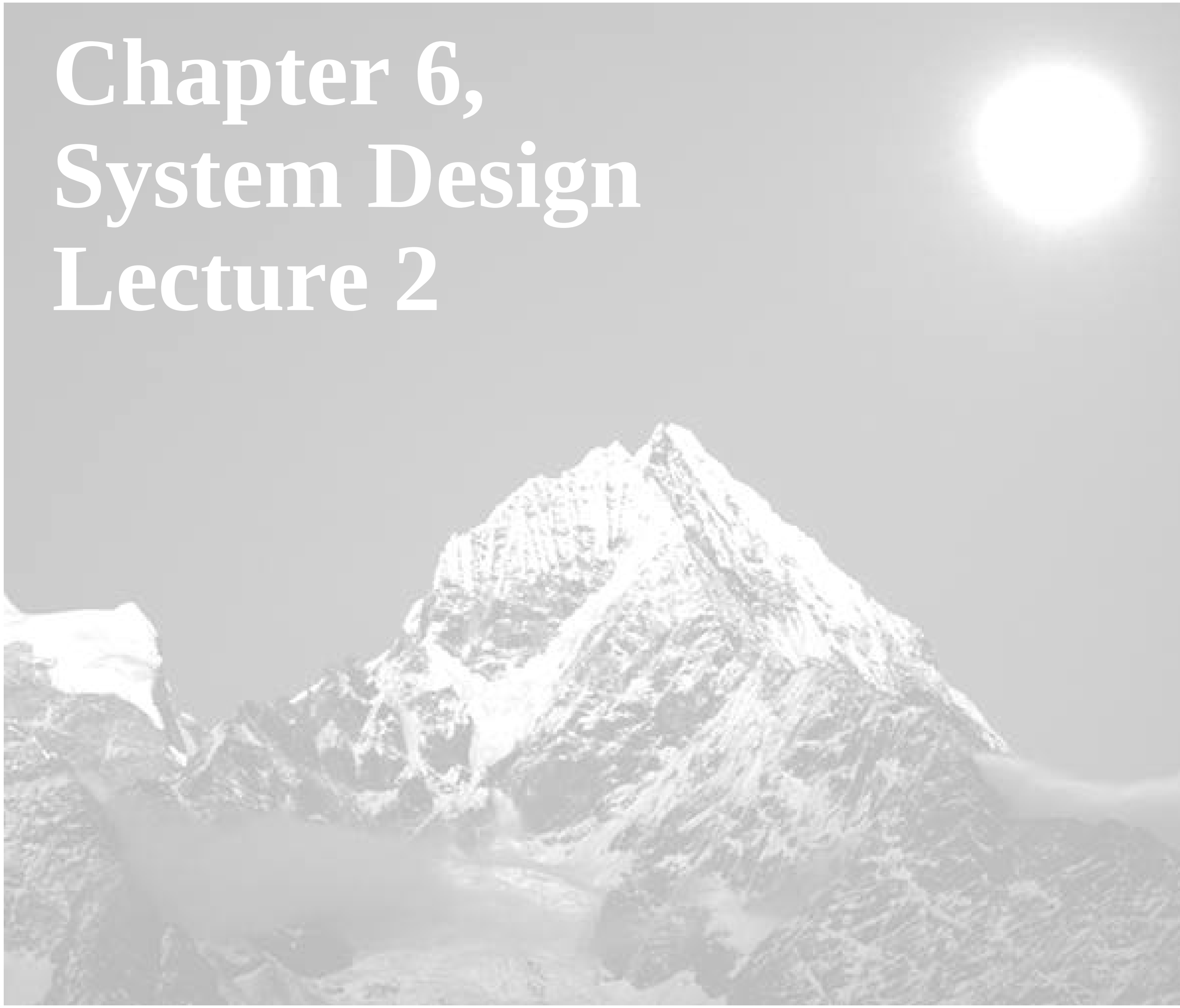
Object-Oriented Software Engineering
Conquering Complex and Changing Systems



Chapter 6, System Design Lecture 2

Preliminaries

- Guest lectures
  - **December 15: Andersen Consulting**
  - **December 22: Viant**
  - **January 18:   sd&m**
- Tomorrow December 1:
  - **Design Pattern I lecture by Tao Zhang**
- For STARS students:
  - **SDD milestone moved to December 8**
  - **SDD review moved to December 11**

