Chapter 7 System Design: Addressing Design Goals

Object-Oriented Software Engineering
Using UML, Patterns, and Java

Overview

System Design I

- ✓ Overview of System Design
- ✓ Design Goals
- ✓ Subsystem Decomposition
  - ✓ Architectural Styles

System Design II

- Hardware/Software Mapping
- Persistent Data Management
- Global Resource Handling and Access Control
- Software Control
- Boundary Conditions

Hardware Software Mapping

- This system design activity addresses two questions:
  - How shall we realize the subsystems: With hardware or with software?
  - How do we map the object model onto the chosen hardware and/or software?
    - Mapping the Objects:
      - Processor, Memory, Input/Output
    - Mapping the Associations:
      - Network connections

Mapping Objects onto Hardware

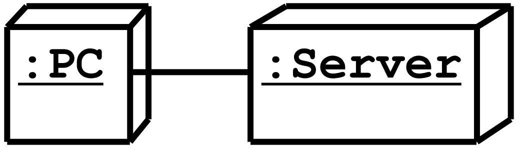
- Control Objects -> Processor
  - Is the computation rate too demanding for a single processor?
  - Can we get a speedup by distributing objects across several processors?
  - How many processors are required to maintain a steady state load?
- Entity Objects -> Memory
  - Is there enough memory to buffer bursts of requests?
- Boundary Objects -> Input/Output Devices
  - Do we need an extra piece of hardware to handle the data generation rates?
  - Can the desired response time be realized with the available communication bandwidth between subsystems?

Two New UML Diagram Types

- Deployment Diagram:
  - Illustrates the distribution of components at run-time.
  - Deployment diagrams use nodes and connections to depict the physical resources in the system.
- Component Diagram:
  - Illustrates dependencies between components at design time, compilation time and runtime

Deployment Diagram

- Deployment diagrams are useful for showing a system design after these system design decisions have been made:
  - Subsystem decomposition
  - Concurrency
  - Hardware/Software Mapping

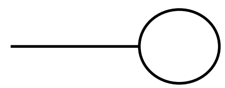
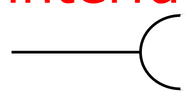


- A deployment diagram is a graph of nodes and connections ("communication associations")
  - Nodes are shown as 3-D boxes
  - Connections between nodes are shown as solid lines
  - Nodes may contain components
    - Components can be connected by "lollipops" and "grabbers"
    - Components may contain objects (indicating that the object is part of the component).

