UML: Unified Modeling Language

Objectives of UML

- UML is a general purpose notation that is used to
  - visualize
  - specify
  - construct and
  - document

  the artifacts of a software system

Structural Diagrams

- **Class Diagram** – set of classes and their relationships. Describes interface to the class (set of operations describing services)
- **Object Diagram** – set of objects (class instances) and their relationships
- **Component Diagram** – logical groupings of elements and their relationships
- **Deployment Diagram** - set of computational resources (nodes) that host each component

Behavioral Diagram

- **Use Case Diagram** – high-level behaviors of the system, user goals, external entities: actors
- **Sequence Diagram** – focus on time ordering of messages
- **Collaboration Diagram** – focus on structural organization of objects and messages
- **State Chart Diagram** – event driven state changes of system
- **Activity Diagram** – flow of control between activities

From Requirements to Analysis

- From the Use Case diagrams an initial set of objects and classes can be identified
- This is the first step of analysis
- The second step is to refine the use cases through interaction diagrams

What is a Good Class?

What is a Good Class?

- Should provide a crisp abstraction of something from the problem domain (or solution) domain
- Embody a small well defined set of responsibilities and carry them out well
- Provides clear separation of abstraction, specification, and implementation
- Is understandable and simple yet extendable and adaptable

Object Oriented Decomposition

- Identifying objects which derived from the vocabulary of the problem (and solution) domain
- Algorithmic view highlights the ordering of events
- OO view emphasizes the agents that either cause action or are the subject upon which the actions operate

Object Model

- **Abstraction** – separate behavior from implementation
- **Encapsulation** – separate interface from implementation
- **Modularity** – high cohesion and low coupling
- **Hierarchy** – Inheritance
- **Polymorphism** – dynamic variable binding
- **Typing** – strong enforcement
- **Concurrency** – active vs. inactive
- **Persistence** – existence transcends runtime

Types of Objects

- **Boundary** – represent the interactions between the system and actors
- **Control** – represent the tasks that are performed by the user and supported by the system
- **Entity** – represent the persistent information tracked by the system

Object Modeling

- Given the high-level requirements (use cases)
- Define the object model
  - Identify objects
  - Compile a data dictionary
  - Identify association and aggregations
  - Identify attributes of objects
  - Generalize objects into classes
  - Organize and abstract using inheritance
  - Iterate and refine model
  - Group classes into modules/components

Object identification

- Identifying objects (or object classes) is the most difficult part of object oriented design
- There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers
- Object identification is an iterative process. You are unlikely to get it right first time

Approaches to identification

- Use a grammatical approach based on a natural language description of the system:
  - Objects and attributes are nouns and verbs are operations
- Base the identification on tangible things in the application domain
  - Objects, roles, events, interactions, locations
- Use a behavioural approach and identify objects based on what participates in what behaviour
  - who initiates and participates in those behaviours
- Use a scenario-based analysis. The objects, attributes and methods in each scenario are

Example: Weather Monitoring Station

- This system shall provide automatic monitoring of various weather conditions. Specifically, it must measure:
  - wind speed and direction
  - temperature
  - barometric pressure
  - humidity
- The system shall also provide the following derived measurements:
  - wind chill
  - dew point temperature
  - temperature trend

Weather Monitoring System Requirements

- The system shall have the means of determining the current time and date so that it can report the highest and lowest values for any of the four primary measurements during the previous 24 hour period
- The system shall have a display that continuously indicates all eight primary and derived measurements, as well as current time and date
- Through the use of a keypad the user may direct the system to display the 24 hour low or high of any one primary measurement, with the time of the reported value
- The system shall allow the user to calibrate its sensors against known values, and set the current time and date

Hardware Requirements

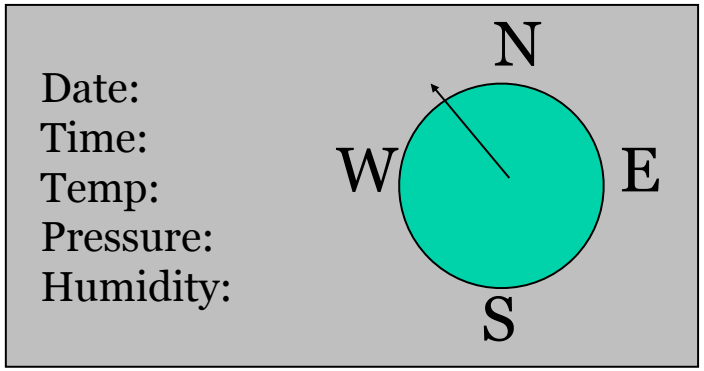
- Use a single board computer
- Time and date are supplied by an on-board clock accessible via memory mapped I/O
- Temperature, barometric pressure, and humidity are measured by on board circuits with remote sensors.
- Wind direction and speed are measured from a boom encompassing a wind vane (16 directions) and cups (which advance a counter every revolution)

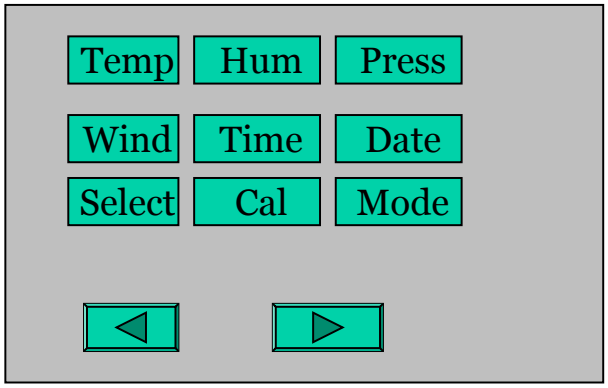
Hardware Requirements (contd.)

- User input is provided through an off the shelf keypad, managed by onboard circuit supplying audible feed back for each key press.
- Display is off the self LCD with a simple set of graphics primitives.
- An onboard timer interrupts the computer every 1/60 second.

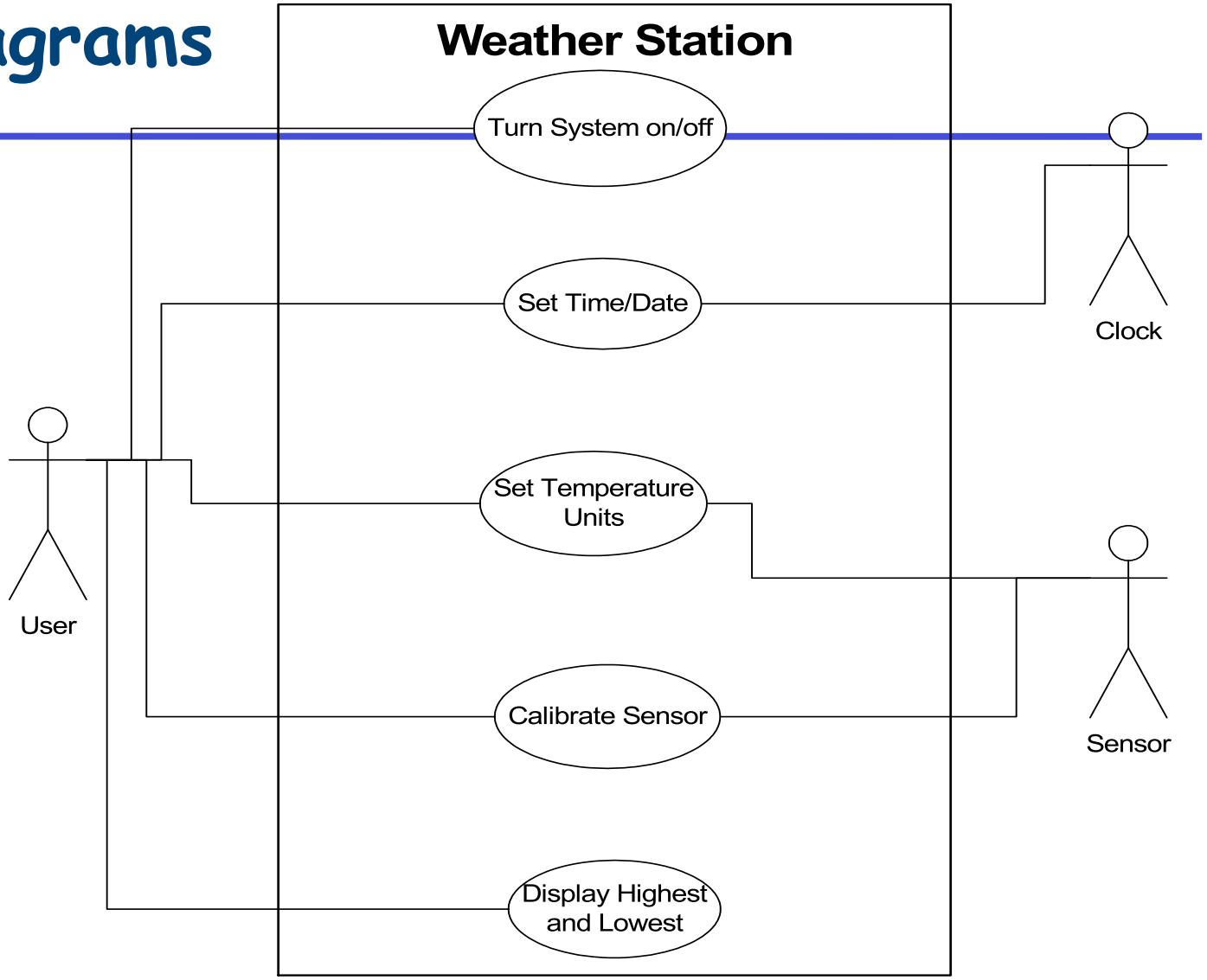
Display and Keypad

- LCDDisplay – Values and current system state (Running, Calibrating, Selecting, Mode)
  - Operations: drawtext, drawline, drawcircle, settextsize, settextstyle, setpensize
- Keypad allows user input and interaction
  - Operations: last key pressed





Use Diagrams



Scenario: Powering Up

1. Power is turned on
2. Each sensor is constructed
3. User input buffer is initialized
4. Static elements of display are drawn
5. Sampling of sensors is initialized

- The past high/low values of each primary measurement is set to the value and time of their first sample.
- The temperature and Pressure trends are flat.