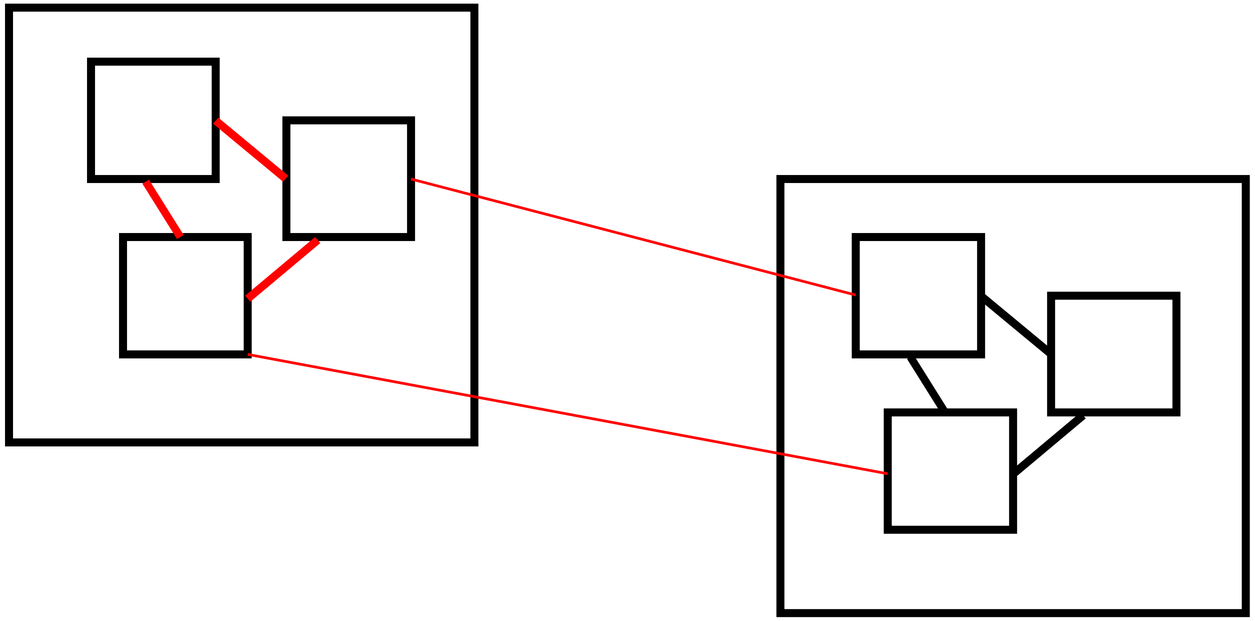
Exercise 6.1 from last week

6.1 Decomposing a system into subsystems reduces the complexity developers have to deal with by simplifying the parts and increasing their coherence. Decomposing a system into simpler parts usually results into increasing a different kind of complexity: Simpler parts also means a larger number of parts and interfaces.

If coherence is the guiding principle driving developers to decompose a system into small parts, which competing principle drives them to keep the total number of parts small?

Solution to exercise 6.1

- Decreasing coupling is the principle that competes with increasing coherence. A large number of parts will result in a large number of interfaces and many dependencies among the parts.



Exercises (continued)

6.2 In Section 6.4.2 on page 193 in the book, we classified design goals into five categories: performance, dependability, cost, maintenance, and end user. Assign one or more categories to each of the following goals:

- **Users must be given a feedback within 1 second after they issue any command.**
- **The TicketDistributor must be able to issue train tickets, even in the event of a network failure.**
- **The housing of the TicketDistributor must allow for new buttons to be installed in the event the number of different fares increases.**
- **The AutomatedTellerMachine must withstand dictionary attacks (i.e., users attempting to discover a identification number by systematic trial).**
- **The user interface of the system should prevent users from issuing commands in the wrong order.**

Solution to exercise 6.2

- Users must be given a feedback within 1 second after they issue any command. **[Performance]**
- The TicketDistributor must be able to issue train tickets, even in the event of a network failure. **[Dependability]**
- The housing of the TicketDistributor must allow for new buttons to be installed in the event the number of different fares increases. **[Maintenance]**
- The AutomatedTellerMachine must withstand dictionary attacks (i.e., users attempting to discover a identification number by systematic trial). **[Dependability]**
- The user interface of the system should prevent users from issuing commands in the wrong order. **[End user]**

Overview

System Design I (previous lecture)

- **0. Overview of System Design**
- **1. Design Goals**
- **2. Subsystem Decomposition**

System Design II

- **3. Concurrency**
- **4. Hardware/Software Mapping**
- **5. Persistent Data Management**
- **6. Global Resource Handling and Access Control**
- **7. Software Control**
- **8. Boundary Conditions**

3. Concurrency

- Identify concurrent threads and address concurrency issues.
- Design goal: response time, performance.

- Threads
  - **A *thread* of control is a path through a set of state diagrams on which a single object is active at a time.**
  - **A thread remains within a state diagram until an object sends an event to another object and waits for another event**
  - **Thread splitting: Object does a nonblocking send of an event.**

Concurrency (continued)

- ♦ Two objects are inherently concurrent if they can receive events at the same time without interacting
- ♦ Inherently concurrent objects should be assigned to different threads of control
- ♦ Objects with mutual exclusive activity should be folded into a single thread of control (Why?)

Concurrency Questions

- ♦ Which objects of the object model are independent?
- ♦ What kinds of threads of control are identifiable?
- ♦ Does the system provide access to multiple users?
- ♦ Can a single request to the system be decomposed into multiple requests? Can these requests be handled in parallel?