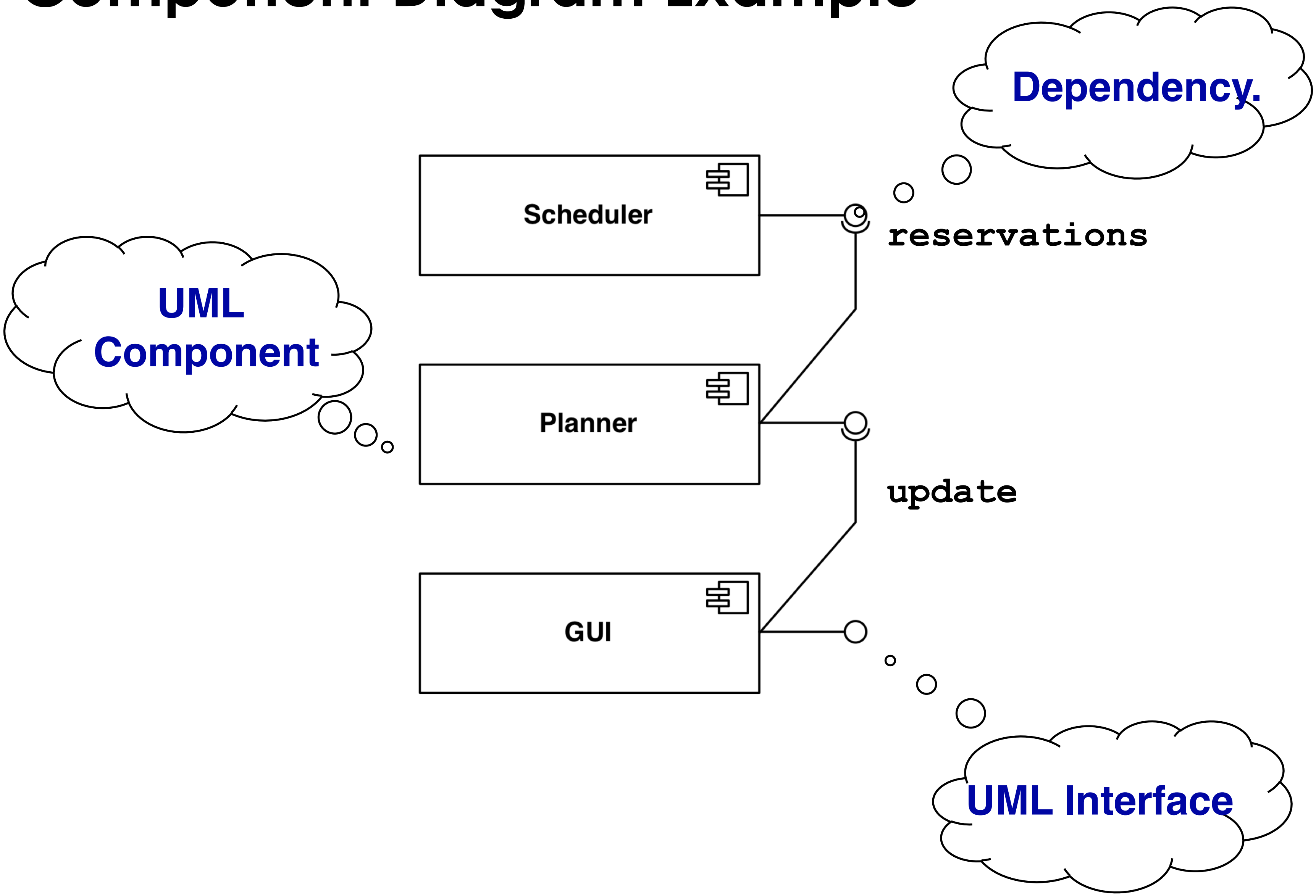
UML Component Diagram

- Used to model the top-level view of the system design in terms of components and dependencies among the components. Components can be
  - Source code, linkable libraries, executables
- The dependencies (edges in the graph) are shown as dashed lines with arrows from the client component to the supplier component:
  - The lines are often also called connectors
  - The types of dependencies are implementation language specific
- Informally also called “software wiring diagram“ because they show how the software components are wired together in the overall application.