Scenario: Setting Time and Date

1. User presses Select key
2. System displays selecting
3. User presses any one of the keys Time or Date. Any other key is ignored except Run
4. System flashes the corresponding label
5. Users presses Up or Down to change date or time.
6. Control passes back to step 3 or 5

Scenario: Display Highest and Lowest

1. User presses Select key
2. System displays selecting
3. User presses any one of the keys (Wind, Temp, Humidity, Pressure). Any other key is ignored except Run
4. System flashes the corresponding label
5. Users presses Up or Down to select display of highest or lowest in 24 hour period. Any other key press is ignored except for Run
6. System displays value with time of occurrence
7. Control passes back to step 3 or 5

Identify Objects

- From the vocabulary of the domain

Identify Objects

- From the vocabulary of the domain
- User, clock, sensor, temperature, LCDDisplay, Keypad, time, date, wind speed, humidity, barometer, calibrator, metric units, English units, input manager, sensor sampler, wind direction, display manager, trend, pressure, current time, current date, current temp, high temp, low temp, change temp, change time, power up, power down, input buffer, trend, key, running, selecting

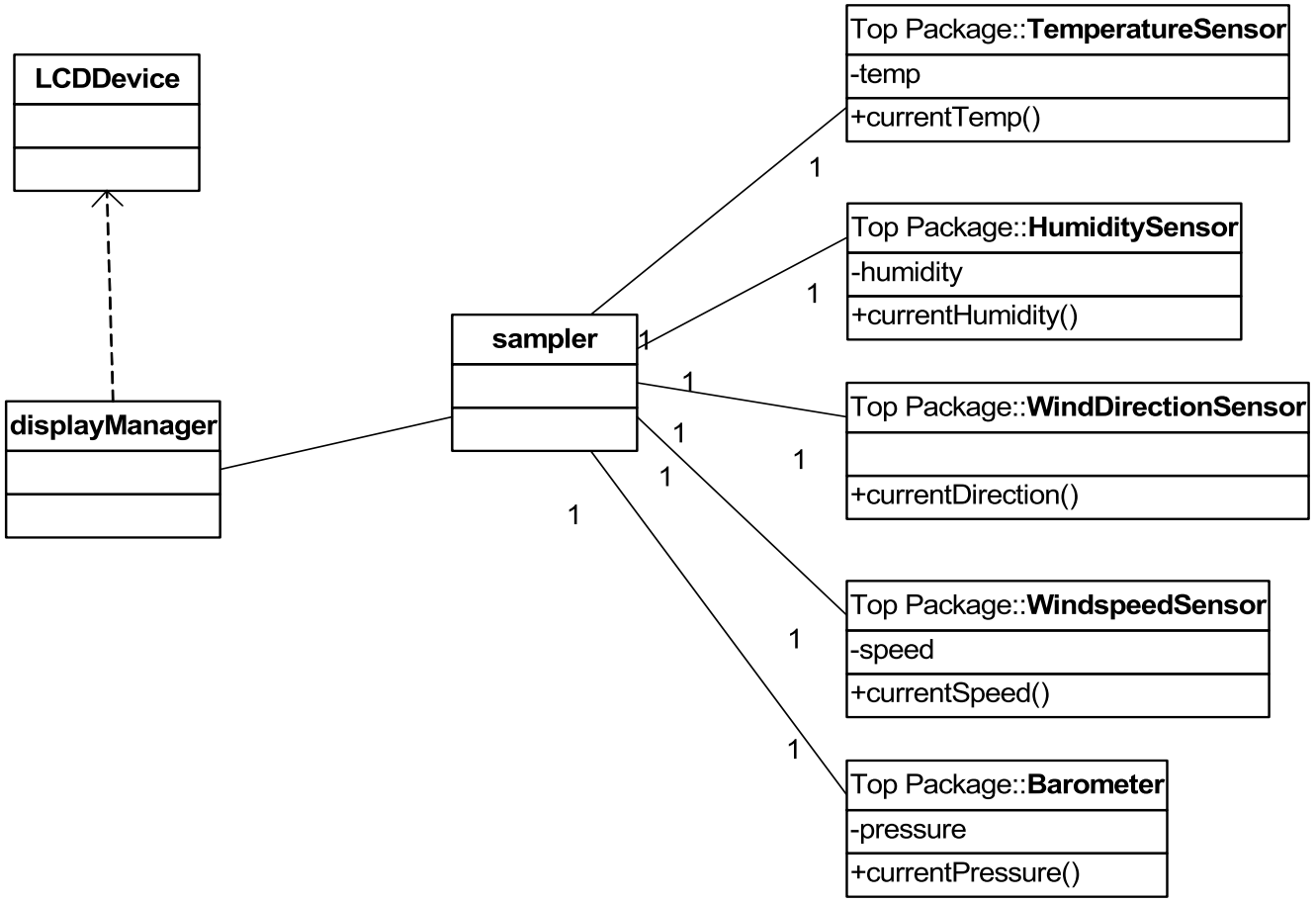
Eliminate Terms

- Refine the model by eliminating
- Redundancy – classes that represent the same concept
- Irrelevant classes – things you don't care about
- Vague classes – ill defined boundaries
- Attributes – describe parts of objects
- Operators – sequence of actions are often mistaken for classes
- Roles – what it is not the role it plays
- Implementation details – save it for later

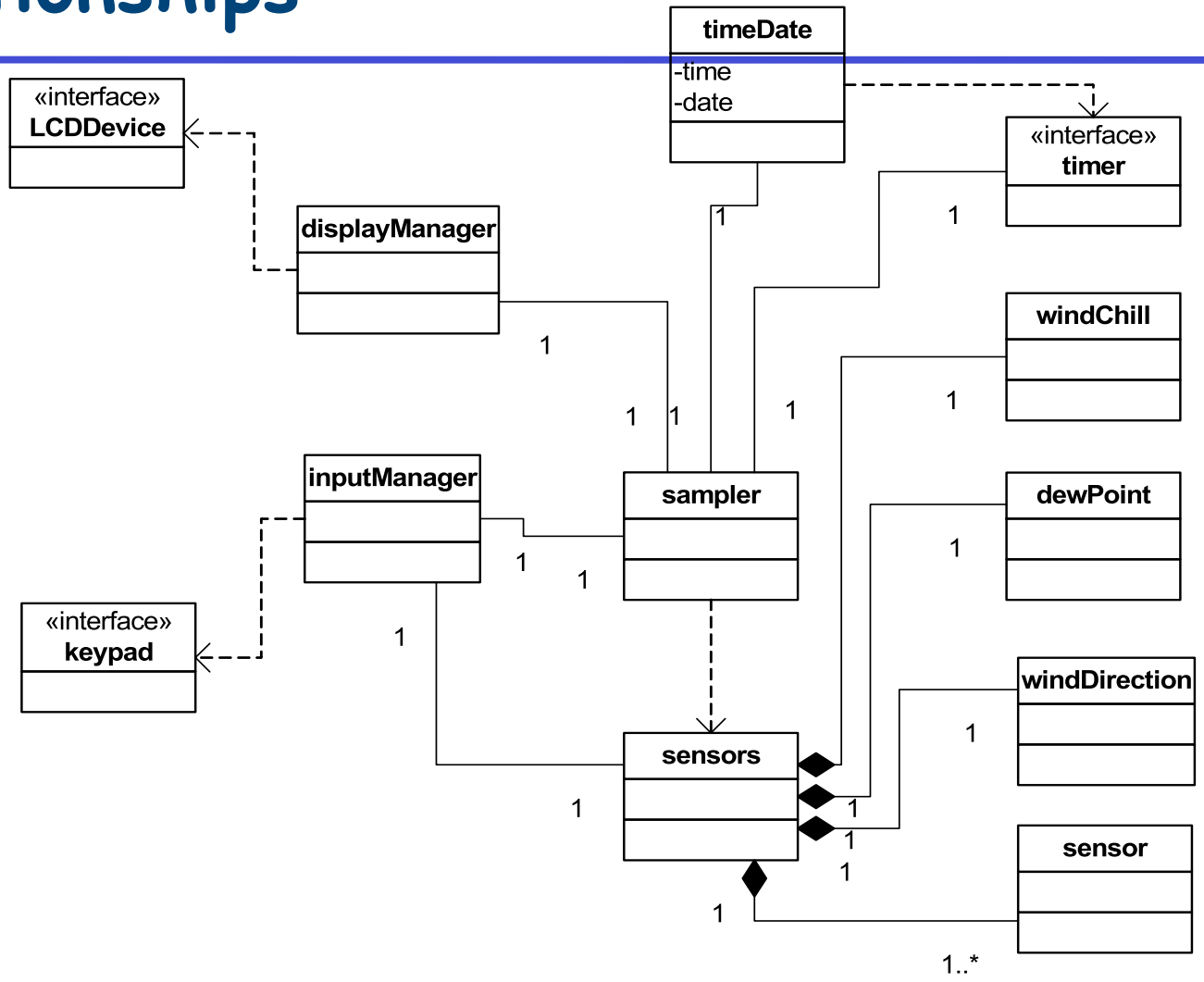
New Data Dictionary

- Time & Date
- Sensors: Temperature, Pressure, Humidity, Wind Speed, Wind Direction
- Keypad
- Input Manager
- Display (LCD Device)
- Display Manager
- Timer (clock)
- Sensor Sampler

Relationships



Relationships



More UML Diagrams

- Modeling Behavior
- Interaction Diagrams
- State Chart Diagrams
- Activity Diagrams

Refining the Object Model

- Typically, only very simplistic object models can be directly derived from use cases
- A better understanding of the behavior of each use case is necessary (i.e., analysis)
- Use interaction diagrams to specify and detail the behavior of use cases
- This helps to identify and refine key abstractions and relationships
- Operations, attributes, and messages are also identified during this process

Interaction Diagrams

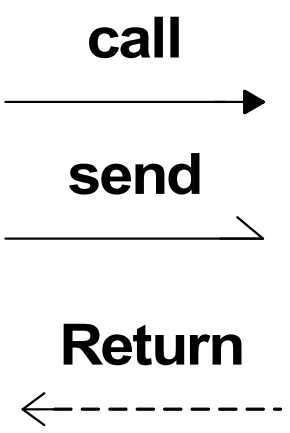
- There is one (or more) Interaction diagram per use case
  - Represent a sequence of interactions
  - Made up of objects, links, and messages
- Sequence diagrams
  - Models flow of control by time ordering
  - Emphasizes passing messages over time
  - Shows simple iteration and branching
- Collaboration diagrams
  - Models flow of control by organization
  - Structural relationships among instances in the interaction

