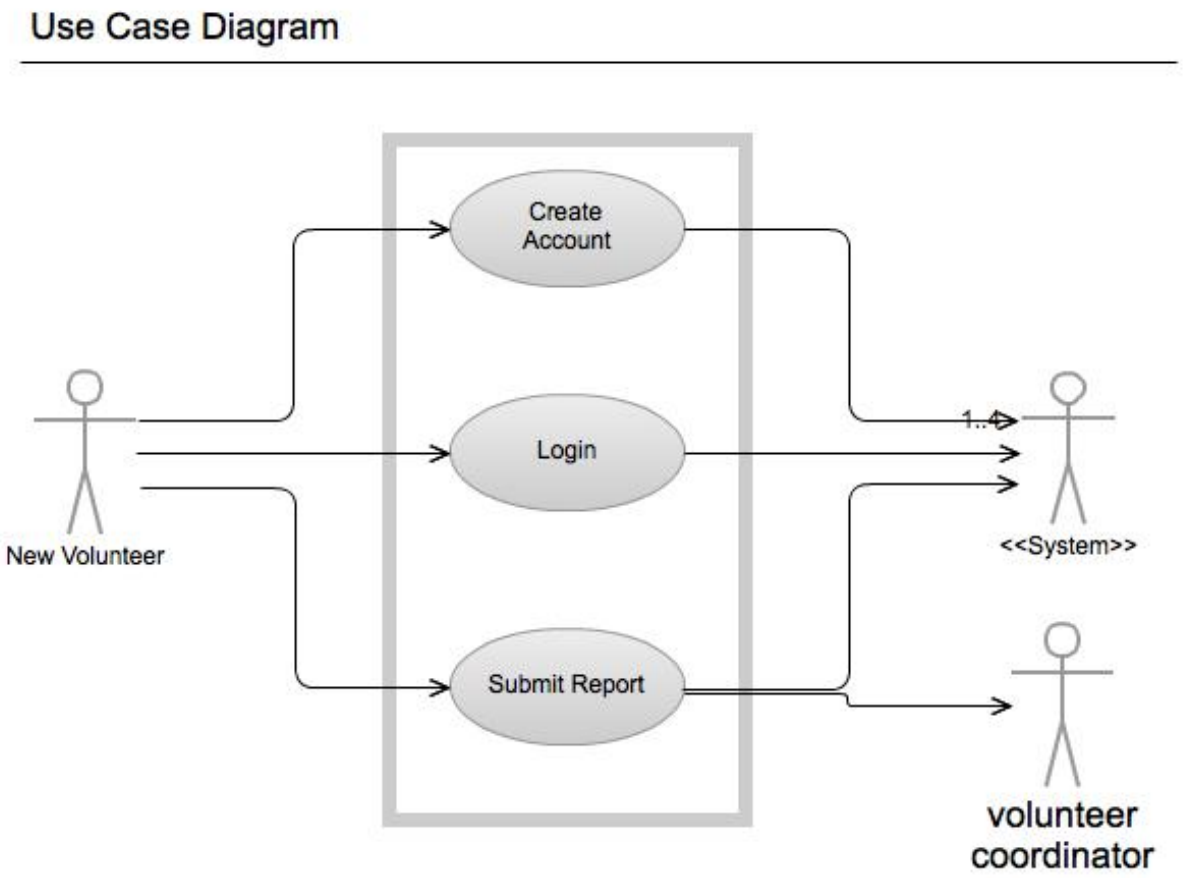
Description:

- Title
- Detail Level
- High level steps in common success scenario
- Lower level detail worth noting

What is a Use Case Diagram?