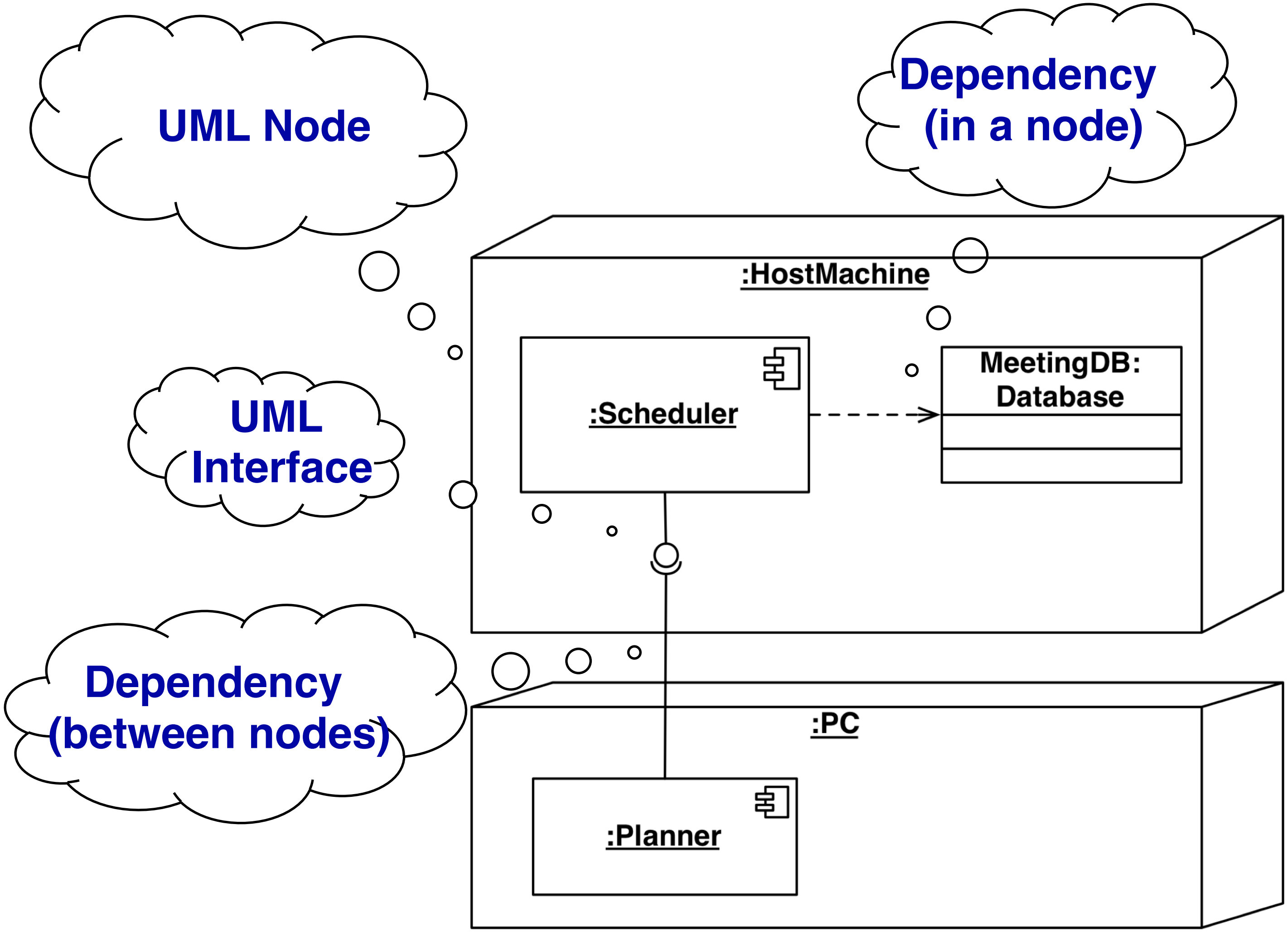
UML Interfaces: Lollipops and Sockets

- A UML interface describes a group of operations used or created by UML components.
  - There are two types of interfaces: provided and required interfaces.
    - A provided interface is modeled using the lollipop notation 
    - A required interface is modeled using the socket notation. 
- A port specifies a distinct interaction point between the component and its environment.
  - Ports are depicted as small squares on the sides of classifiers.

Component Diagram Example



Deployment Diagram Example



Data Management

- Some objects in the system model need to be persistent:
  - Values for their attributes have a lifetime longer than a single execution
- A persistent object can be realized with one of the following mechanisms:
  - Filesystem:
    - If the data are used by multiple readers but a single writer
  - Database:
    - If the data are used by concurrent writers and readers.

Data Management Questions

- How often is the database accessed?
  - What is the expected request (query) rate? The worst case?
  - What is the size of typical and worst case requests?
- Do the data need to be archived?
- Should the data be distributed?
  - Does the system design try to hide the location of the databases (location transparency)?
- Is there a need for a single interface to access the data?
  - What is the query format?
- Should the data format be extensible?

Mapping Object Models

- UML object models can be mapped to relational databases
- The mapping:
  - Each class is mapped to its own table
  - Each class attribute is mapped to a column in the table
  - An instance of a class represents a row in the table
- Methods are not mapped

Global Resource Handling

- Discusses access control
- Describes access rights for different classes of actors
- Describes how object guard against unauthorized access.

Defining Access Control

- In multi-user systems different actors usually have different access rights to different functionality and data
- How do we model these accesses?
  - During analysis we model them by associating different use cases with different actors
  - During system design we model them determining which objects are shared among actors.

Global Resource Questions

- Does the system need authentication?
- If yes, what is the authentication scheme?
  - User name and password? Access control list
  - Tickets? Capability-based
- What is the user interface for authentication?
- Does the system need a network-wide name server?
- How is a service known to the rest of the system?
  - At runtime? At compile time?
  - By Port?
  - By Name?

Control Flow

- How does the system sequence operations?
- Is the system event driven?
- Can it handle more than one user interaction at a time?
- The choice of control flow has an impact on the interfaces of subsystems.
    - If an event-driven control is selected, subsystems will provide event handlers.
    - If threads are selected, subsystems must guarantee mutual exclusion in critical sections.

Centralized vs. Decentralized Designs

- Centralized Design
  - One control object or subsystem ("spider") controls everything
    - Pro: Change in the control structure is very easy
    - Con: The single control object is a possible performance bottleneck
- Decentralized Design
  - Not a single object is in control, control is distributed; That means, there is more than one control object
    - Con: The responsibility is spread out
    - Pro: Fits nicely into object-oriented development

Boundary Conditions

- Initialization
  - The system is brought from a non-initialized state to steady-state
- Termination
  - Resources are cleaned up and other systems are notified upon termination
- Failure
  - Possible failures: Bugs, errors, external problems
- Good system design foresees fatal failures and provides mechanisms to deal with them.