Implementing Concurrency

- Concurrent systems can be implemented on any system that provides
  - **physical concurrency (hardware)**

or

  - **logical concurrency (software)**

4. Hardware Software Mapping

- This activity addresses two questions:
  - **How shall we realize the subsystems: Hardware or Software?**
  - **How is the object model mapped on the chosen hardware & software?**
    - **Mapping Objects onto Reality: Processor, Memory, Input/Output**
    - **Mapping Associations onto Reality: Connectivity**
- Much of the difficulty of designing a system comes from meeting externally-imposed hardware and software constraints.
  - **Certain tasks have to be at specific locations**

Mapping the Objects

- Processor issues:
  - **Is the computation rate too demanding for a single processor?**
  - **Can we get a speedup by distributing tasks across several processors?**
  - **How many processors are required to maintain steady state load?**
- Memory issues:
  - **Is there enough memory to buffer bursts of requests?**
- I/O issues:
  - **Do you need an extra piece of hardware to handle the data generation rate?**
  - **Does the response time exceed the available communication bandwidth between subsystems or a task and a piece of hardware?**

Mapping the Subsystems Associations: Connectivity

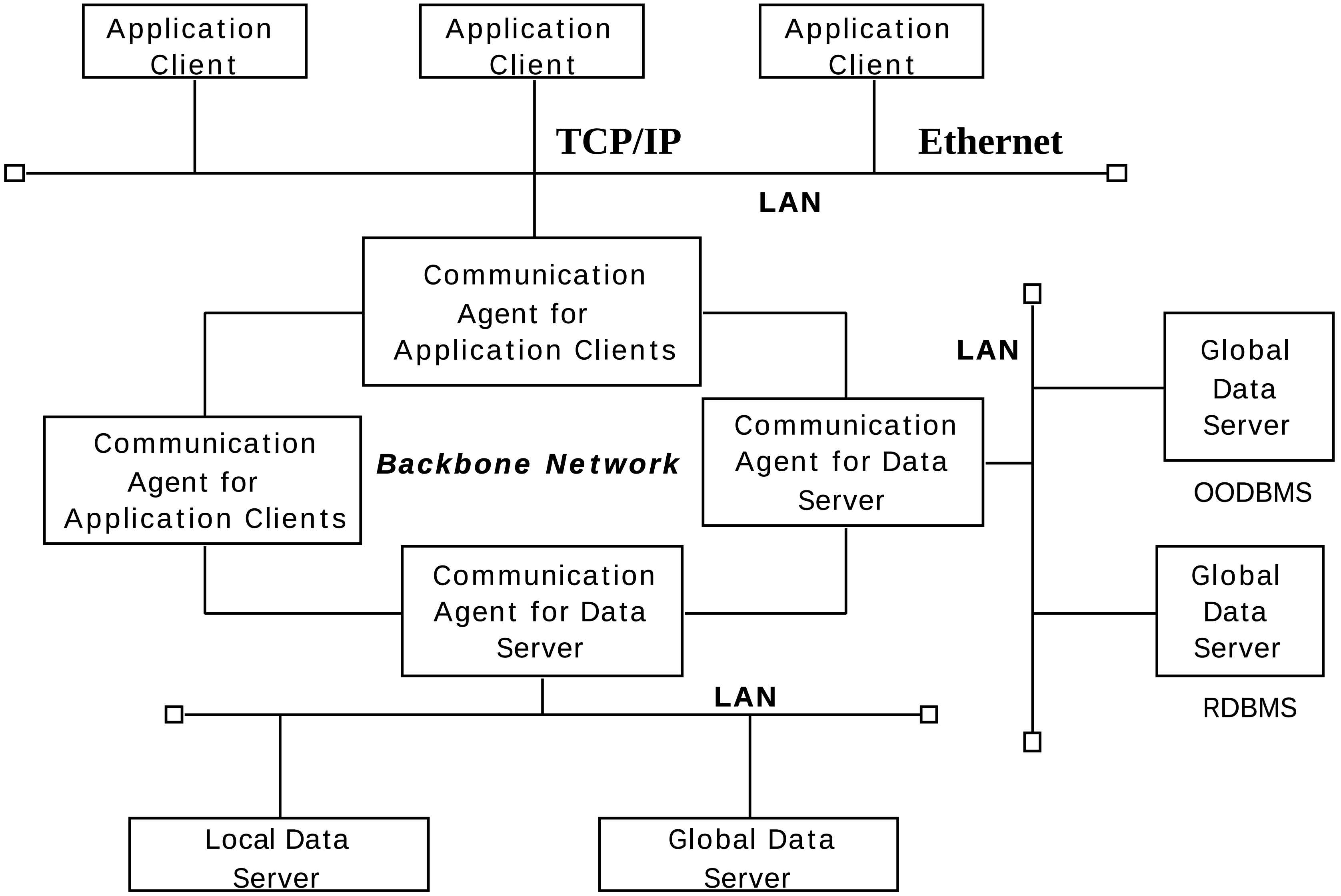
- Describe the *physical connectivity* of the hardware
  - **Often the physical layer in ISO's OSI Reference Model**
    - **Which associations in the object model are mapped to physical connections?**
    - **Which of the client-supplier relationships in the analysis/design model correspond to physical connections?**

- Describe the *logical connectivity* (subsystem associations)
  - **Identify associations that do not directly map into physical connections:**
    - **How should these associations be implemented?**

Connectivity in Distributed Systems

- If the architecture is distributed, we need to describe the network architecture (communication subsystem) as well.
- Questions to ask
  - **What are the transmission media? (Ethernet, Wireless)**
  - **What is the Quality of Service (QOS)? What kind of communication protocols can be used?**
  - **Should the interaction asynchronous, synchronous or blocking?**
  - **What are the available bandwidth requirements between the subsystems?**
    - **Stock Price Change -> Broker**
    - **Icy Road Detector -> ABS System**