- A graphical representation of the main success scenario(s)
- It acts as a table of contents for the use case set or a high level picture of the individual use case
- A. Cockburn: Writing Effective Use Cases (2001) suggests scheme for defining level of use cases:
  - Sea Level: Primary High Level Overview Level
  - Fish Level: Detailed definition below Sea Level
  - Kite Level: How use cases fit into broader business level interactions

Example Use Case Diagram



- Three actors
  - 2 human
  - 1 system

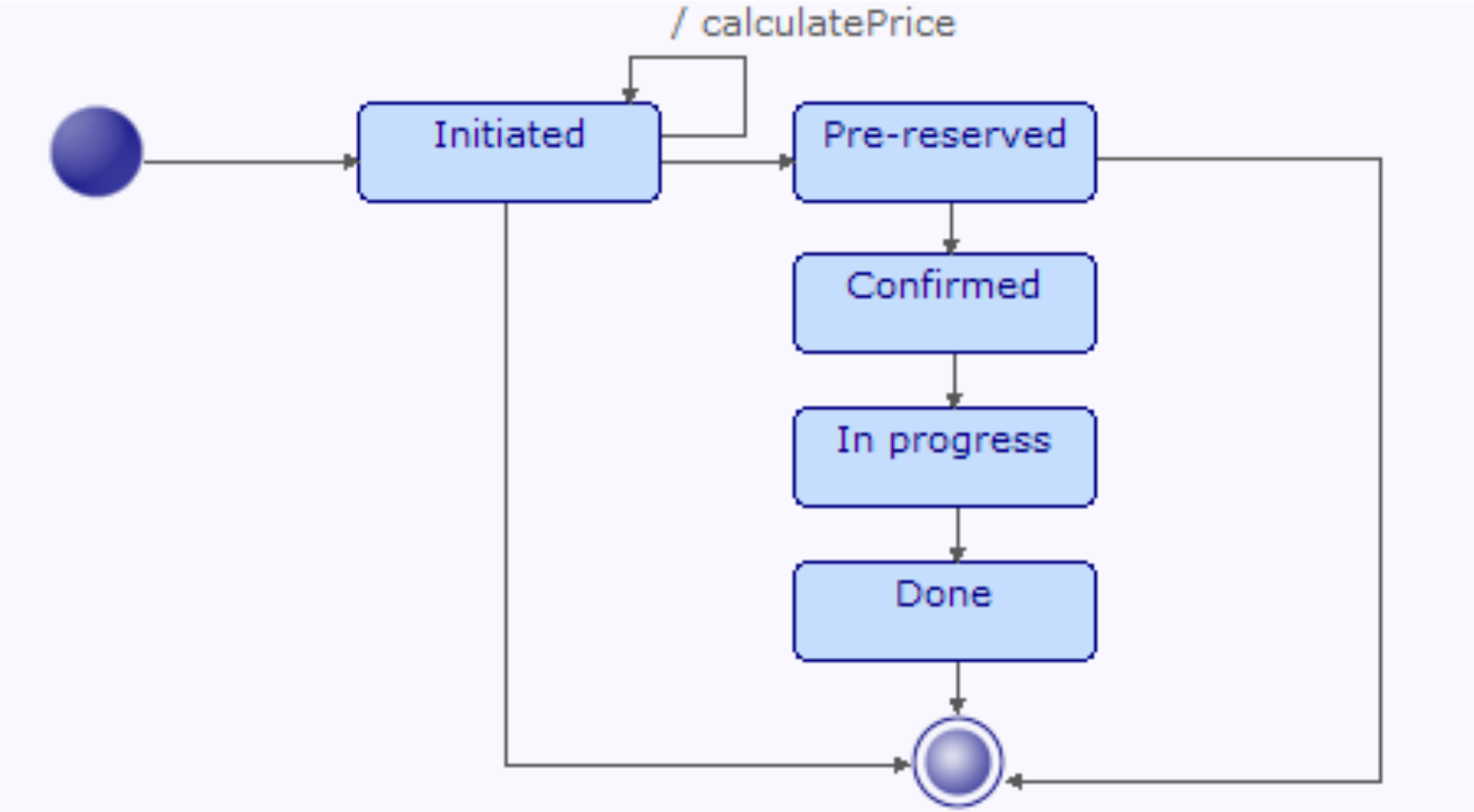
State Machine Diagram (behavioral)