Sequence Diagrams

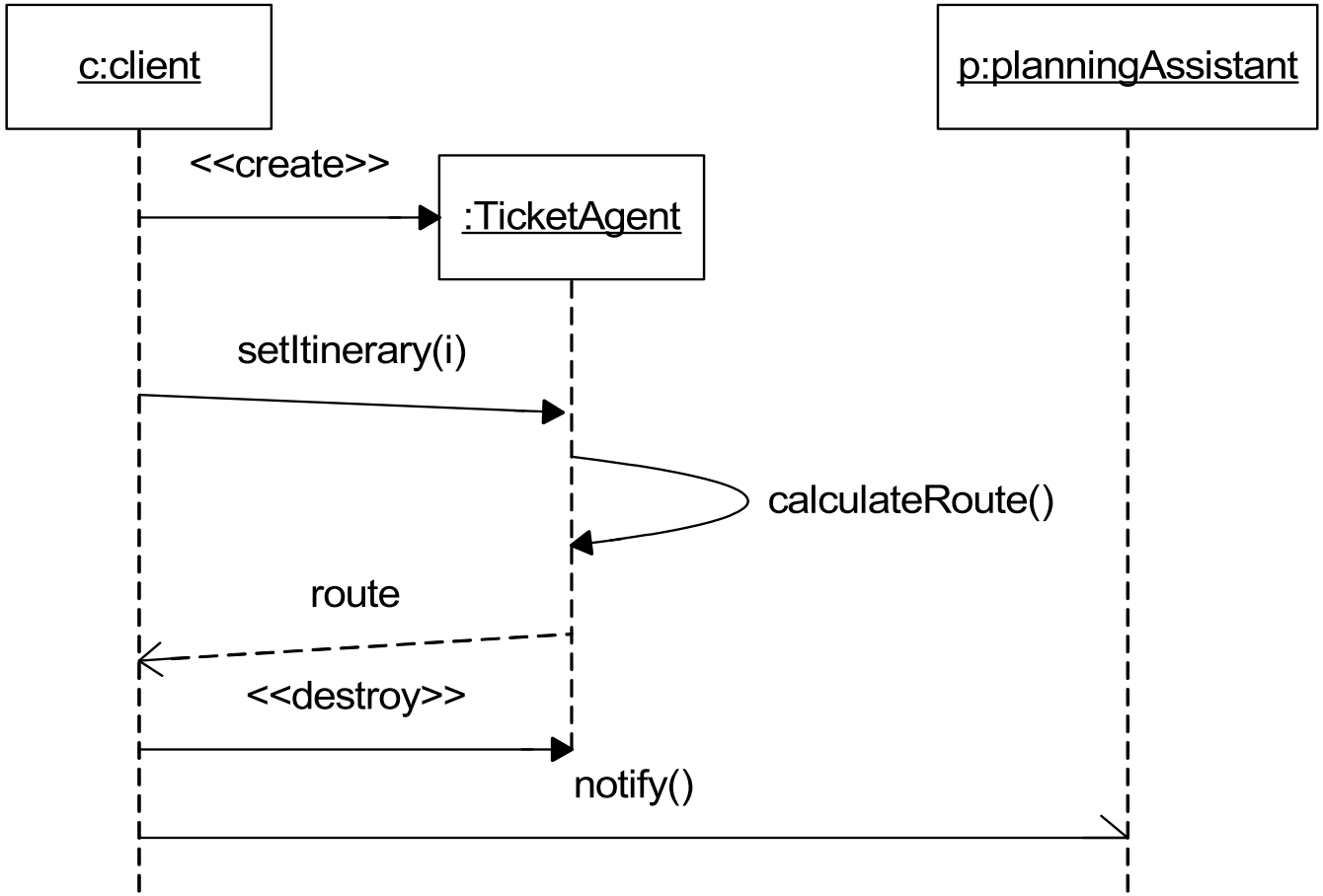
- X-axis is objects
  - Object that initiates interaction is left most
  - Object to the right are increasingly more subordinate
- Y-axis is time
  - Messages sent and received are ordered by time
- Object life lines represent the existence over a period of time
- Activation (double line) is the execution of the procedure.

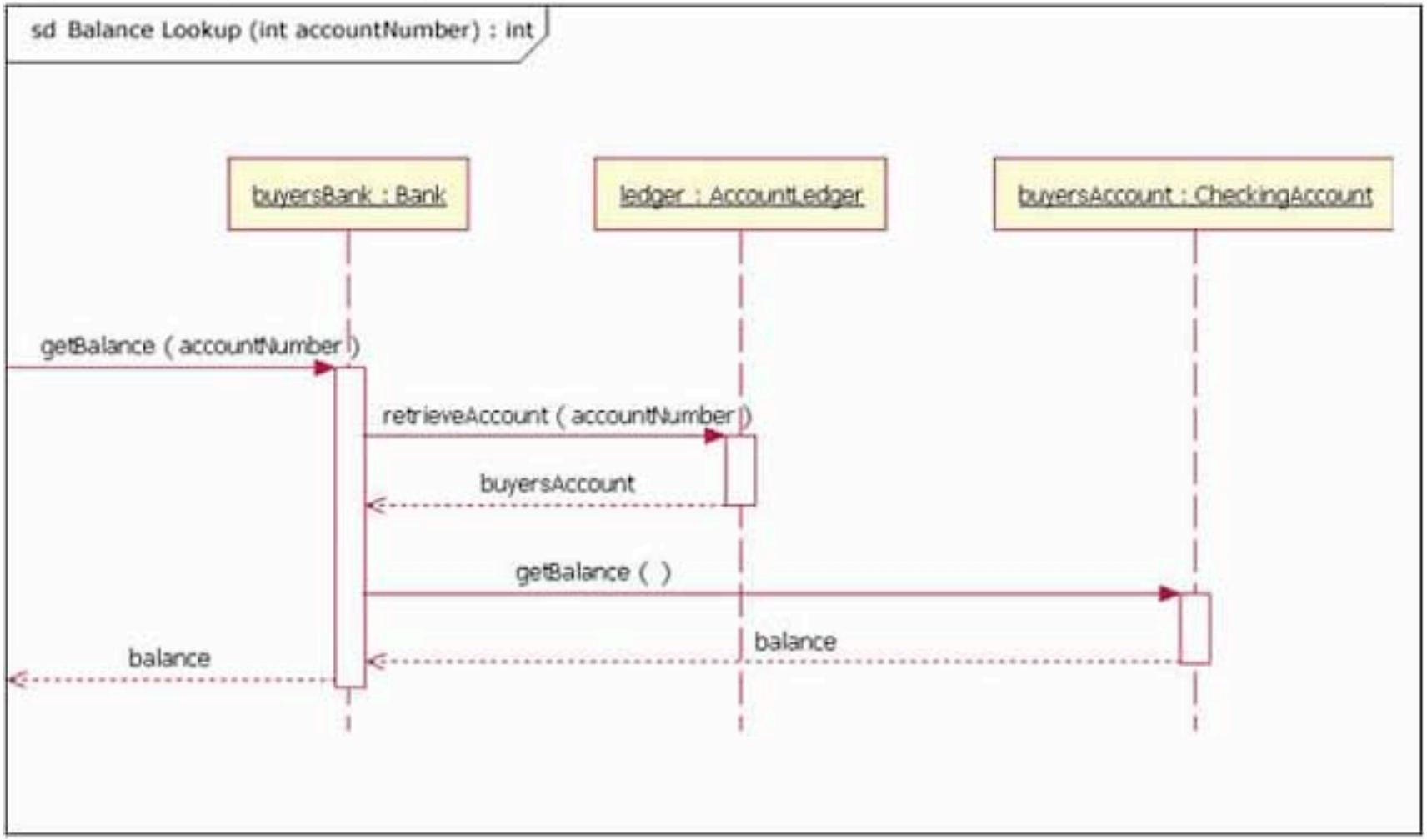
Message Passing

- Send – sends a signal (message) to an object
- Return – returns a value to a caller
- Call – invokes an operation
- Stereotypes
  - <<create>>
  - <<destroy>>



Example UML Sequence Diagram




sd Balance Lookup (int accountNumber) : int
buyersBank : Bank
ledger : AccountLedger
buyersAccount : CheckingAccount
getBalance (accountNumber)
retrieveAccount (accountNumber)
buyersAccount
getBalance ( )
balance
balance

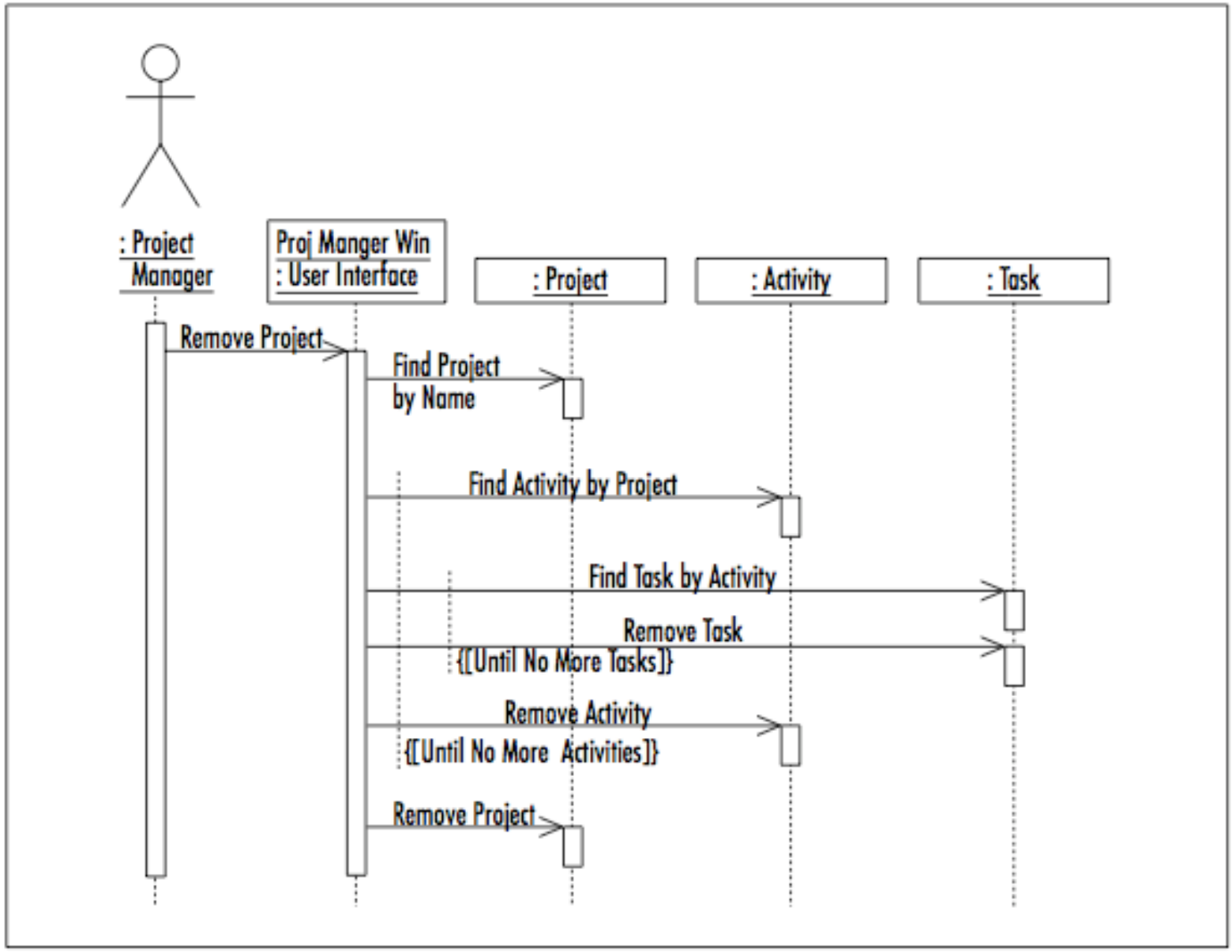


Figure 4-18: Remove Project Sequence Diagram (Version 2)