Typical Example of a Physical Connectivity Drawing



Hardware/Software Mapping Questions

- What is the connectivity among physical units?
  - **Tree, star, matrix, ring**
- What is the appropriate communication protocol between the subsystems?
  - **Function of required bandwidth, latency and desired reliability**
- Is certain functionality already available in hardware?
- Do certain tasks require specific locations to control the hardware or to permit concurrent operation?
  - **Often true for embedded systems**
- General system performance question:
  - **What is the desired response time?**

Drawing Subsystems in UML

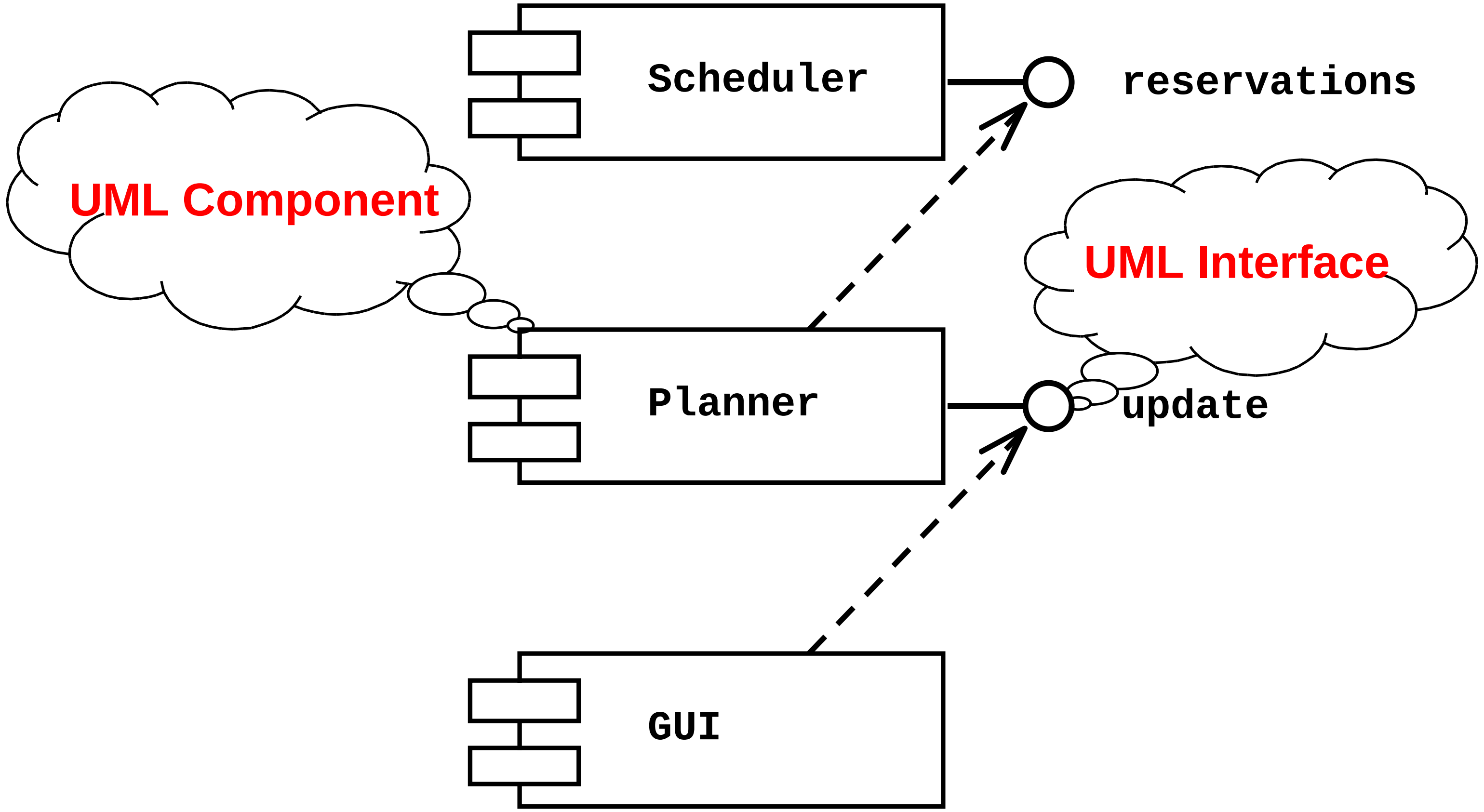
- System design must model static and dynamic structures:
  - **Component Diagrams for static structures**
    - **show the structure at design time or compilation time**
  - **Deployment Diagram for dynamic structures**
    - **show the structure of the run-time system**

- Note the lifetime of components
  - **Some exist only at design time**
  - **Others exist only until compile time**
  - **Some exist at link or runtime**

Component Diagram

- Component Diagram
  - **A graph of components connected by dependency relationships.**
  - **Shows the dependencies among software components**
    - **source code, linkable libraries, executables**
- Dependencies are shown as dashed arrows from the client component to the supplier component.
  - **The kinds of dependencies are implementation language specific.**
- A component diagram may also be used to show dependencies on a façade:
  - **Use dashed arrow the corresponding UML interface.**

