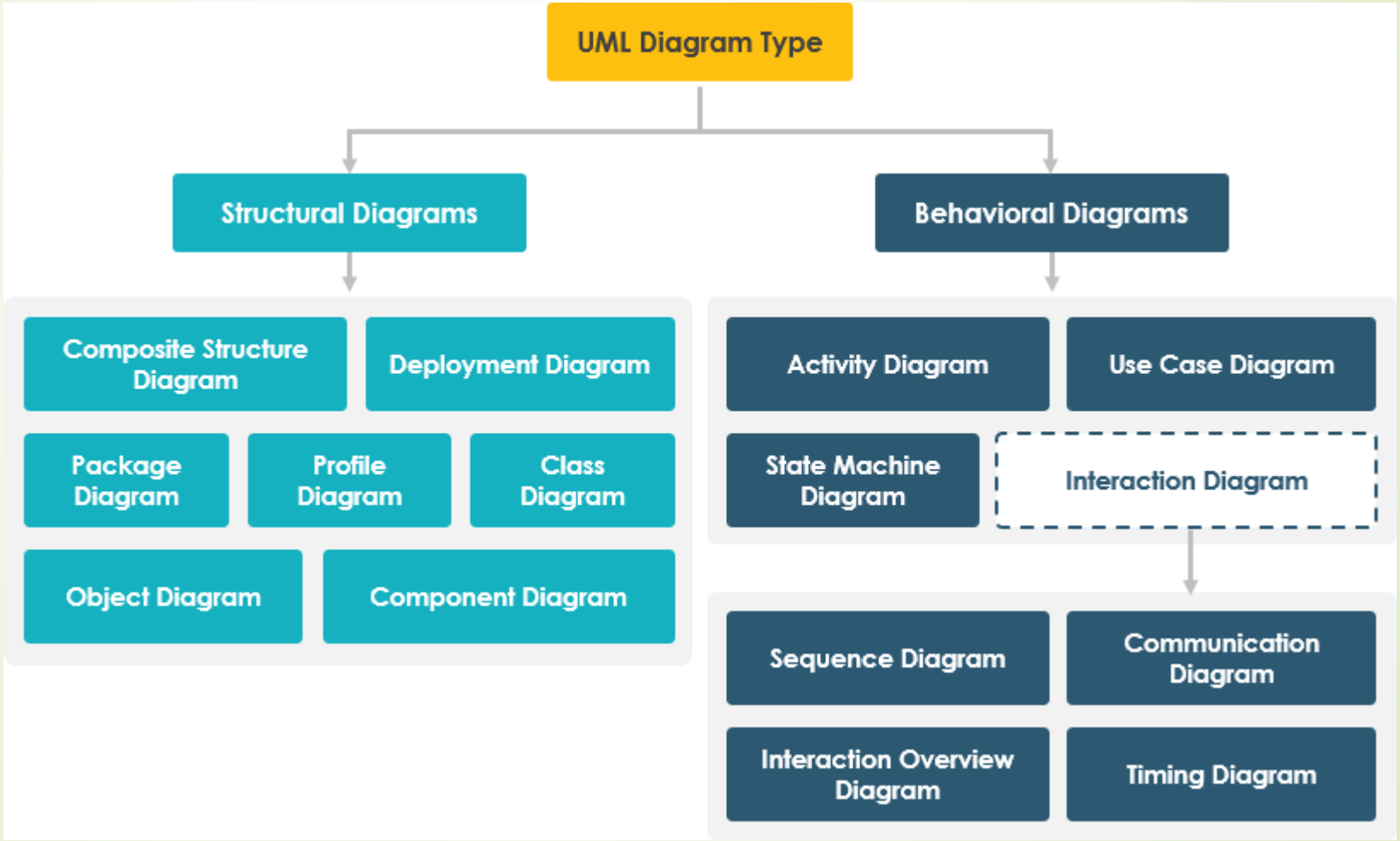
Unified Modeling Language

Lecturer: Adel Vahdati

UML

- The Unified Modeling Language (UML) is a standard language for **specifying**, **visualizing**, **constructing** and **documenting** the artifacts of software systems, as well as for business modeling and other non-software systems.
- The UML represents a collection of best modeling practices that have proven successful in the modeling of large and complex systems