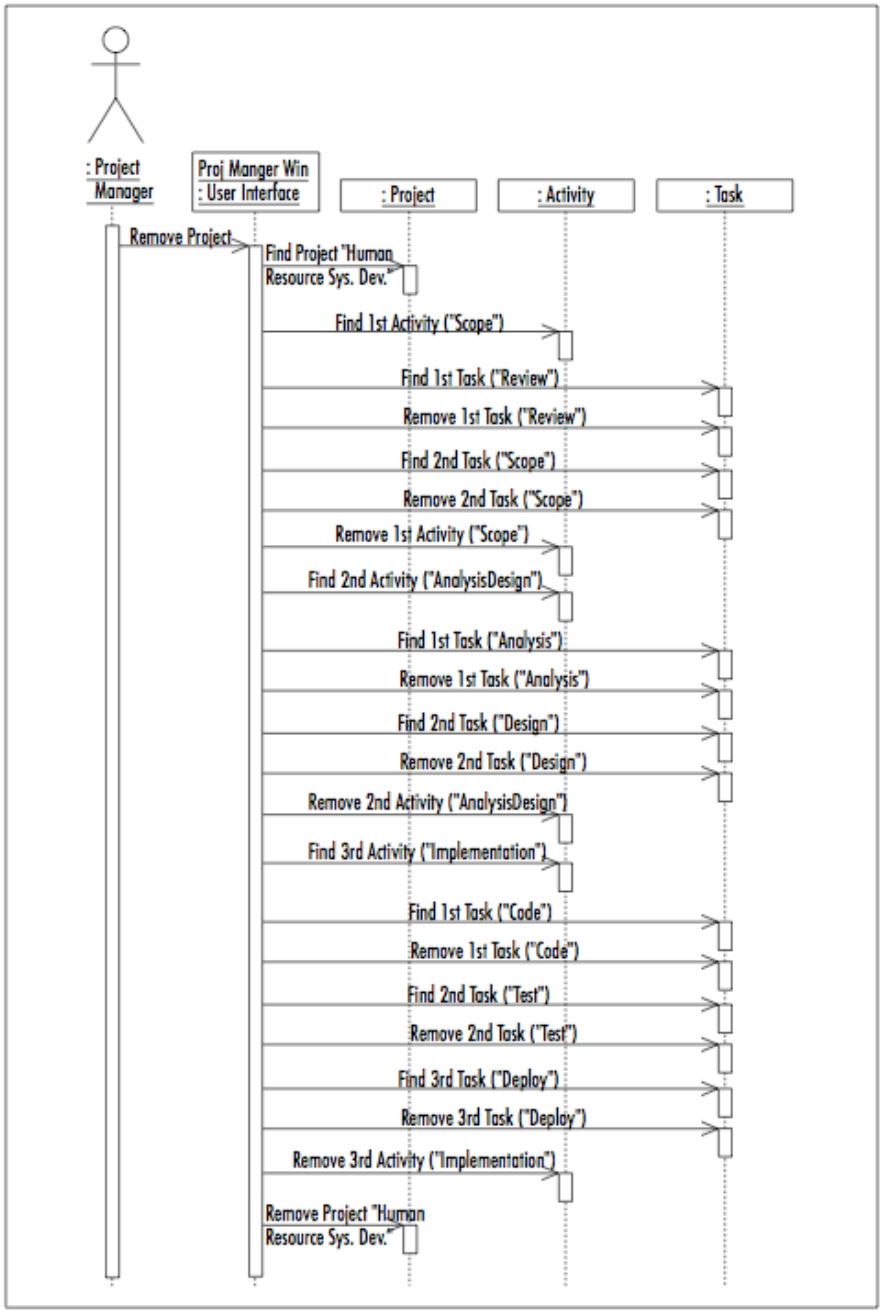
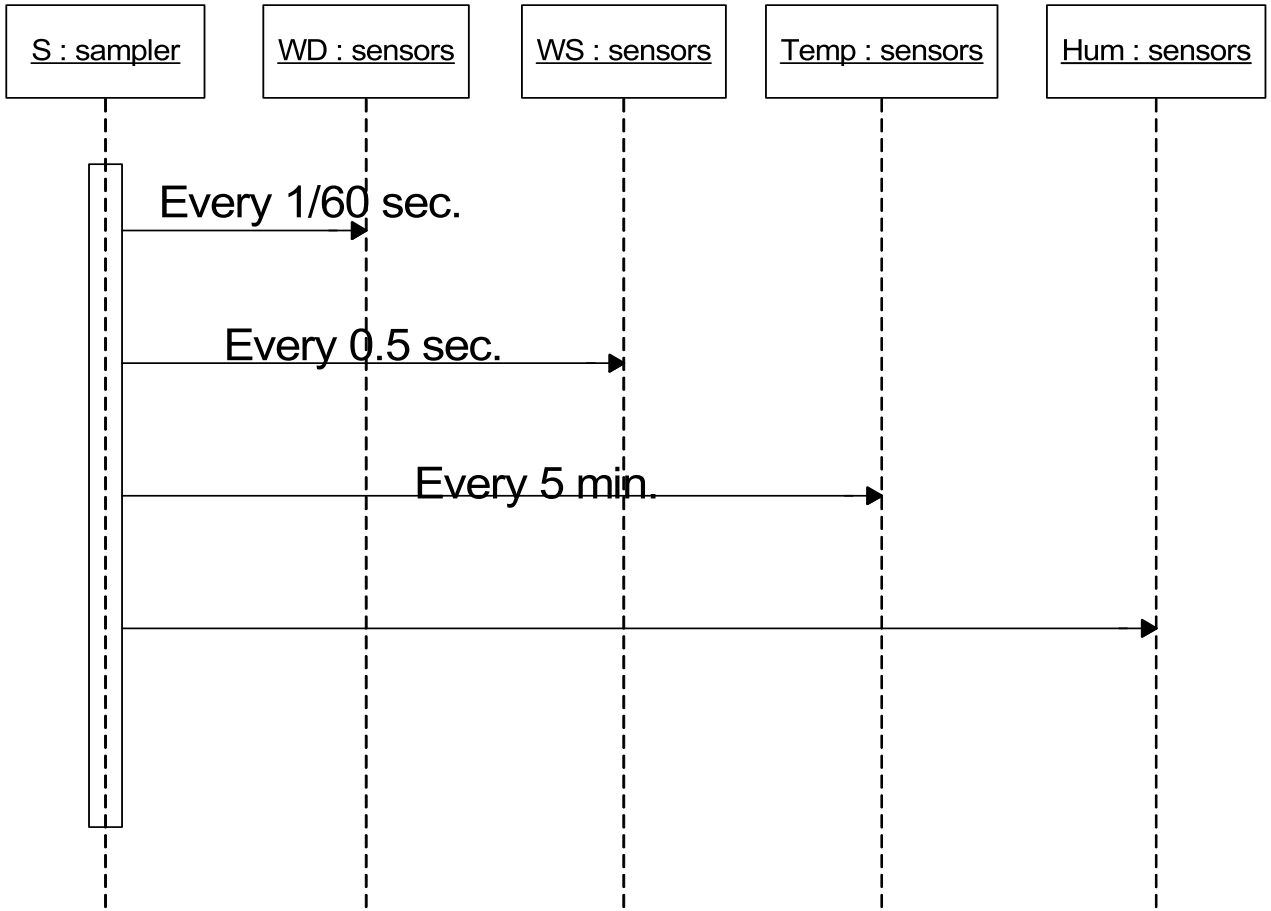


Figure 4-19: Remove Project Scenario (Sequence Diagram)

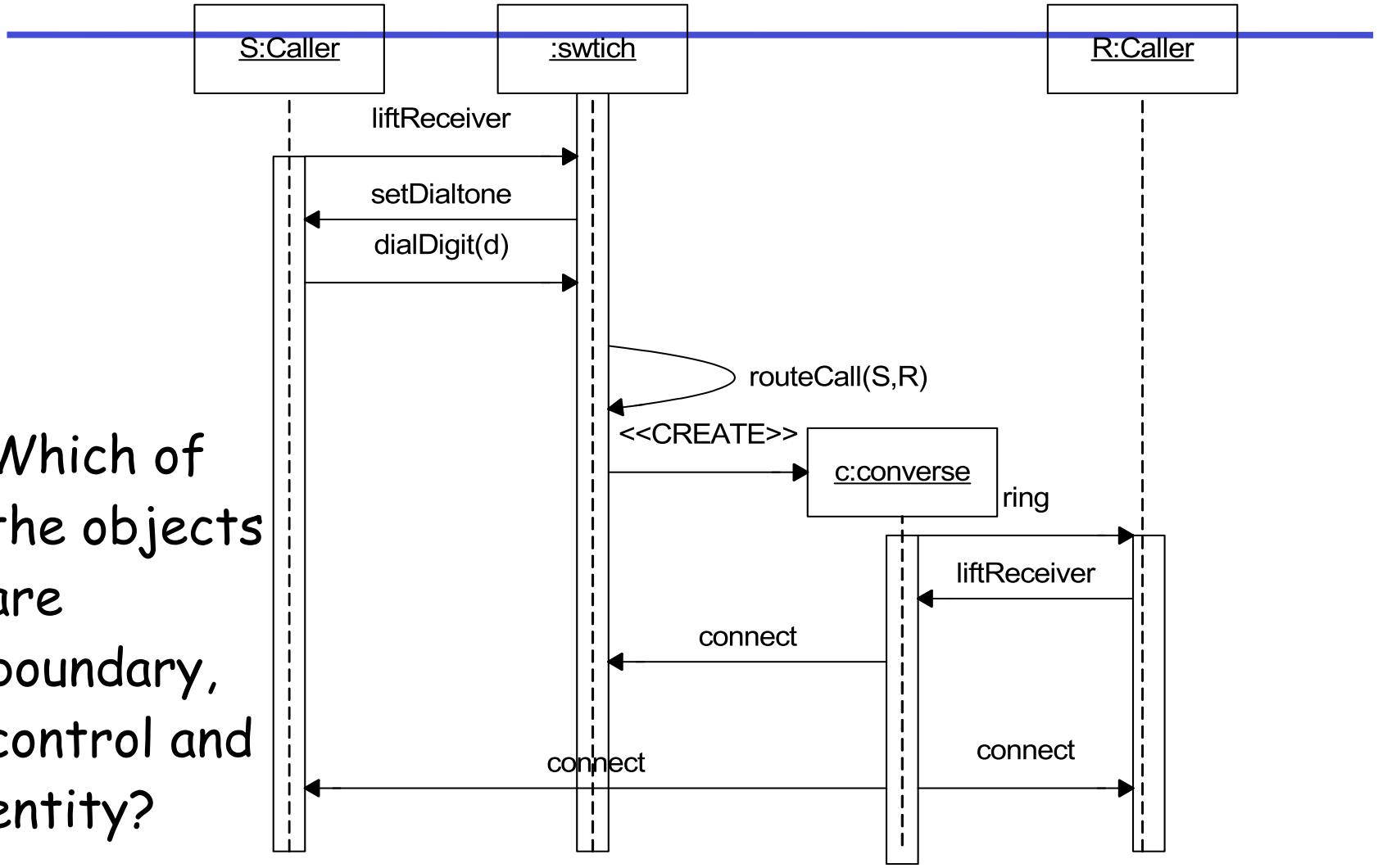
Example: Weather Station



Types of Diagrams

- **Structural Diagrams** – focus on static aspects of the software system
  - Class, Object, Component, Deployment
- **Behavioral Diagrams** – focus on dynamic aspects of the software system
  - Use-case, Interaction, State Chart, Activity

Example: Phone Call



Which of the objects are boundary, control and entity?