Component Diagram Example

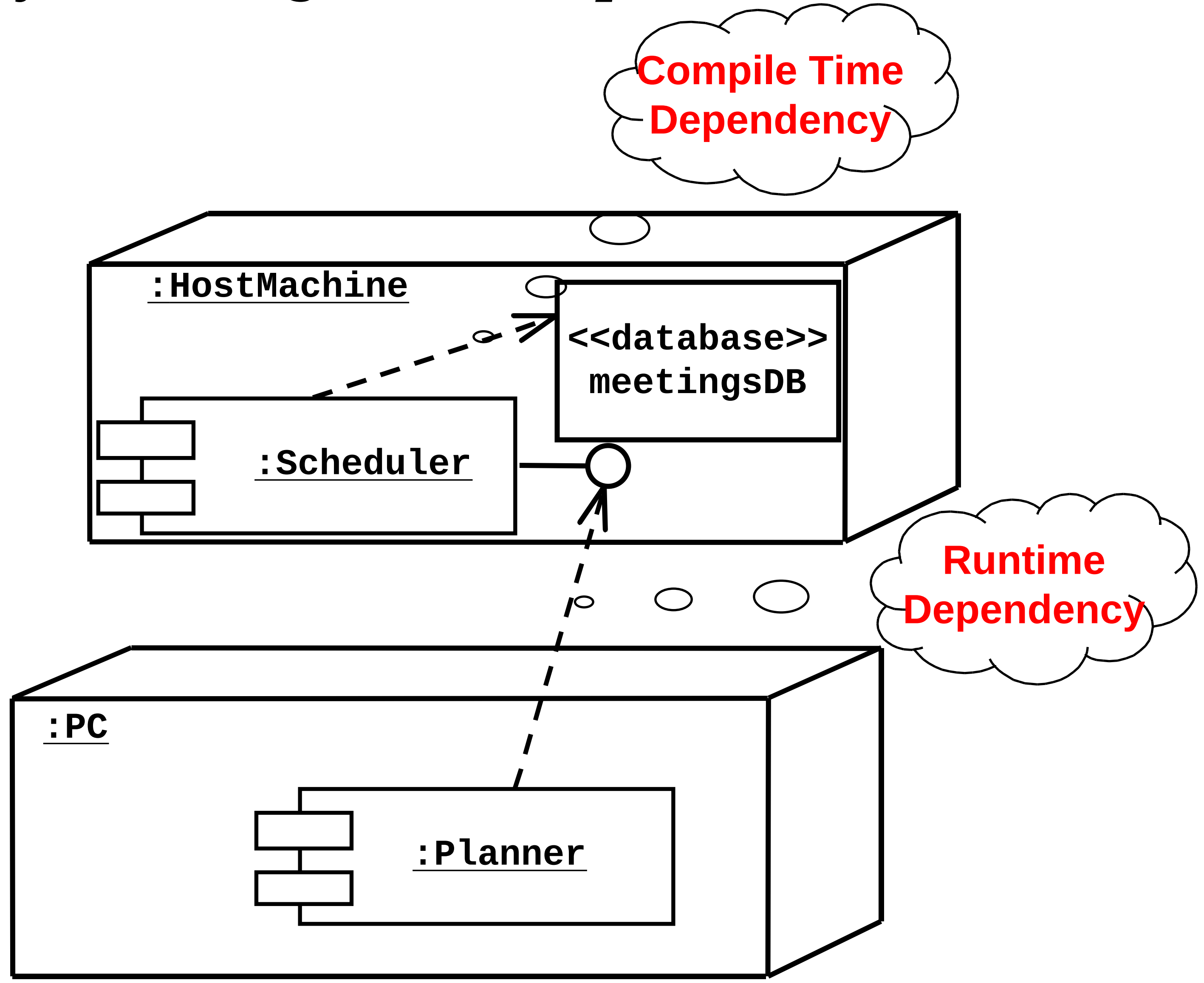


Deployment Diagram

- Deployment diagrams are useful for showing a system design after the following decisions are made
  - **Subsystem decomposition**
  - **Concurrency**
  - **Hardware/Software Mapping**

- A deployment diagram is a graph of nodes connected by communication associations.
  - **Nodes are shown as 3-D boxes.**
  - **Nodes may contain component instances.**
  - **Components may contain objects (indicating that the object is part of the component)**

Deployment Diagram Example



5. Data Management

- Some objects in the models need to be persistent
  - **Provide clean separation points between subsystems with well-defined interfaces.**
- A persistent object can be realized with one of the following
  - **Data structure**
    - **If the data can be volatile**
  - **Files**
    - **Cheap, simple, permanent storage**
    - **Low level (Read, Write)**
    - **Applications must add code to provide suitable level of abstraction**
  - **Database**
    - **Powerful, easy to port**
    - **Supports multiple writers and readers**

File or Database?

- When should you choose a file?
  - **Are the data voluminous (bit maps)?**
  - **Do you have lots of raw data (core dump, event trace)?**
  - **Do you need to keep the data only for a short time?**
  - **Is the information density low (archival files,history logs)?**
- When should you choose a database?
  - **Do the data require access at fine levels of details by multiple users?**
  - **Must the data be ported across multiple platforms (heterogeneous systems)?**
  - **Do multiple application programs access the data?**
  - **Does the data management require a lot of infrastructure?**

Database Management System

- Contains mechanisms for describing data, managing persistent storage and for providing a backup mechanism
- Provides concurrent access to the stored data
- Contains information about the data (“meta-data”), also called data schema.