UML 2 Diagram Superstructure



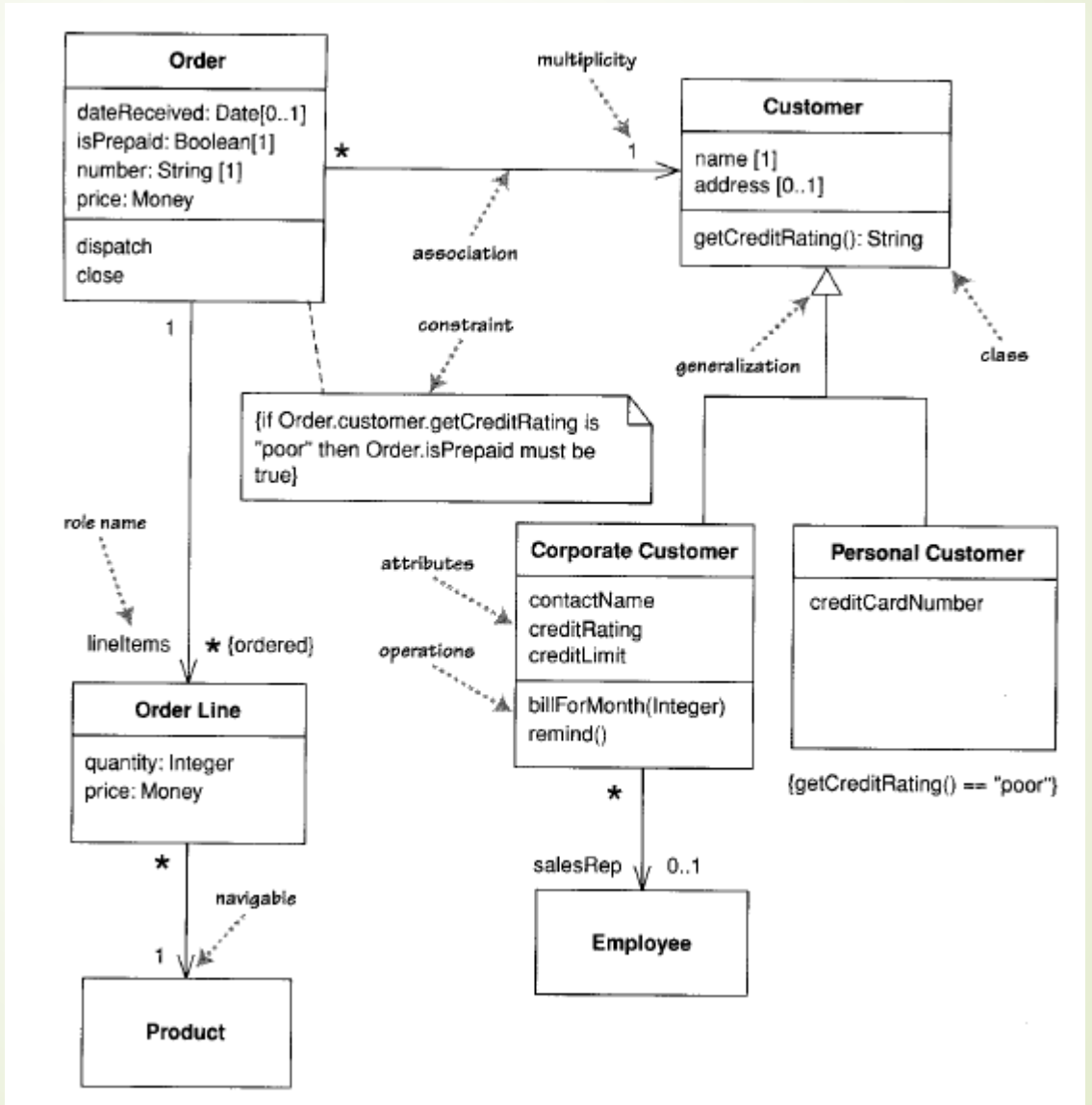
UML Diagrams

- UML 2.0 defines thirteen types of diagrams, divided into three categories:
- **Structure Diagrams**
  - Class Diagram
  - Object Diagram
  - Component Diagram
  - Composite Structure Diagram
  - Package Diagram
  - Deployment Diagram
- **Behavior Diagrams**
  - Use Case Diagram
  - Activity Diagram
  - State Machine Diagram.
- **Interaction Diagrams,** all derived from the more general Behavior Diagram
  - Sequence Diagram,
  - Communication Diagram,
  - Timing Diagram
  - Interaction Overview Diagram.

Class Diagrams

- Describes the types of objects in the System and the various kinds of static relationships that exist among them .
- Show the properties and operations of a Class and the constraints that apply to the way objects are connected .

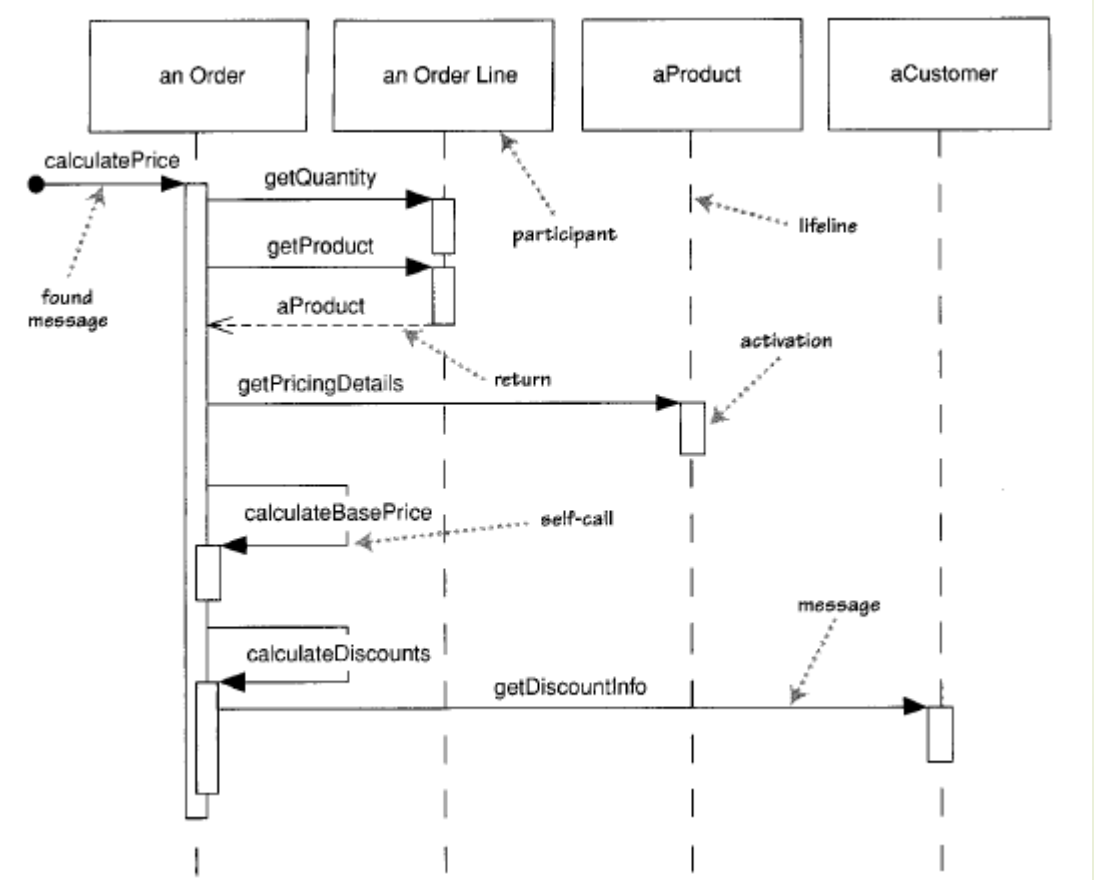
Class Diagrams



Sequence Diagrams

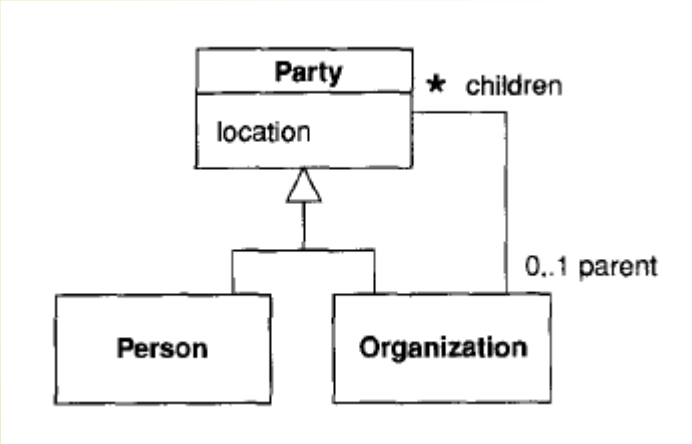
- Interaction diagrams describe how groups of objects collaborate in some behavior.
- A sequence diagram captures the behavior of a single scenario.
- The diagram shows a number of example objects and the messages that are passed between these objects within the use case .