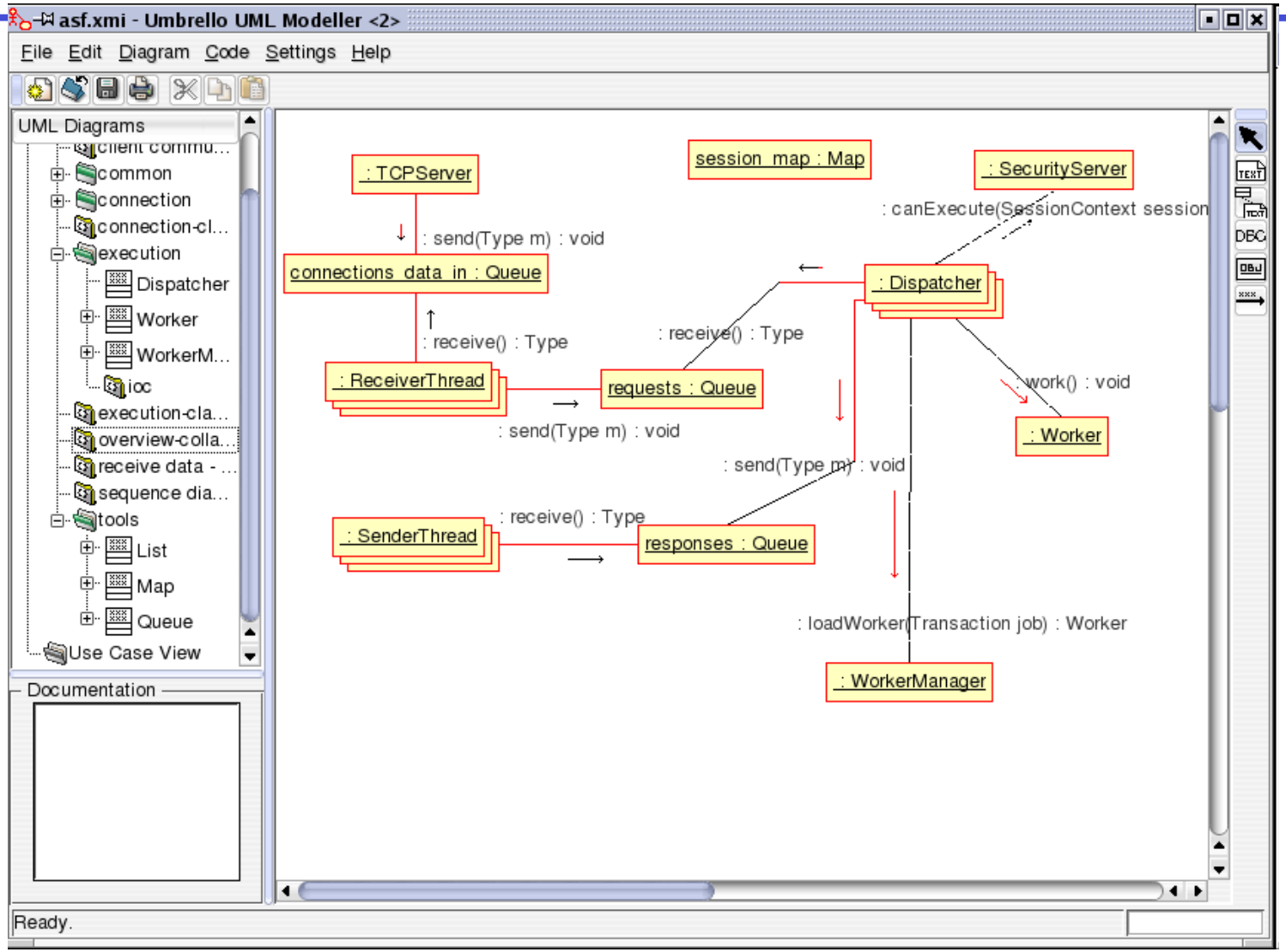
Properties of Sequence Diagrams

- Initiator is leftmost object (**boundary** object)
- Next is typically a **control** object
- Then comes **entity** objects

Collaboration Diagrams

- Emphasizes the organization of the objects that participate in an interaction
- Association
- Messages, flow, and sequencing

Example: Collaboration Diagram



Collaboration vs. Sequence

- The two diagrams show almost the same information
- Collaboration diagrams show more static structure (however, class diagrams are better at this)
- Sequence diagrams clearly highlight the orderings and very useful for multi-tasking

Summary (Interaction Diagrams)

- Well structured interaction diagrams:
  - Is focused on communicating one aspect of a system's dynamics
  - Contains only those elements that are essential to understanding
- Diagrams should have meaningful names
- Layout diagram to minimize line crossings
- Use branching sparingly (leave for activity diagrams)

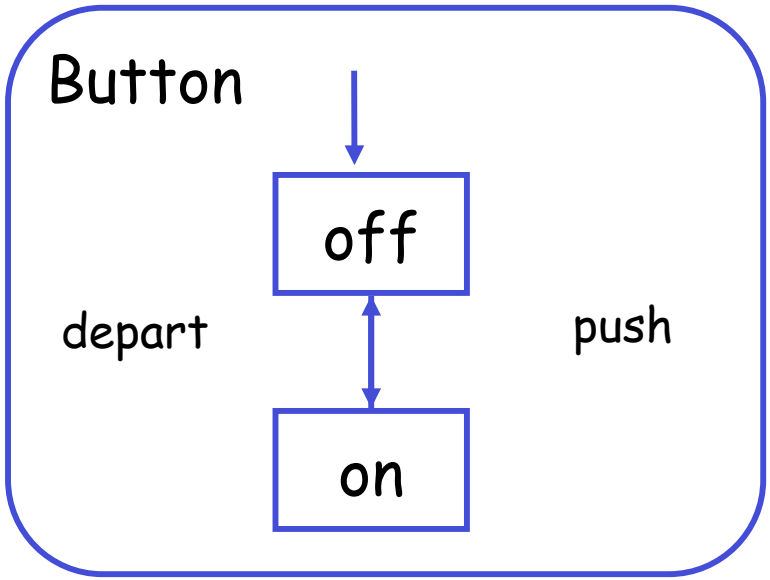
State Diagrams

- Finite state machines (i.e., automata, Mealy/ Moore, state transition)
- Used to describe the behavior of one object (or sometimes an operator) for a number of scenarios that affect the object
- They are not good for showing interaction between objects (use interaction diagrams)
- Only use when the behavior of an object is complex and more detail is needed

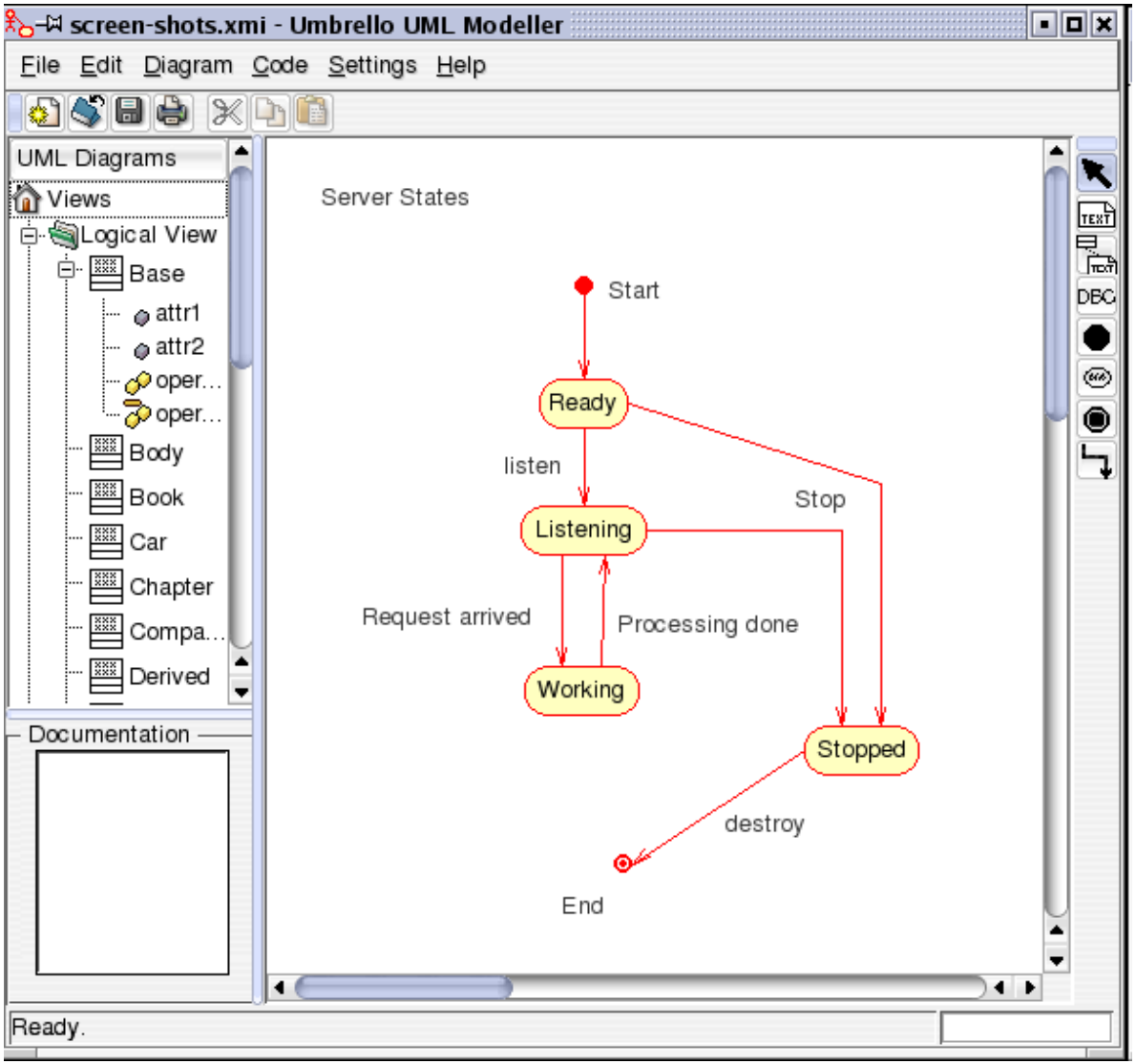
State Diagram Features

- Event – something that happens at a specific point (e.g., alarm goes off)
- Condition – something that has a duration
  - Alarm is on
  - Fuel level is low
- State – an abstraction of the attributes and relationships of an object (or system)
  - The fuel tank is in a too low level when the fuel level is below level x for n seconds

Example: on/off Switch



Example


screen-shots.xmi - Umbrello UML Modeller
File Edit Diagram Code Settings Help
UML Diagrams
Views
Logical View
Base
attr1
attr2
oper...
oper...
Body
Book
Car
Chapter
Compa...
Derived
Documentation
Server States
Start
Ready
listen
Stop
Listening
Request arrived
Processing done
Working
Stopped
destroy
End
Ready.