Issues To Consider When Selecting a Database

- Storage space
  - **Database require about triple the storage space of actual data**
- Response time
  - **Mode databases are I/O or communication bound (distributed databases). Response time is also affected by CPU time, locking contention and delays from frequent screen displays**
- Locking modes
  - ***Pessimistic locking:* Lock before accessing object and release when object access is complete**
  - ***Optimistic locking:* Reads and writes may freely occur (high concurrency!) When activity has been completed, database checks if contention has occurred. If yes, all work has been lost.**
- Administration
  - **Large databases require specially trained support staff to set up security policies, manage the disk space, prepare backups, monitor performance, adjust tuning.**

Object-Oriented Databases

- Support all fundamental object modeling concepts
  - **Classes, Attributes, Methods, Associations, Inheritance**
- Mapping an object model to an OO-database
  - **Determine which objects are persistent.**
  - **Perform normal requirement analysis and object design**
  - **Create single attribute indices to reduce performance bottlenecks**
  - **Do the mapping (specific to commercially available product). Example:**
    - **In ObjectStore, implement classes and associations by preparing C++ declarations for each class and each association in the object model**

Relational Databases

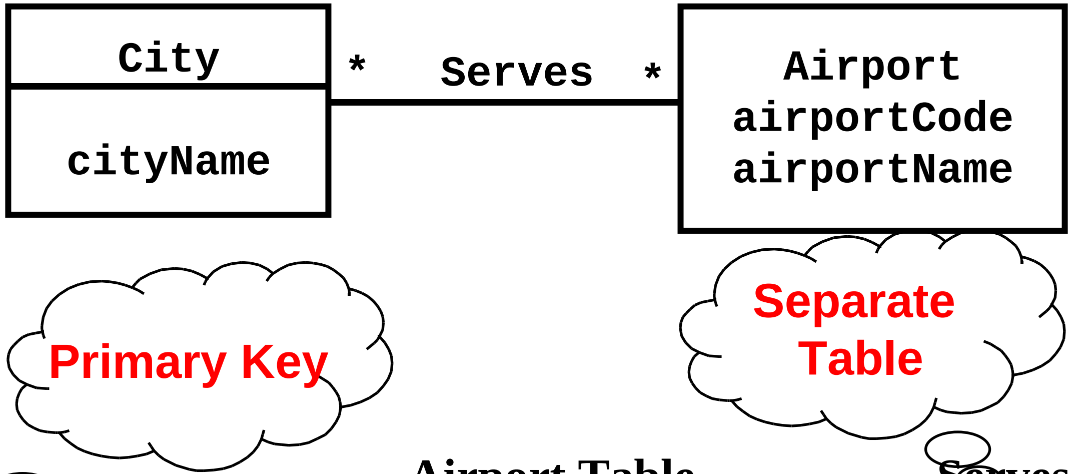
- Based on relational algebra
- Data is presented as 2-dimensional tables. Tables have a specific number of columns and and arbitrary numbers of rows
  - **Primary key: Combination of attributes that uniquely identify a row in a table. Each table should have only one primary key**
  - **Foreign key: Reference to a primary key in another table**
- SQL is the standard language defining and manipulating tables.
- Leading commercial databases support constraints.
  - **Referential integrity, for example, means that references to entries in other tables actually exist.**

Mapping an object model to a relational database

- UML object models can be mapped to relational databases:
  - **Some degradation occurs because all UML constructs must be mapped to a single relational database construct - the table.**
- UML mappings
  - **Each *class* is mapped to a table**
  - **Each class *attribute* is mapped onto a column in the table**
  - **An *instance* of a class represents a row in the table**
  - ***A many-to-many association* is mapped into its own table**
  - **A *one-to-many association* is implemented as buried foreign key**
- Methods are not mapped