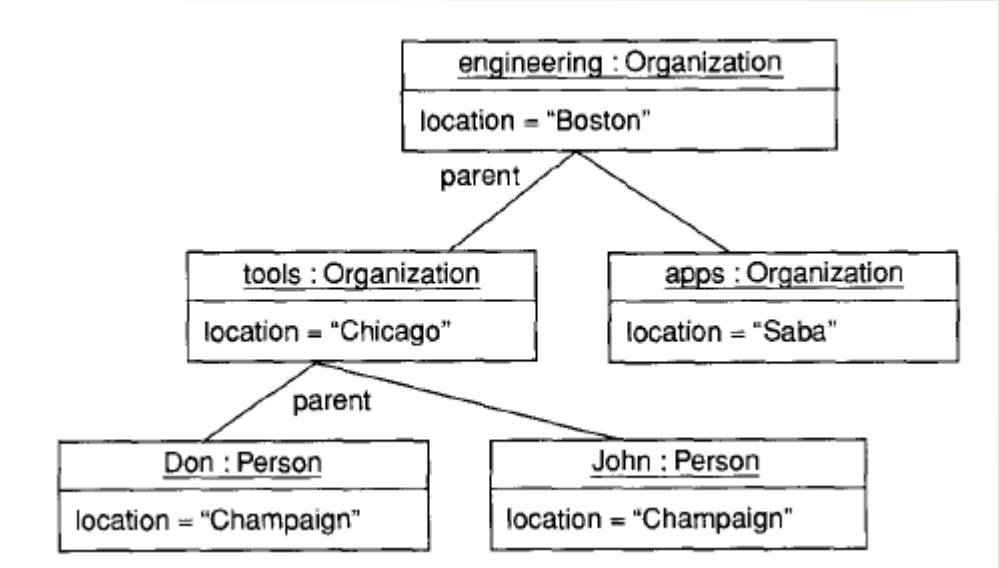
Sequence Diagrams



Object Diagrams

- An object diagram is a snapshot of the objects in a System at a point in time.
- Because it Shows instances rather than classes, an object diagram is often called an instance diagram.

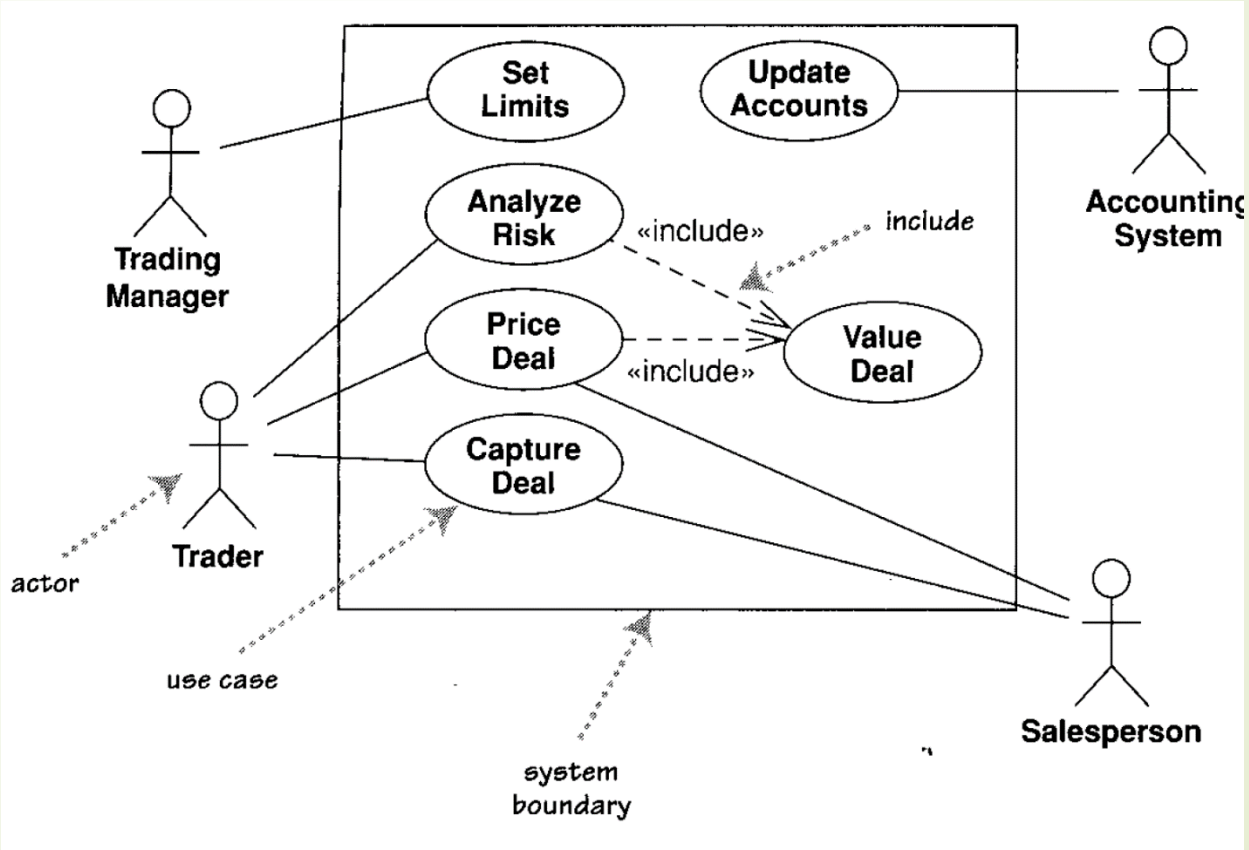




Use Cases

- A technique for capturing the functional requirements of a System.
- Describing the typical interactions between the users of a System and the System itself
- Providing a narrative of how a System is used.
- **A scenario** is a sequence of steps describing an interaction between a user and a System .
- Each use case has a **primary actor**, which calls an the System to deliver a service.
  - The actor with the goal the use case is trying to satisfy and is
  - Usually, but not always, the Initiator of the use case.
- There may be other actors as well with which the System communicates while carrying out the use case . These are known as **secondary actors** .