In OO state machine diagrams are used for a single class to show the lifetime behavior of an object

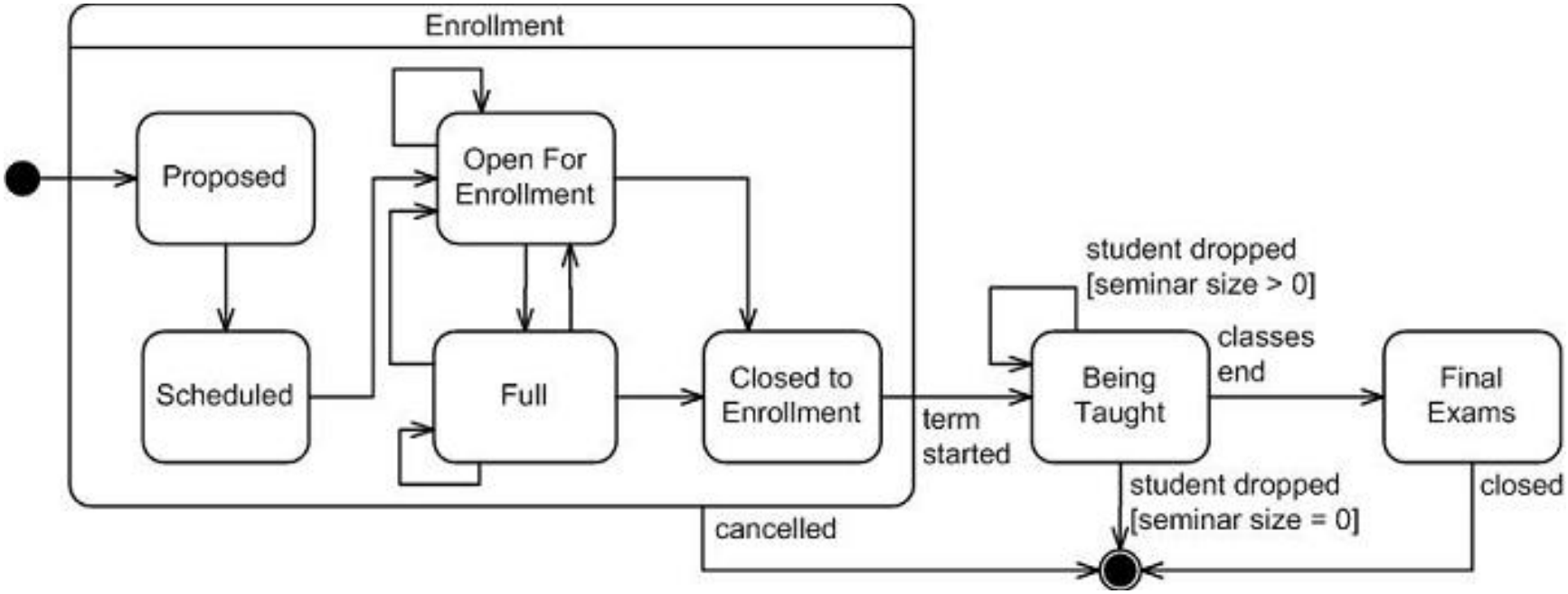
Parts of a State Machine Diagram:

- Initial pseudostate (state at creation) is indicated using a solid dot and arrow
- Arrow transition indicates movement to another state
- Transition label and its parts are optional (trigger-signature [guard]/activity)
- States can react to internal events as well (putting the trigger event, guard, & activity in the state itself)
- States can indicate entry or exit activities within the state definition
- States can be idle or occupied in some ongoing work
- Titled boxes can be used to indicate a super-state
- End-state of the object is indicated by solid dot with circle