Turning Object Models into Tables I

Many-to-Many Associations: Separate Table for Association



City Table

cityName
Houston
Albany
Munich
Hamburg

Airport Table

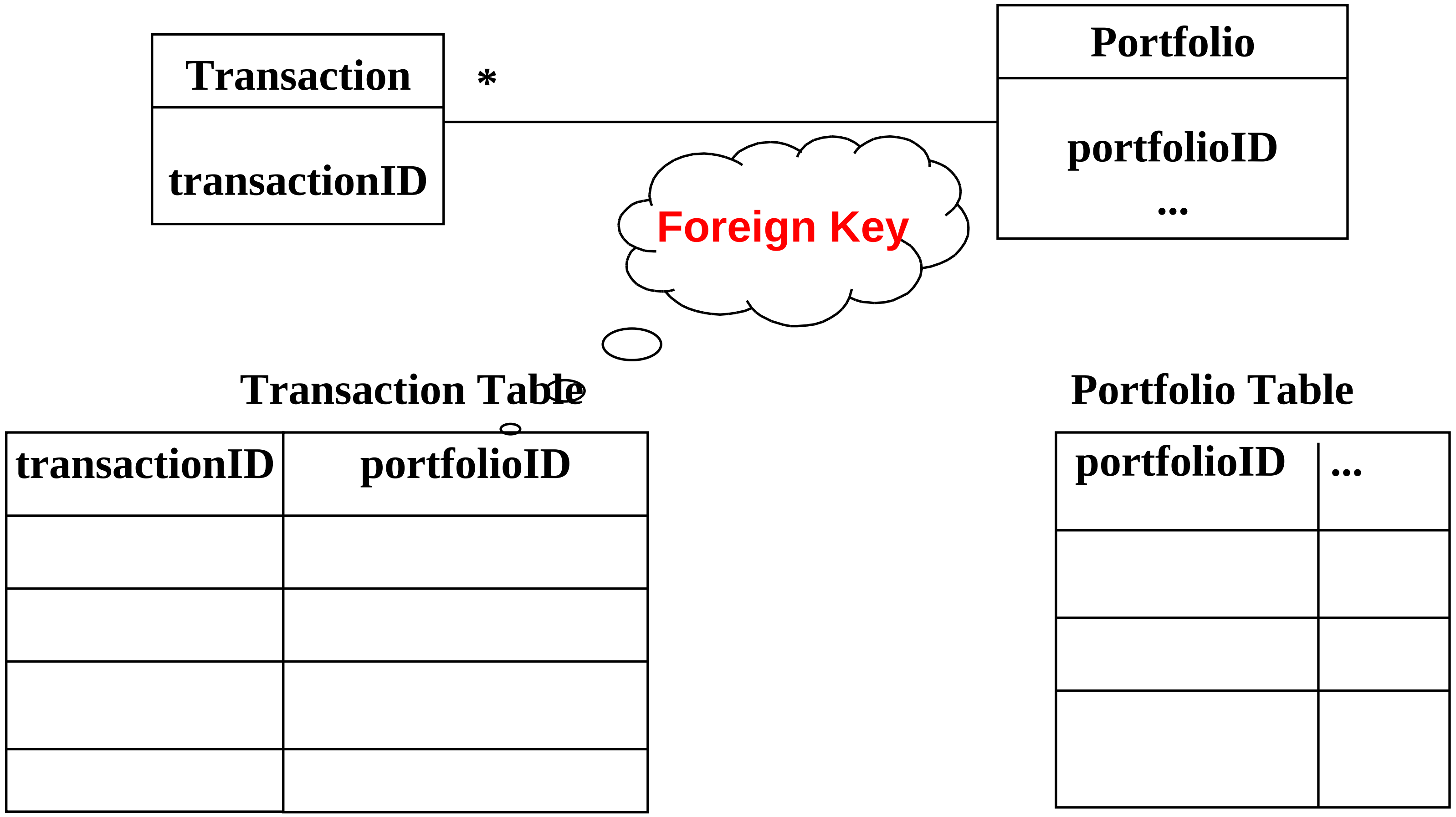
airportCode	airportName
IAH	Intercontinental
HOU	Hobby
ALB	Albany County
MUC	Munich Airport
HAM	Hamburg Airport

Serves Table

cityName	**airportCode**
Houston	IAH
Houston	HOU
Albany	ALB
Munich	MUC
Hamburg	HAM

Turning Object Models into Tables II

1-To-Many or Many-to-1 Associations: Buried Foreign Keys



Data Management Questions

- Should the data be distributed?
- Should the database be extensible?
- How often is the database accessed?
- What is the expected request (query) rate? In the worst case?
- What is the size of typical and worst case requests?
- Do the data need to be archived?
- 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 database be relational or object-oriented?

6. Global Resource Handling

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

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?**

7. Decide on Software Control

A. Choose implicit control (non-procedural or declarative languages)

- **Rule-based systems**
- **Logic programming**

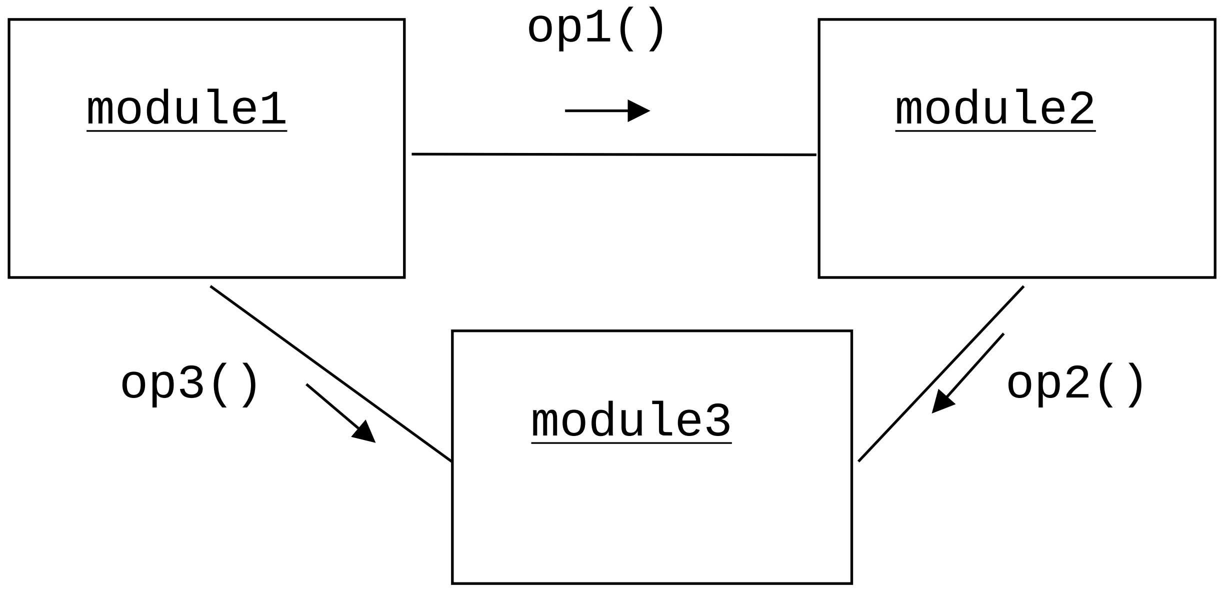
B. Or choose explicit control (procedural languages)