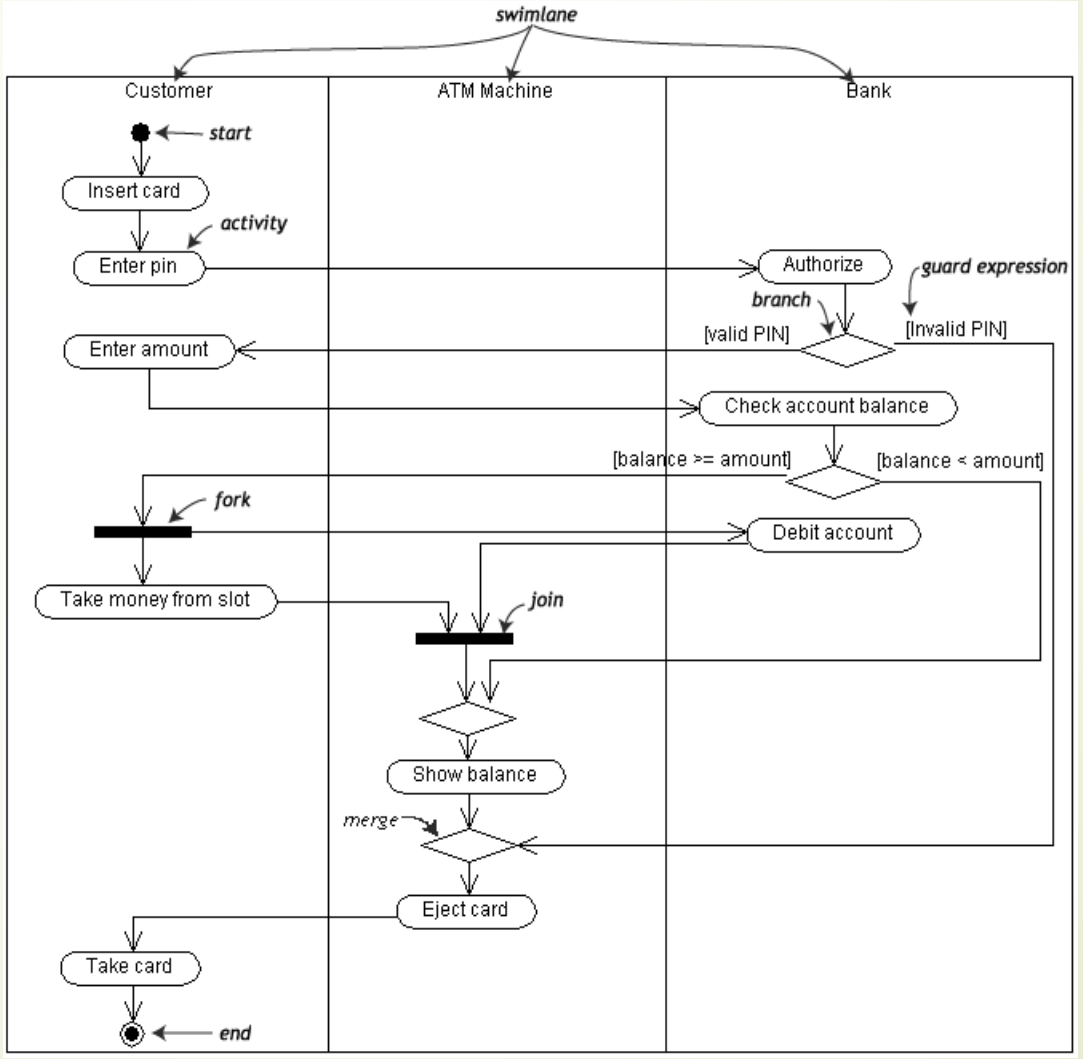
Use Cases Diagrams



Activity Diagrams

- Activity diagrams are a technique to describe procedural logic, business process, and work flow.
- The principal difference between them and flowchart notation is that they Support Parallel behavior.