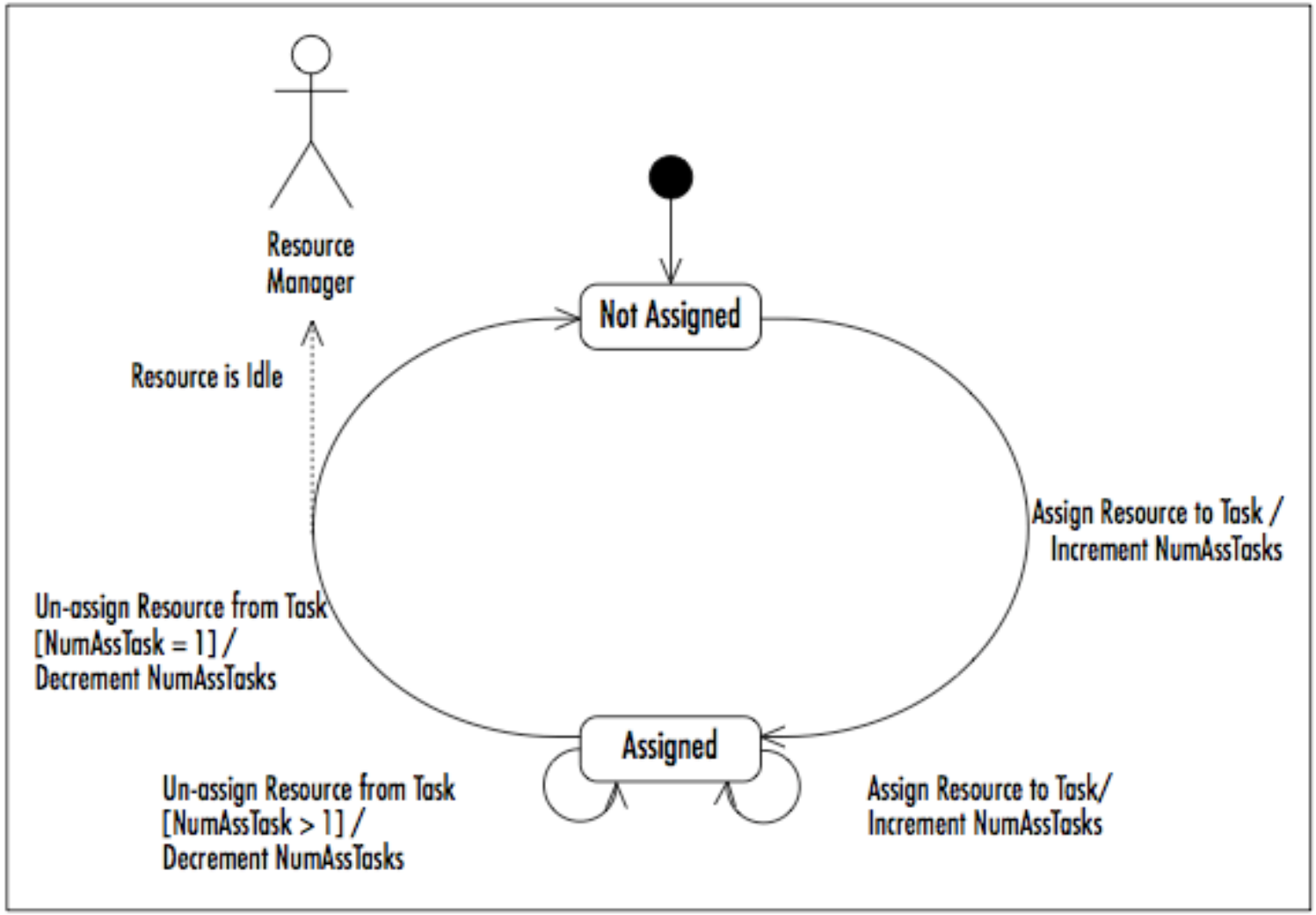
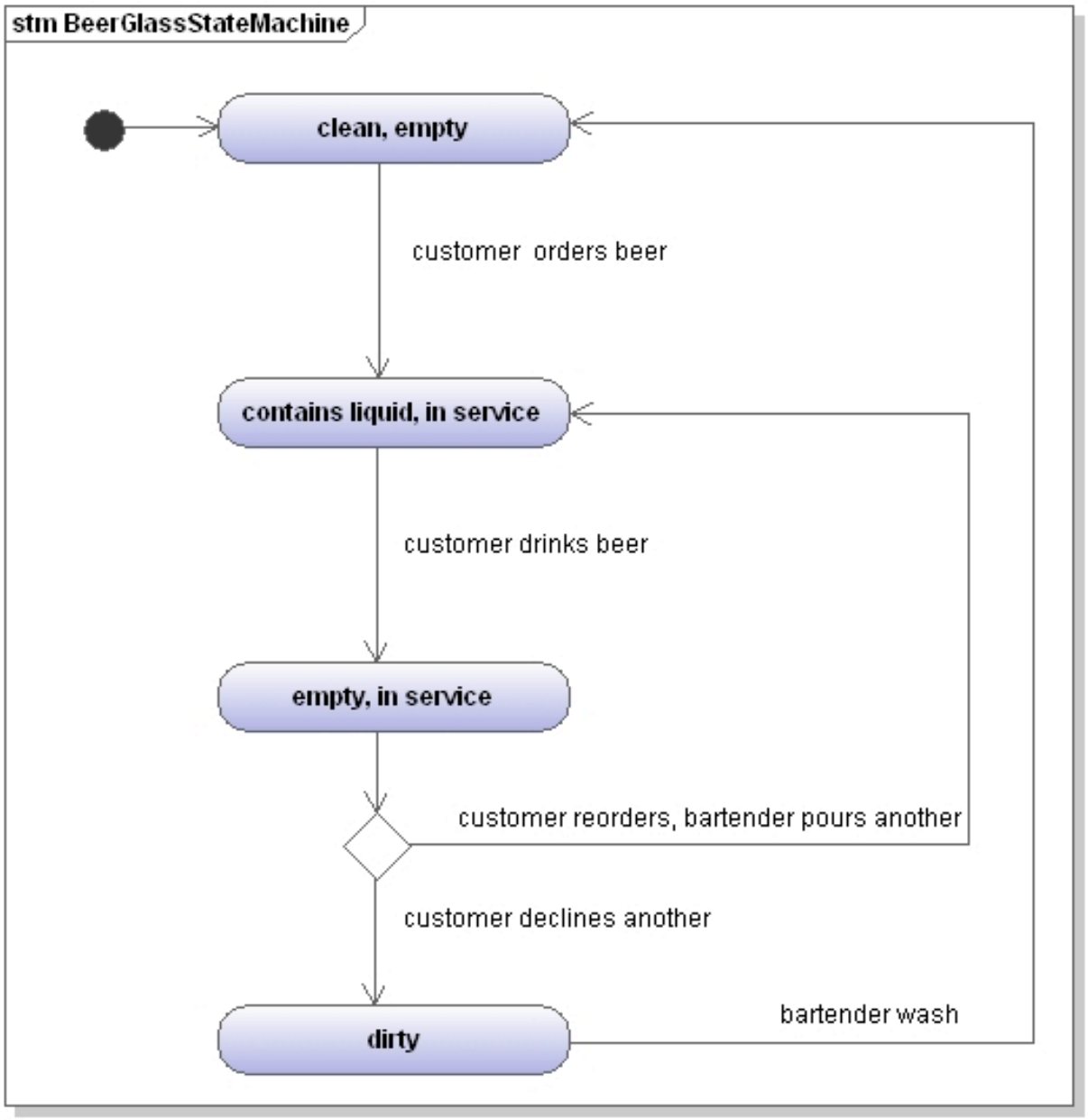


Figure 4-30: Resource Statechart Diagram


stm BeerGlassStateMachine
clean, empty
customer orders beer
contains liquid, in service
customer drinks beer
empty, in service
customer reorders, bartender pours another
customer declines another
dirty
bartender wash
Generated by UModel
www.altova.com

Activity Diagrams

- Special form of a state machine (flow chart) – intended to model computations and workflows
- States of the executing the computation not the states of an object
- Flow between activity states is caused by the end of a computation rather then an event
- Emphasis on control flow

Why Activity Diagrams

- Flowcharts (abet a bit glorified) are not very amiable to OO
- Not part of any previous notations
- Suitable for modeling the business activities
- OO and UML is becoming very prevalent in business applications