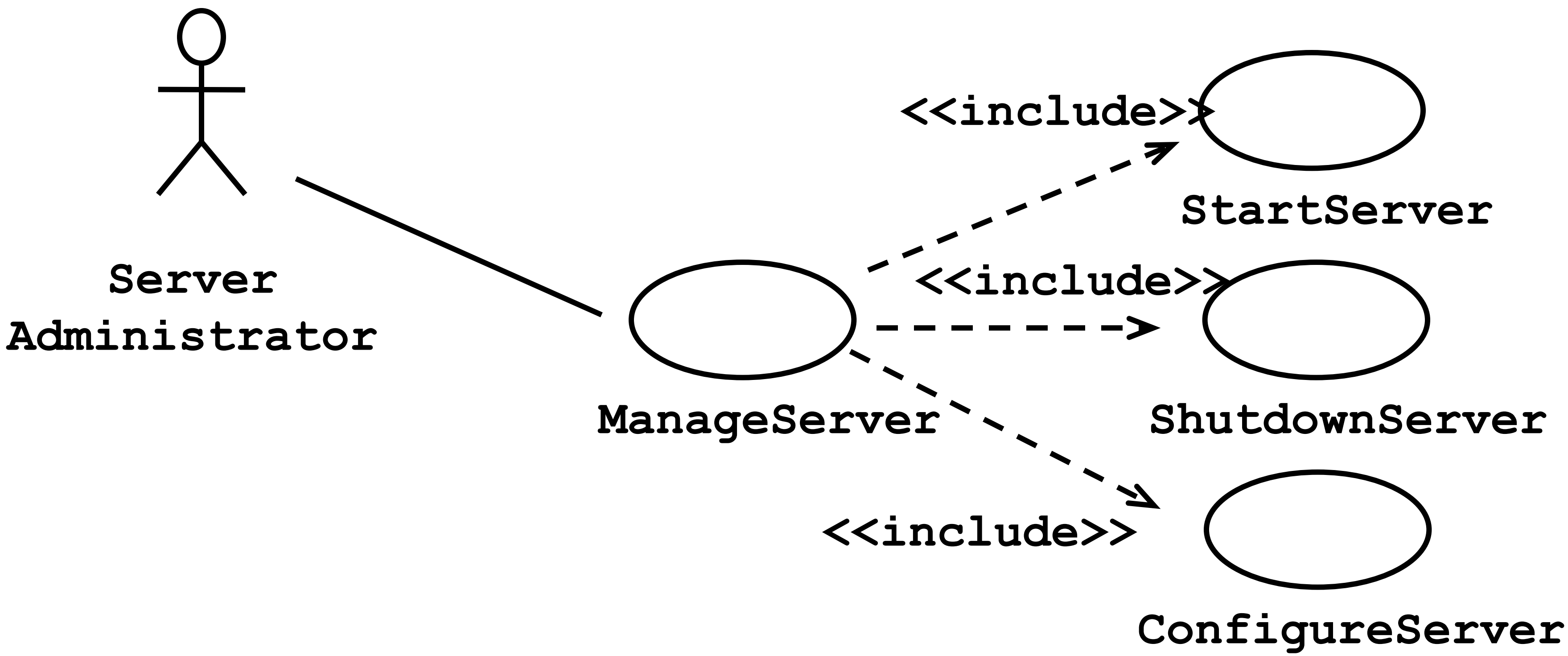
Boundary Condition Questions

- Initialization
  - What data need to be accessed at startup time?
  - What services have to be registered?
  - What does the user interface do at start up time?
- Termination
  - Are single subsystems allowed to terminate?
  - Are subsystems notified if a single subsystem terminates?
  - How are updates communicated to the database?
- Failure
  - How does the system behave when a node or communication link fails?
  - How does the system recover from failure?.

Modeling Boundary Conditions

- Boundary conditions are best modeled as use cases with actors and objects
- We call them boundary use cases or administrative use cases
- Actor: often the system administrator
- Interesting use cases:
  - Start up of a subsystem
  - Start up of the full system
  - Termination of a subsystem
  - Error in a subsystem or component, failure of a subsystem or component.

ManageServer Boundary Use Case



Summary

- System design activities:
  - Concurrency identification
  - Hardware/Software mapping
  - Persistent data management
  - Global resource handling
  - Software control selection
  - Boundary conditions
- Each of these activities may affect the subsystem decomposition
- Two new UML Notations
  - UML Component Diagram: Showing compile time and runtime dependencies between subsystems
  - UML Deployment Diagram: Drawing the runtime configuration of the system.