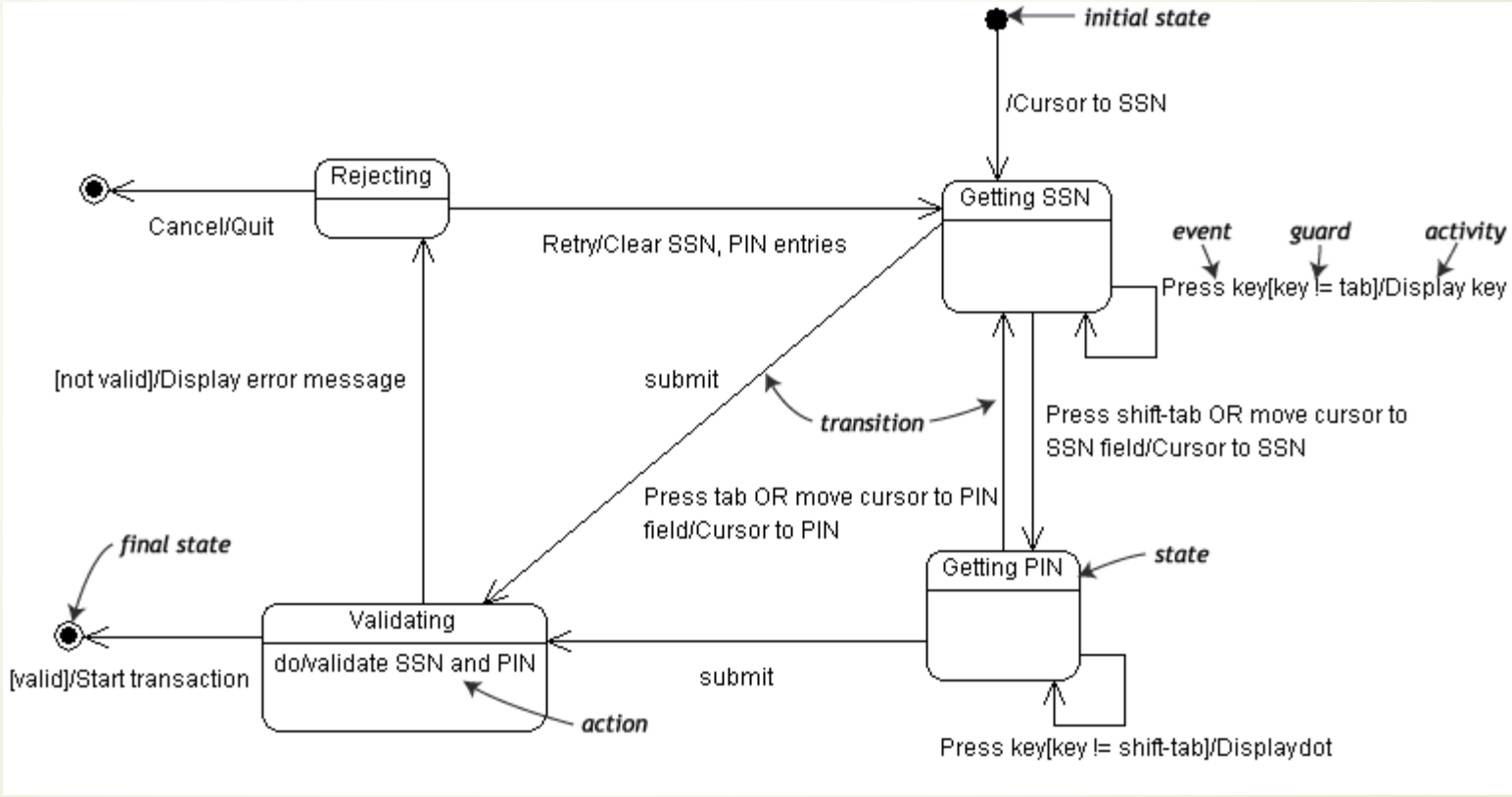
Activity Diagrams



State Machine Diagrams

- Stare diagrams are good at describing the behavior of an object across several use cases.
- The transition indicates a movement from one State to another
- Each transition has a Label that comes in three parts : **trigger-signature** **[guard]**/**activity**. All the parts are optional
  - The **trigger-signature** is usually a single event that triggers a potential change of State
  - The **guard**, if present, is a Boolean condition that muss be true for the transition to be taken
  - The **activity** is some behavior that's executed during the transition to be

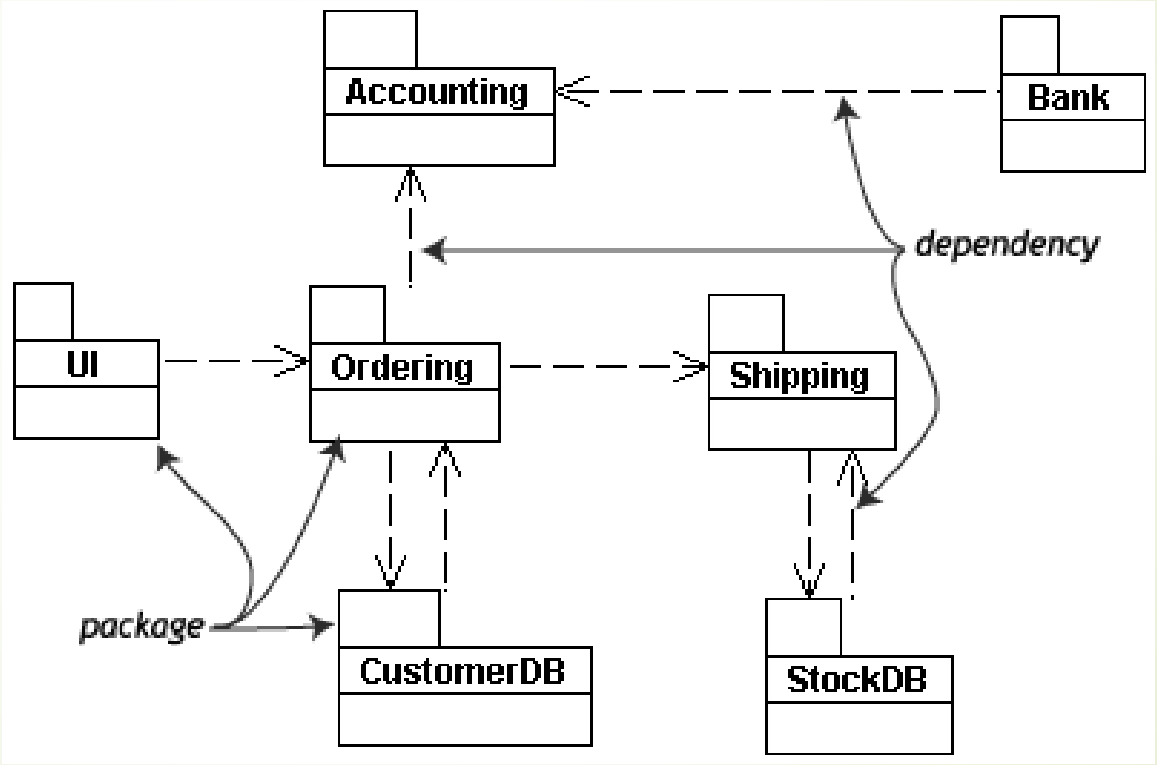
State Machine Diagrams



Package Diagrams

- A package is a grouping construct that allows you to take any construct in the UML and group its elements together into higher-level units .
- Its most common use is to group classes
- Each package represents a namespace, which means that every class must have a unique name within its owning package .
- The UML allows classes in a package to be public or private.
  - A public class is part of the interface of the package and can be used by classes in other packages;
  - a private class is hidden