Example State Diagram



Example with Super-state



Enrollment: Super-state definition

When to Use State Diagrams

- Useful for describing the behavior or an object across several use cases
- It is useful for showing the lifecycle of an individual object
- It is NOT useful for describing objects that collaborate with one another
- It can be combined with interaction and activity diagrams to provide useful detail

Activity Diagram(behavioral)

Used to describe procedural logic, business process, and work flow. They are particularly useful for documenting parallel behavior

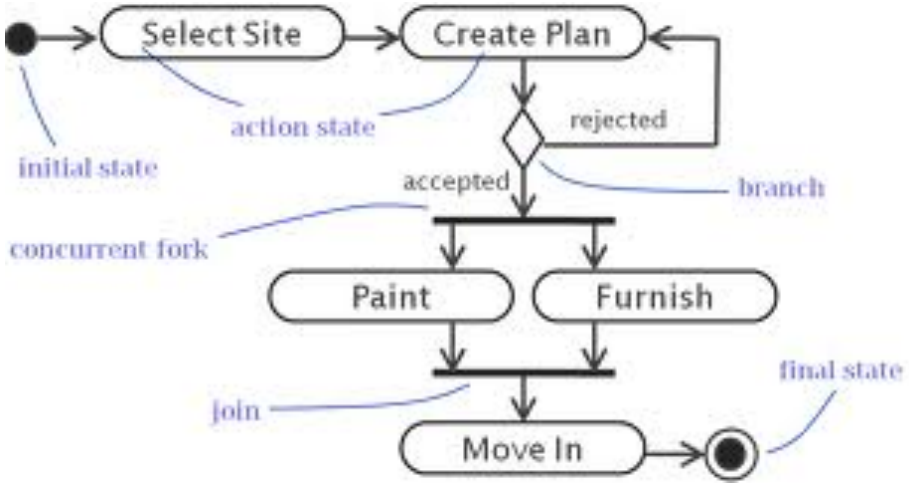
Activity Diagram - Overview

- Significantly extended and modified between UML 1.x and UML 2.x
- In UML 1.x they were positioned as a special case of a state diagram, In UML 2.0 this distinction was removed
- It is particularly useful for documenting concurrent algorithms and parallel processing
- They aren't used much, but may be particularly useful where there are instances of parallel processing
- They are sometimes used to document use cases, but they may not be the best way to communicate with domain experts

Activity Diagram – Components

- Action: each box in activity diagram represents an action
- Initial node: indicated with the dot/arrow notation of state diagrams
- Fork: one incoming flow and several outgoing flows
- Join: used to synchronize flow when parallel actions are complete
- Decision: conditional behavior delineated by the decision
- Merge: marks the end of the conditional behavior