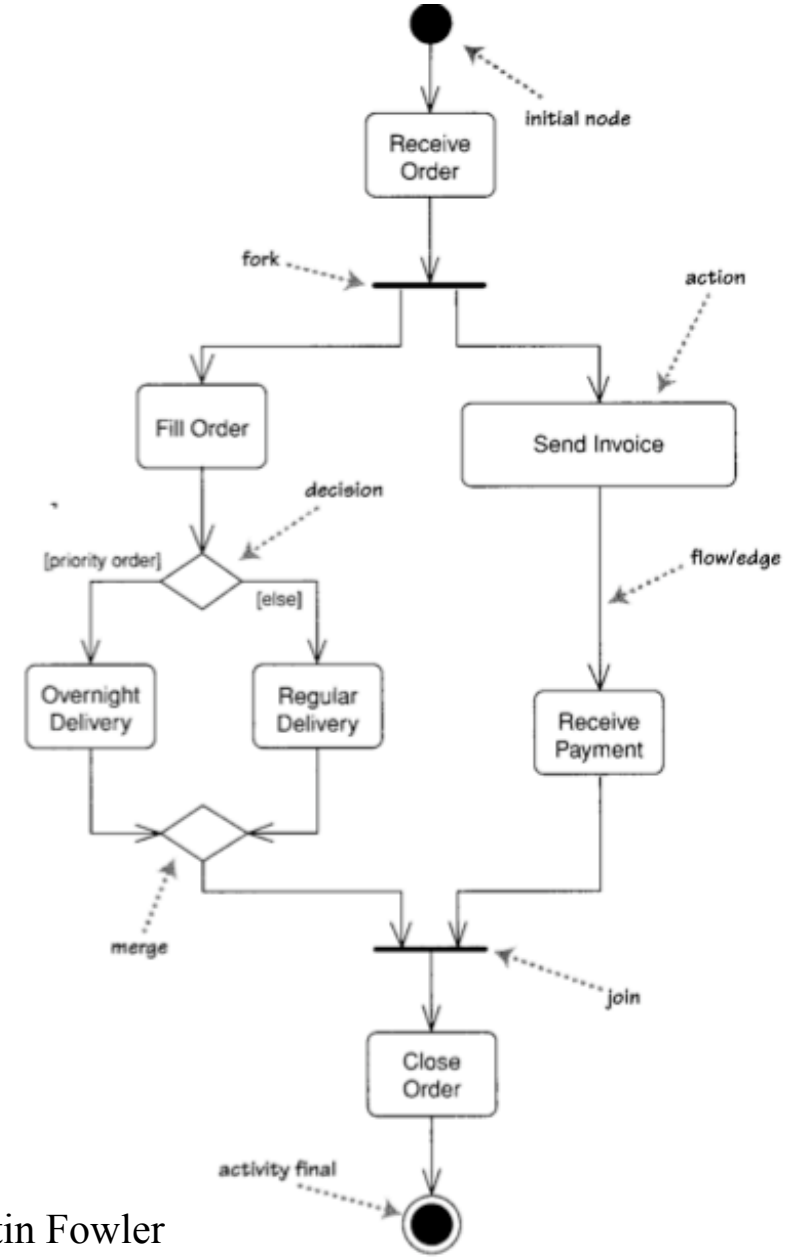


Figure 11.1 *A simple activity diagram*

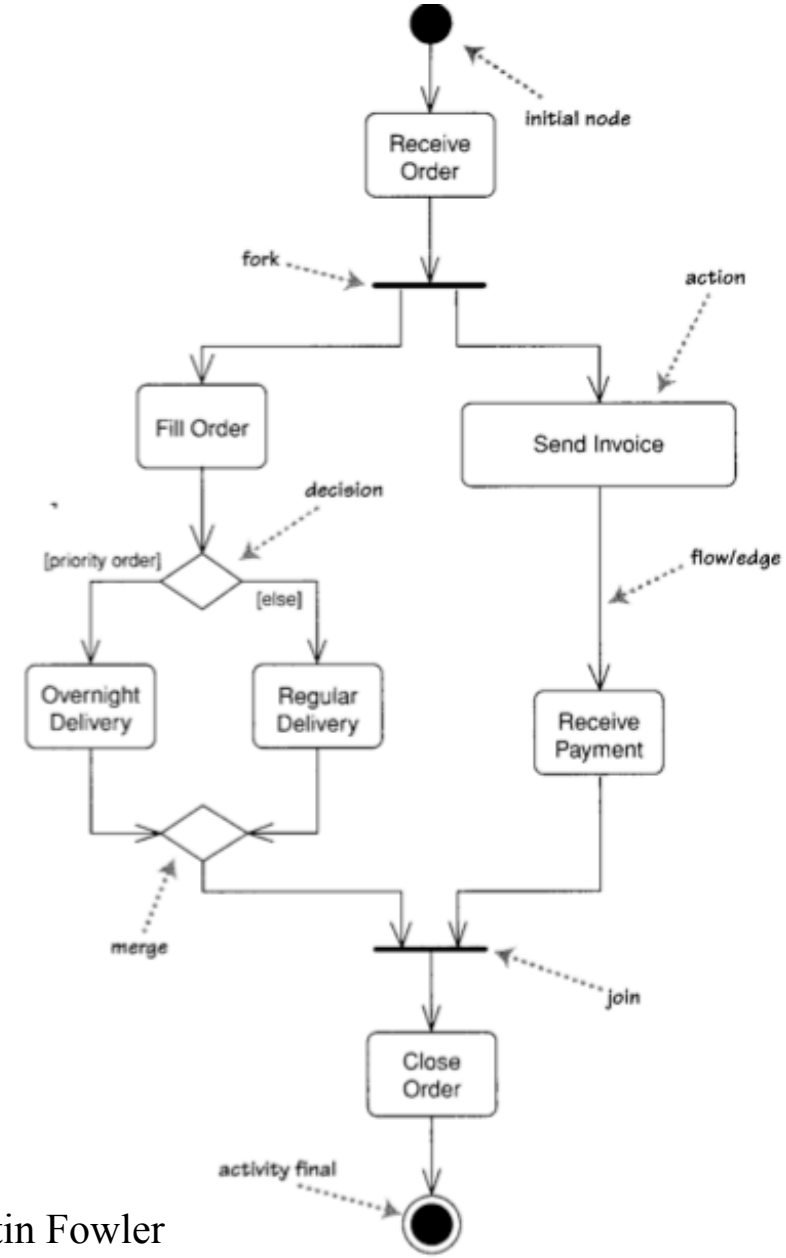


Figure 11.1 *A simple activity diagram*

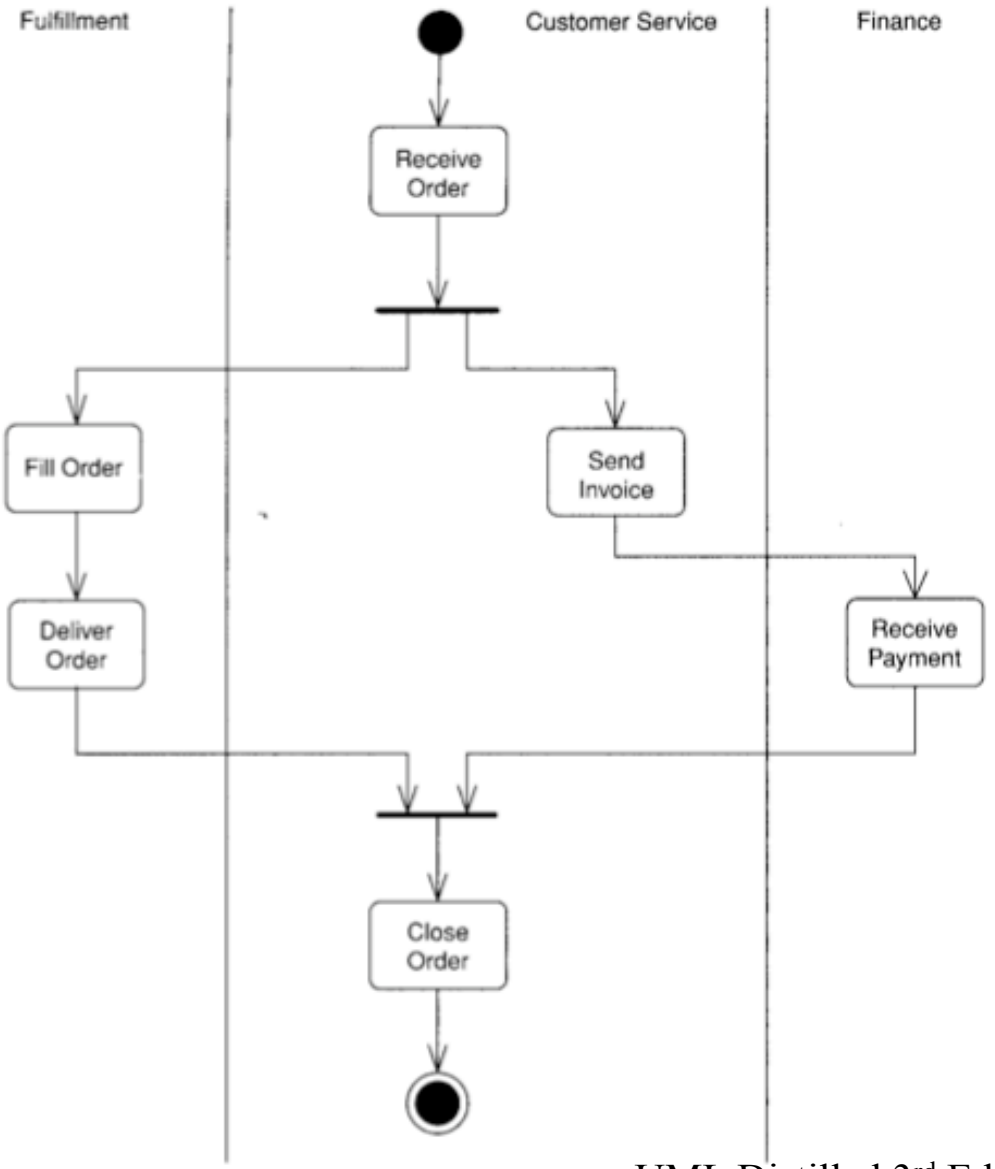


Figure 11.4 *Partitions on an activity diagram*

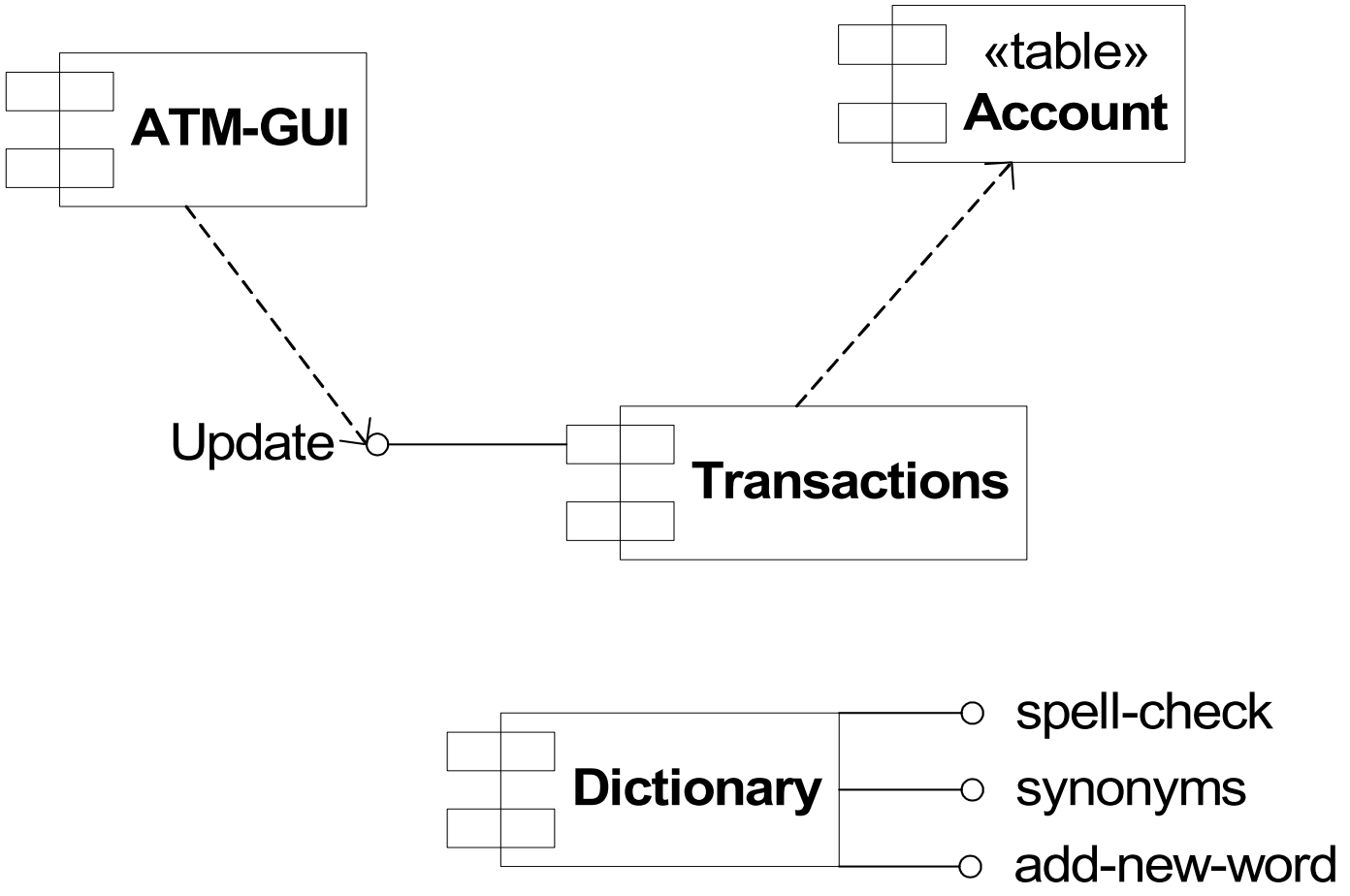
Even More UML

- Implementation and Architectural Diagrams
- Component diagrams
- Deployment diagrams