Package Diagrams



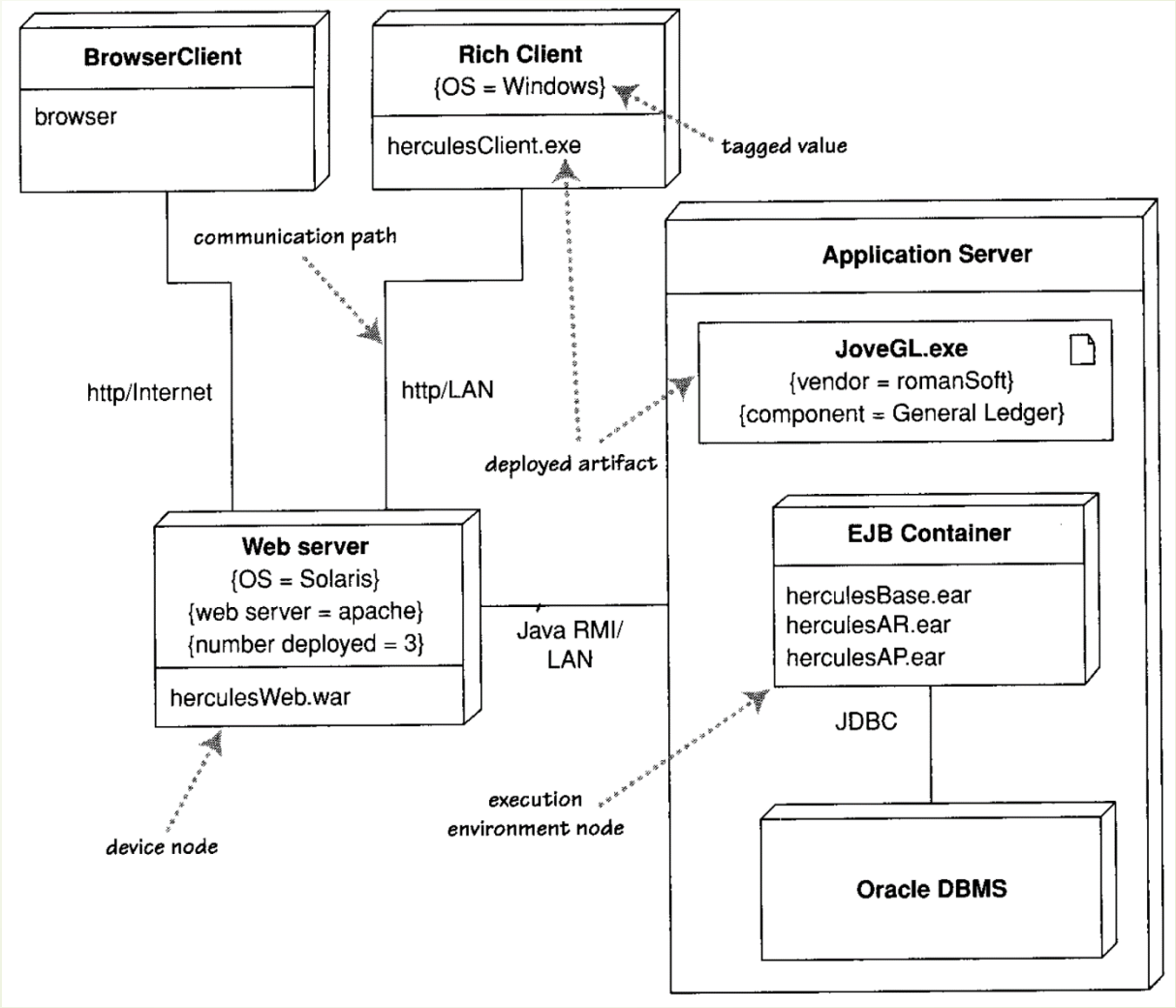
Deployment Diagrams

- Deployment diagrams show a system's physical layout, revealing which pieces of software run an what pieces of hardware.
- The main items an the diagram are nodes connected by communication paths.
- A node is something that can host some software. Nodes come in two forms :
  - A **device** is hardware, it may be a Computer or a simpler piece of hardware connected to a system .
  - An **execution environment** is software that itself hosts or contains other software, examples are an operating System or a Container process .

Deployment Diagrams (2)

- The nodes contain artifacts, which are the physical manifestations of software:
  - Files: These files might be executables,
  - Data files, Configuration files,
  - HTML documents
- Listing an artifact within a node Shows that the artifact is deployed to that node in the running System.

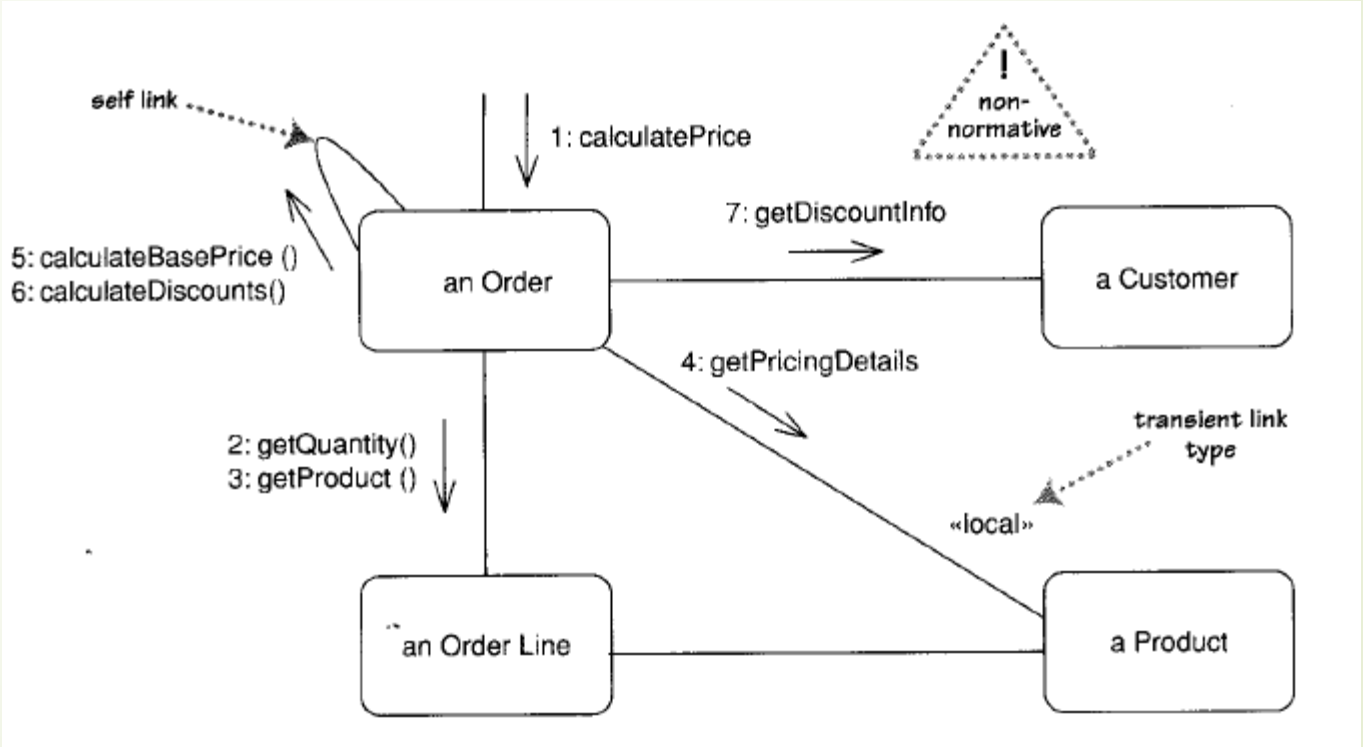
Deployment Diagrams (3)