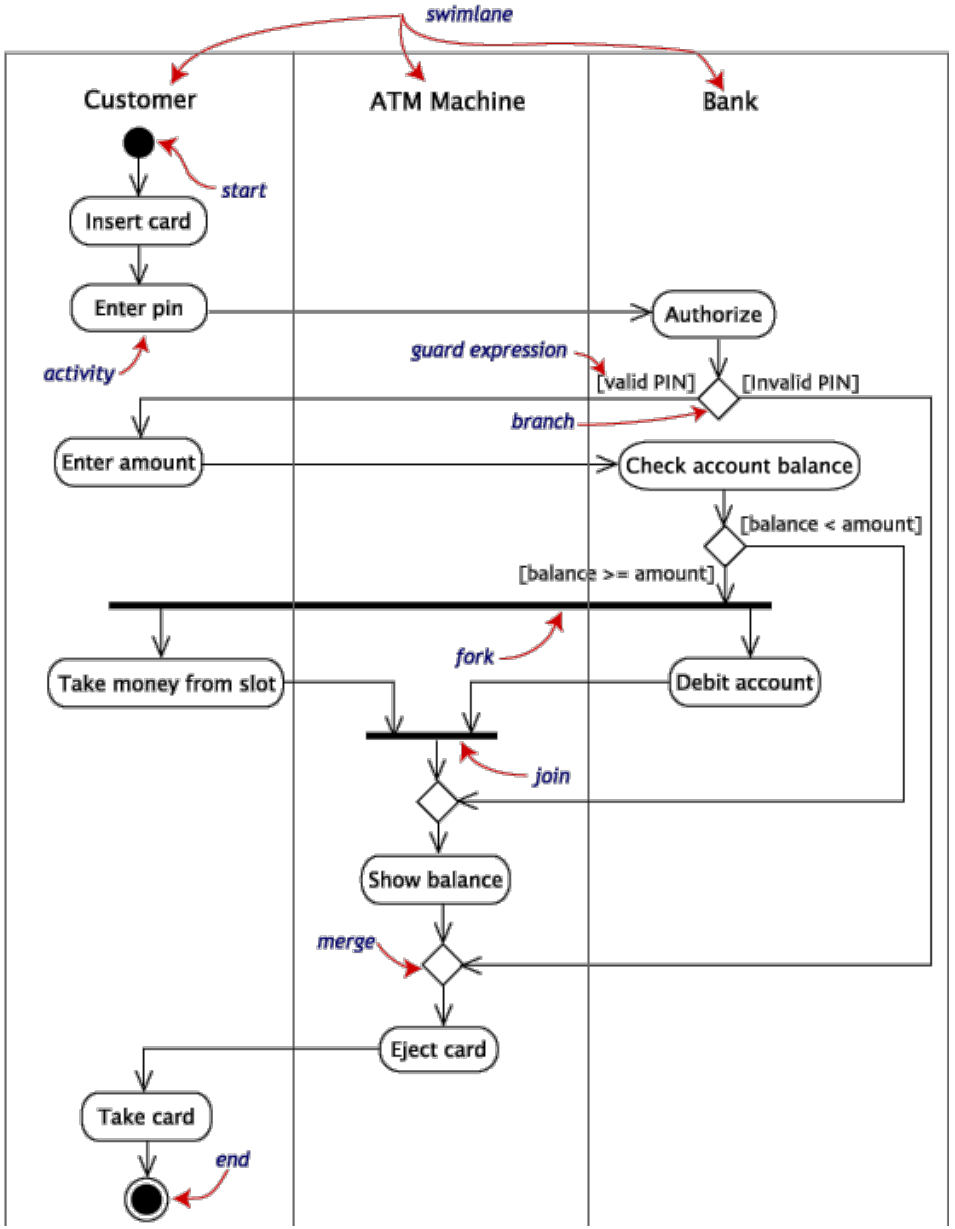
Example Activity Diagram



Partitions (Swim Lanes)