Component Diagrams

- A component is a physical thing that conforms to and realizes a set of interfaces
- Bridge between logical and physical models
- Can represent object libraries, COM components, Java Beans, etc.
- Classes represent logical abstractions, components represent physical things that reside on a node (machine)
- Components are reachable only through interface

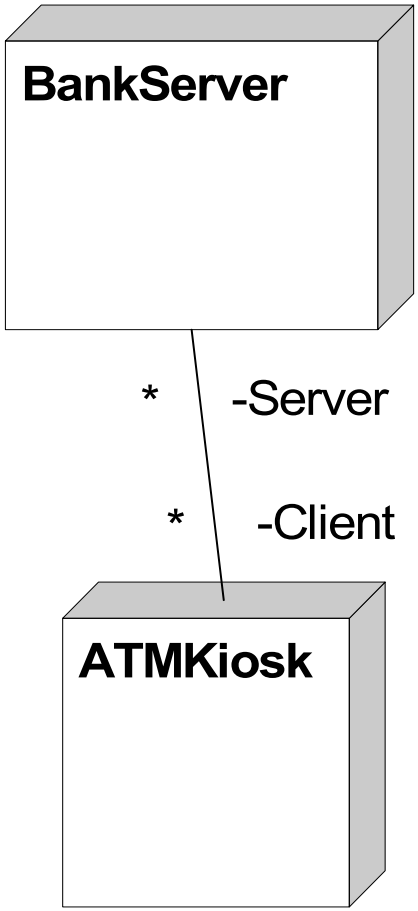
Examples


ATM-GUI
«table»
Account
Update
Transactions
Dictionary
spell-check
synonyms
add-new-word

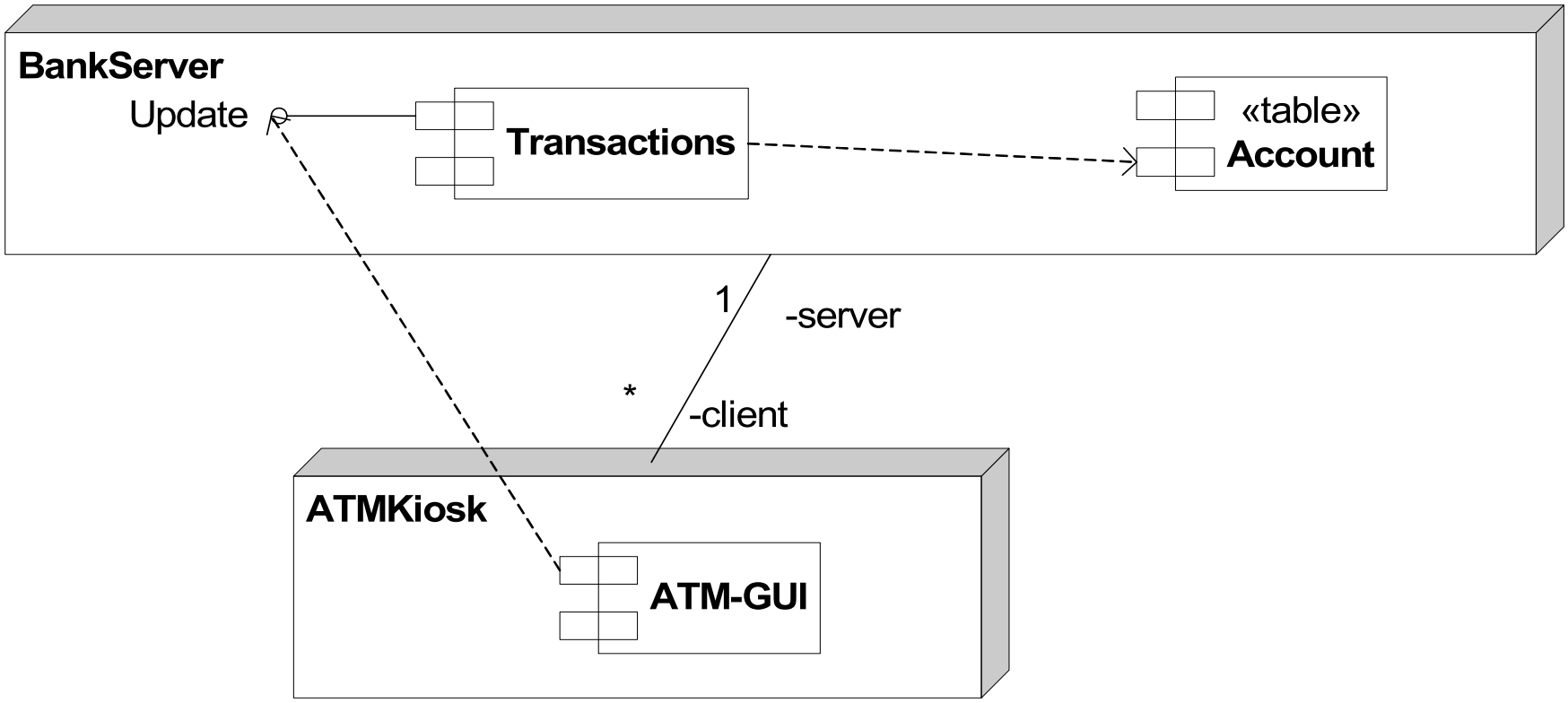
Deployment Diagrams

- Nodes are physical elements that represent a computational resource (machine)
- Association between nodes
- Components are allocated to nodes (one or more)
- Components represent the physical packaging of logical elements
- Nodes represent the physical deployment of components

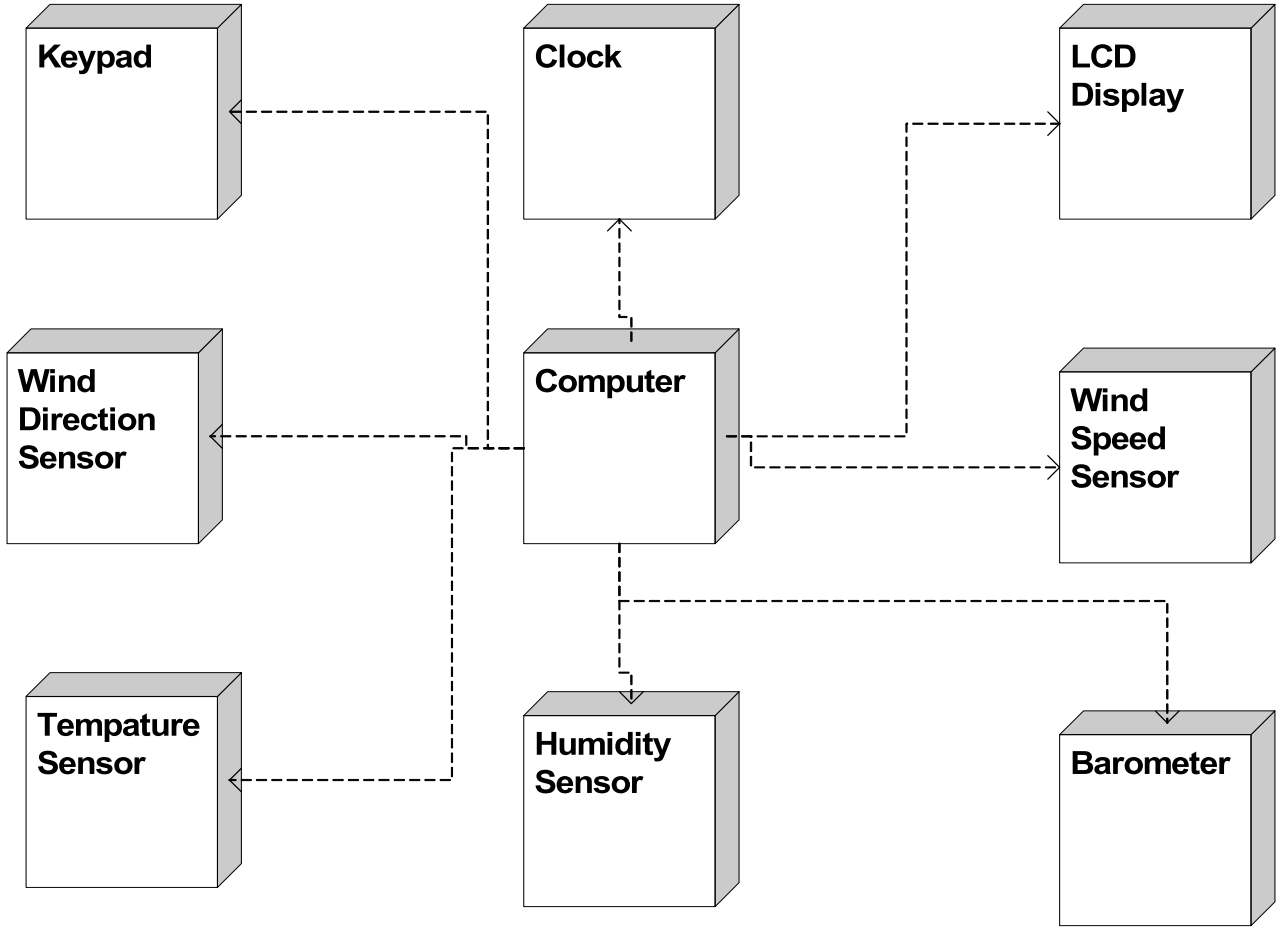
Example



With Components



Weather Station



ArgoUML

- http://argouml.tigris.org/

Opinions about UML: What's Good

- A common language
  - Makes it easier to share requirements, specs, designs
- Visual syntax is useful, to a point
  - A (good) picture is worth 1000 words
  - For the non-technical, easier to grasp simple diagrams than simple pseudo-code
- To the extent UML is precise, it forces clarity
  - Much better than natural language
- Commercial tool support

Opinions On UML: What's Bad

- Hodge-podge of ideas
  - Union of most popular modeling languages
  - Sublanguages remain largely unintegrated
- Visual syntax does not scale well
  - Many details are hard to depict visually
    - Ad hoc text attached to diagrams
  - No visualization advantage for large diagrams
    - 1000 pictures are very hard to understand

UML is Happening

- UML is being widely adopted
  - By users
  - By tool vendors
  - By programmers
- A step forward
  - Seems useful
  - First standard for high-levels of software process
  - Expect further evolution, development of UML

Exercise

- Given high-level specification
  - W&M BlackBoard System

- Work in groups of 2-3
- No talking or writing in any natural language
- ONLY using UML to communicate, see if you can come up with a more or less thorough design of the system so that all of you agree on the design of the system

Acknowledgements

- Some slides are courtesy of Rupak Majumdar