Communication Diagrams

- Communication diagrams, a kind of interaction diagram, emphasize the data links between the various participants in the interaction.
- The main question with communication diagrams is when to use them rather than the more common sequence diagrams. A strong part of the decision is personal preference

Communication Diagrams

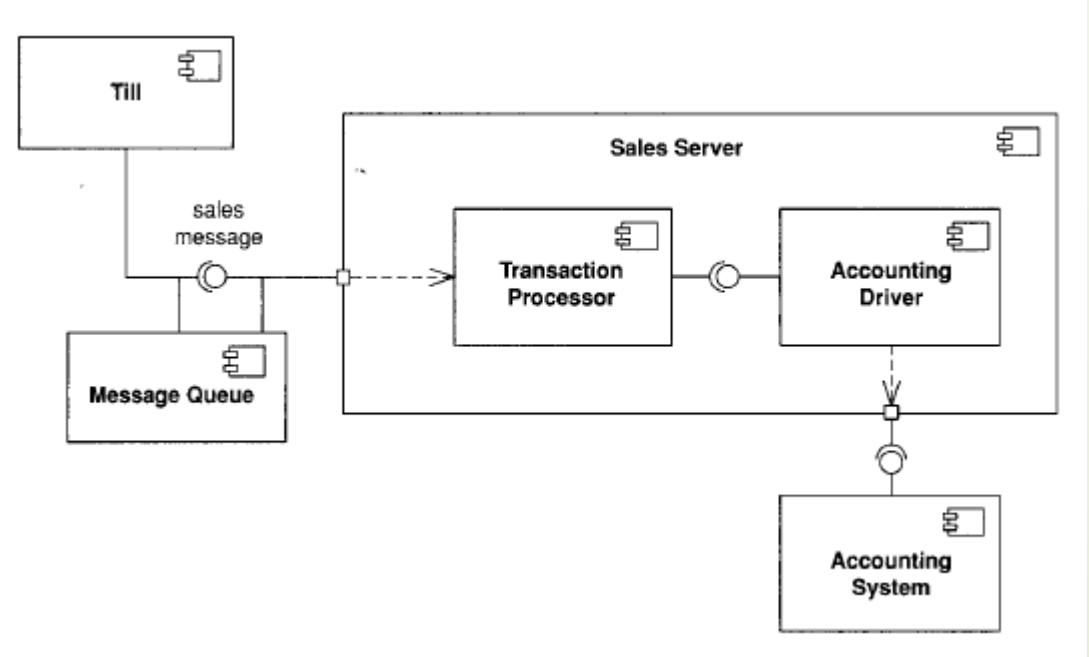


Composite Structures

- This allows you to take a complex object and break it down into parts .
- A good way of thinking about the difference between packages and composite structures is that **packages are a compile-time grouping**, while **composite structures show runtime groupings**
- They are a natural fit for showing components and how they are broken into parts; hence, much of this notation is used in component diagrams.