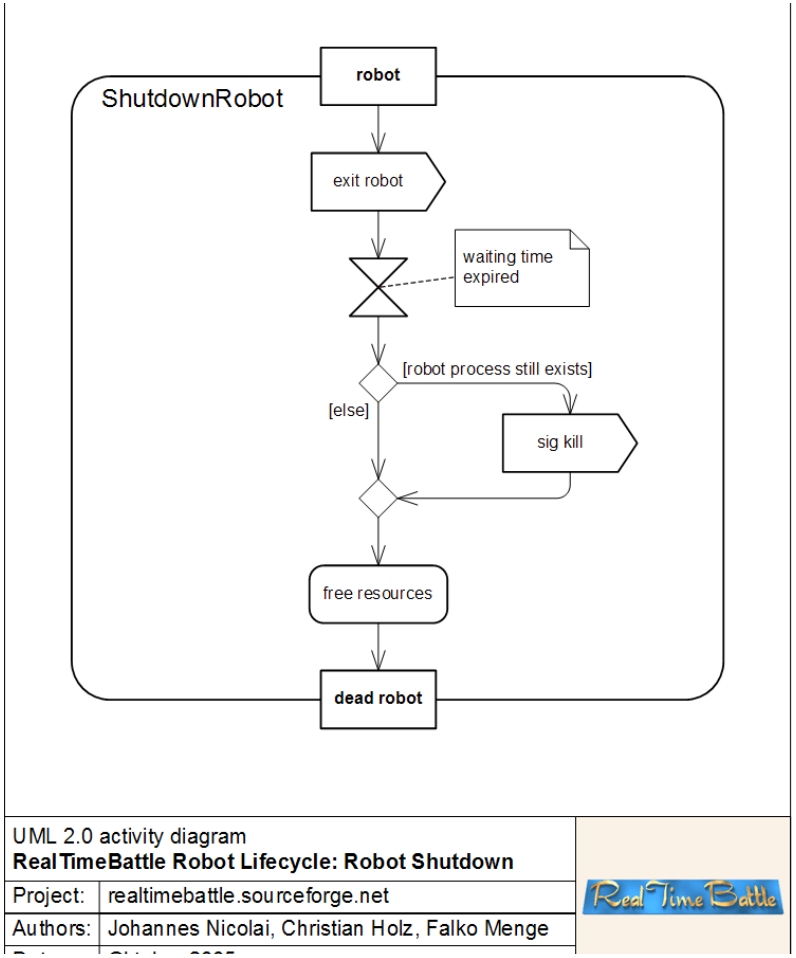
Partitions can be used to show the distributions of activities across multiple actors

The can be a little more difficult to read, but useful if the purpose of the diagram is to show who specifically does what action



Additional features

- Hourglass symbol can be used to indicate timing elements such as dates or timers
- Send Signals (e.g. exit robot)
- Activities can be broken down into sub-activities using box notation (e.g. ShudownRobot)