Component Diagrams

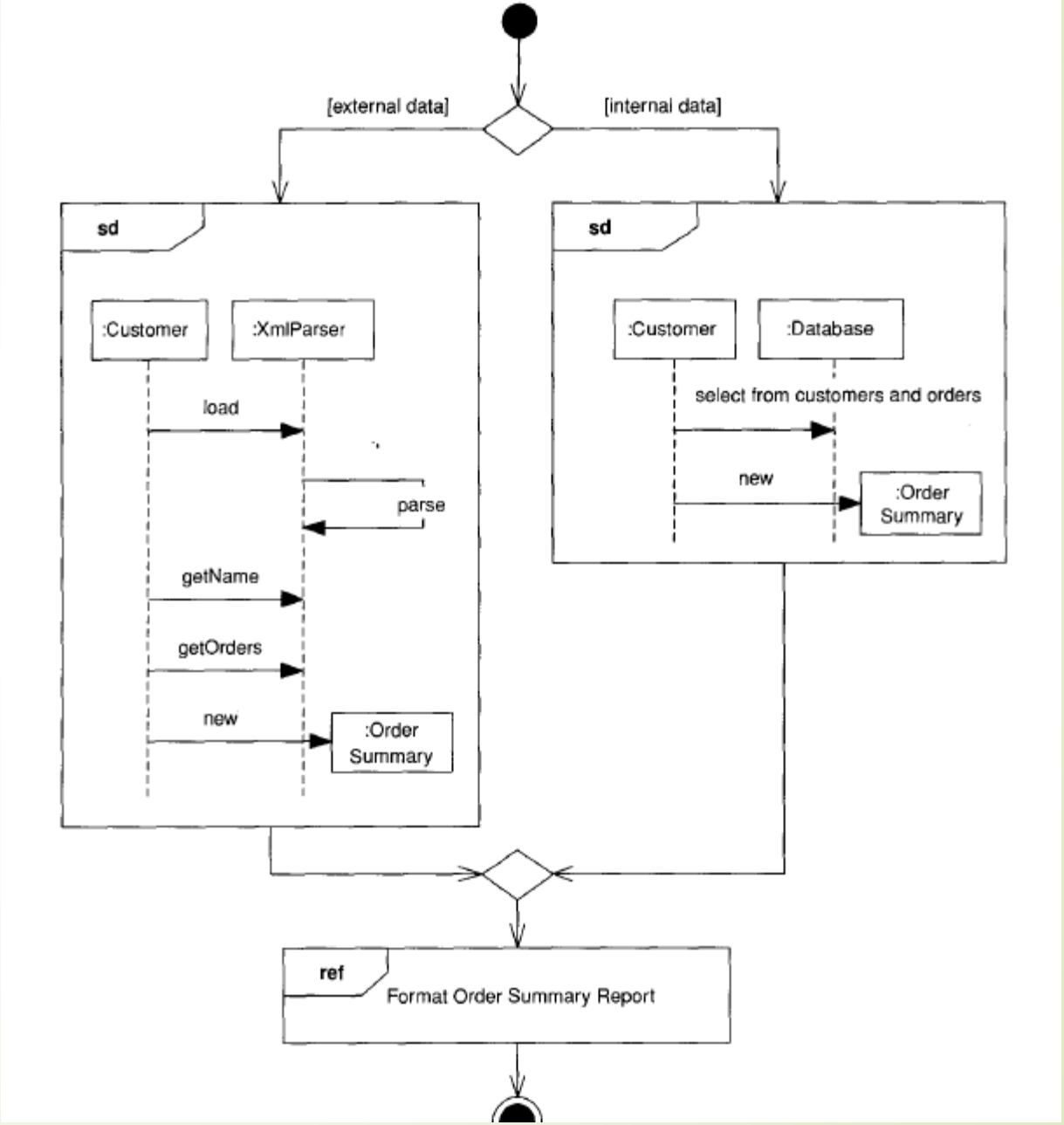
- Use component diagrams when you are dividing your System into components and want to Show their interrelationships through Interfaces or the breakdown of components into a lower-level structure .



Interaction Overview Diagrams

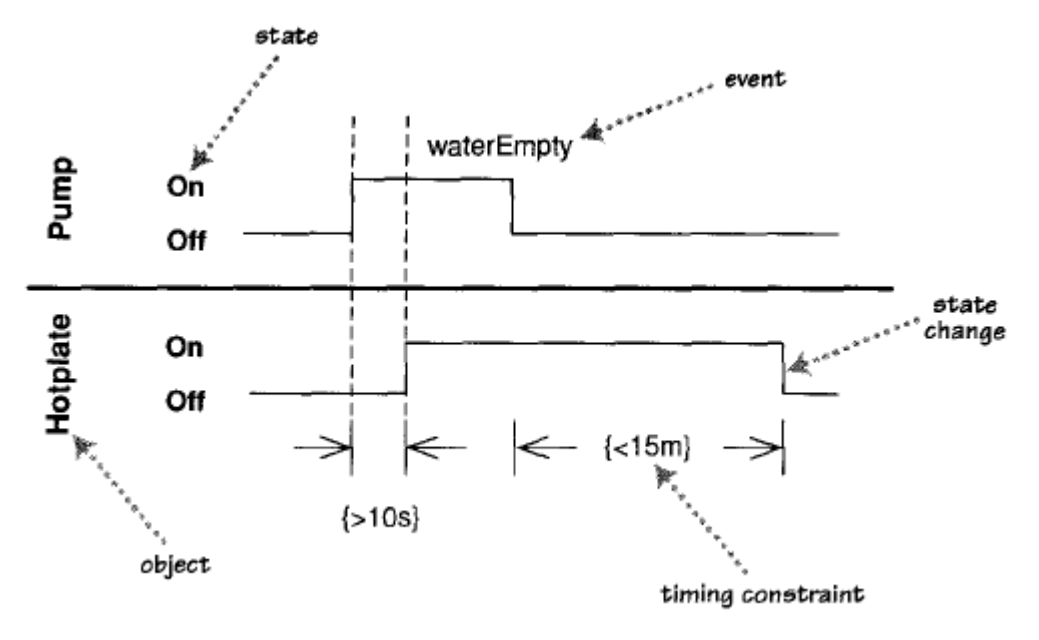
- Interaction overview diagrams are a grafting together of activity diagrams and sequence diagrams .
- You can think of interaction overview diagrams either as
  - activity diagrams in which the activities are replaced by little sequence diagrams
  - a sequence diagram broken up with activity diagram notation used to show control flow.

Interaction Overview Diagrams



Timing Diagrams

- Timing diagrams are useful for showing timing constraints between state changes an different objects .



Fitting the UML into a Process – Requirements Analysis

- Figure out what the users and customers of a software effort want the System to do.
- A number of UML techniques can come in handy here :
  - **Use cases**, which describe how people interact with the System .
  - A **class diagram** drawn from the conceptual perspective, which can be a good way of building up a rigorous vocabulary of the domain .
  - An **activity diagram**, which can Show the work flow of the organization, showing how Software and human activities interact . An activity diagram can Show the context for use cases and also the details of how a complicated use case works
  - A **state diagram**, which can be useful if a concept has an interesting life cycle, with various states and events that change that state .

Fitting the UML into a Process – Design

- When you are doing design, you can get more technical with your diagrams .
- You can use more notation and be more precise about your notation .
  - **Class diagrams** from a software perspective . These show the classes in the software and how they interrelate .
  - **Sequence diagrams** for common scenarios . A valuable approach is to pick the most important and interesting scenarios from the use cases and use CRC cards or sequence diagrams to figure out what happens in the software .
  - **Package diagrams** to show the large-scale organization of the software .
  - **State diagrams** for classes with complex life histories .
  - **Deployment diagrams** to show the physical layout of the software .

Fitting the UML into a Process – Documentation

- Once you have built the Software, you can use the UML to help document what you have done .
- Understanding Legacy Code
  - With modern tools, you can generate detailed diagrams for key parts of a system .

References

- Fowler, M., *UML Distilled,* 3rd Ed. Addison-Wesley, 2004.