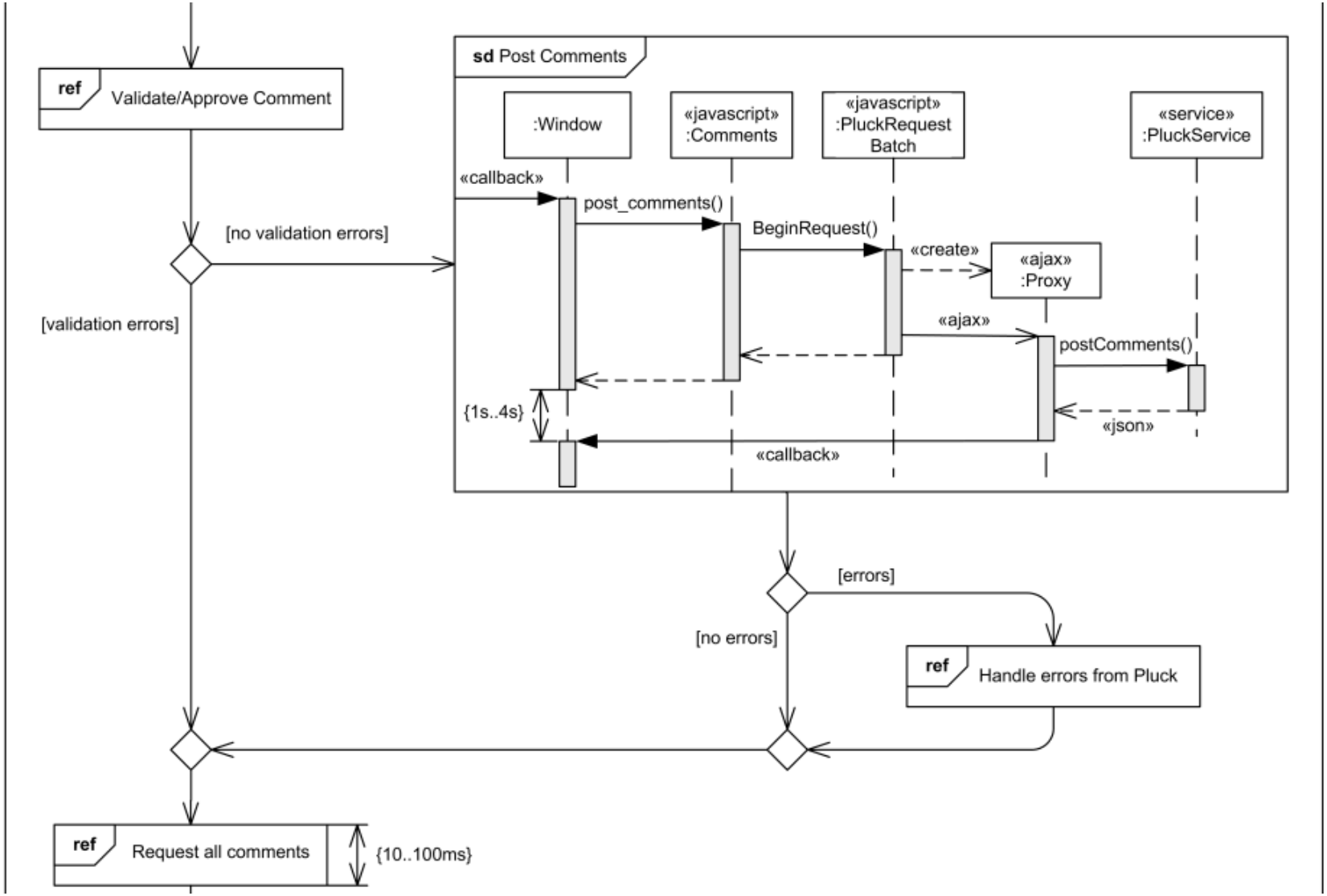


Interaction Overview Diagram (behavioral/interaction)

A merging of activity diagrams and sequence diagrams

Activity boxes replaced by sequence diagrams where more detail is needed

Example of Interaction Overview:



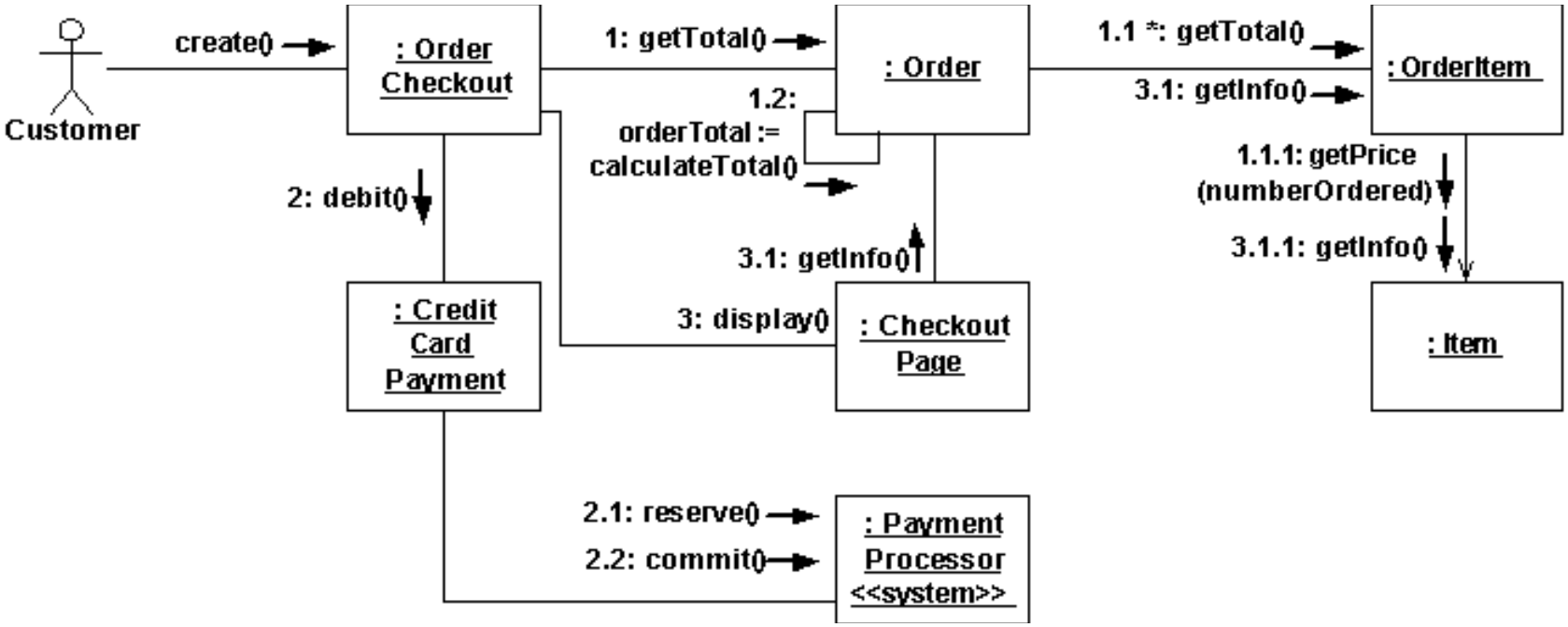
Communication Diagram (behavioral/interaction)

Emphasizes the data links between participants in a particular interaction

Communication Diagram

- Similar to sequence diagrams without lifeline and sequence of messages
- Communication diagrams use simple line notation and numbering scheme to show sequence
- Transient links can be noted using <<local>>, <<parameter>> and <<global>> (not in UML2, but still in common usage)
- Most people prefer sequence diagrams, but useful for simple white-board discussion or when you want to focus on links

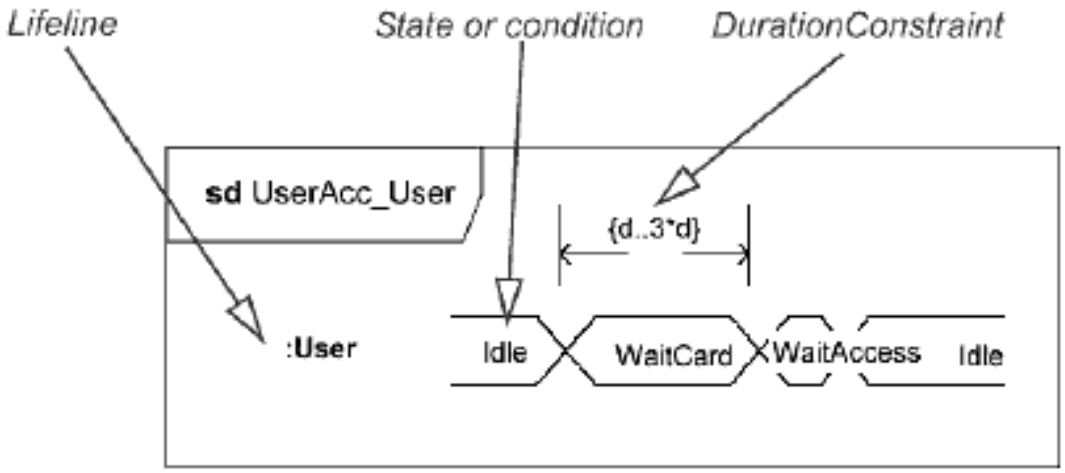
Example Communication Diagram



Timing Diagram (behavioral/ interaction)

Diagram focusing on timing constraints for one or a group of related objects

Example Timing Diagram



Crossed lines indicate a change of state

Timing diagrams are not commonly used

Additional Resources for UML and Object-Oriented Design:

Fowler, M. UML Distilled (Third Edition, 2004) – good overview of UML language

Larman, C. Applying UML and Patterns (2d ed 2001)

Rumbaugh, J., Jacobson, I. and G. Booch The Unified Modeling Language Reference Manual (1999)

Dennis, A., Wixon, B. H., and D. Tegarden System Analysis Design: UML Version 2.0 An Object Oriented Approach (2009)