- **Centralized control**
  - **1. Procedure-driven control**
    - **Control resides within program code. Example: Main program calling procedures of subsystems.**
    - **Simple, easy to build**

Software Control (continued)

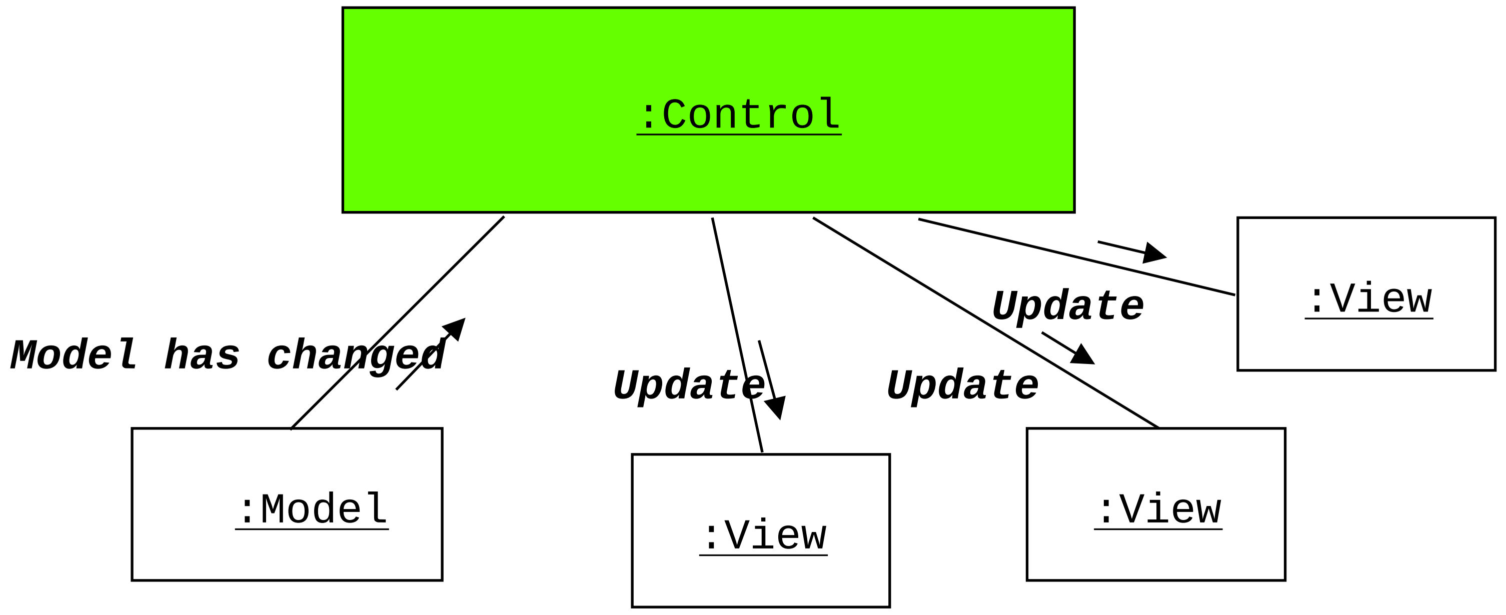
- ♦ **2. Event-driven control**
  - **– Control resides within a dispatcher who calls subsystem functions via callbacks.**
  - **– Flexible, good for user interfaces**
- ◆ **Decentralized control**
  - ♦ **Control resided in several independent objects (supported by some languages).**
  - ♦ **Possible speedup by parallelization, increased communication overhead.**
  - ♦ **Example: Message based system.**

Procedure-Driven Control Example



Event-Based System Example: MVC

- Smalltalk-80 Model-View-Controller
- Client/Server Architecture



Centralized vs. Decentralized Designs

- Should you use a centralized or decentralized design?
- Centralized Design
  - **One control object or subsystem ("spider") controls everything**
  - **Change in the control structure is very easy**
  - **Possible performance bottleneck**
- Decentralized Design
  - **Control is distributed**
  - **Spreads out responsibility**
  - **Fits nicely into object-oriented development**

8. Boundary Conditions

- Most of the system design effort is concerned with steady-state behavior.
- However, the system design phase must also address the initiation and finalization of the system.
  - **Initialization**
    - **Describes how the system is brought from an non initialized state to steady-state ("startup use cases”).**
  - **Termination**
    - **Describes what resources are cleaned up and which systems are notified upon termination ("termination use cases").**
  - **Failure**
    - **Many possible causes: Bugs, errors, external problems (power supply).**
    - **Good system design foresees fatal failures (“failure use cases”).**

Boundary Condition Questions

- 8.1 Initialization
  - **How does the system start up?**
    - **What data need to be accessed at startup time?**
    - **What services have to registered?**
  - **What does the user interface do at start up time?**
    - **How does it present itself to the user?**
- 8.2 Termination
  - **Are single subsystems allowed to terminate?**
  - **Are other subsystems notified if a single subsystem terminates?**
  - **How are local updates communicated to the database?**
- 8.3 Failure
  - **How does the system behave when a node or communication link fails? Are there backup communication links?**
  - **How does the system recover from failure? Is this different from initialization?**

Summary

In this lecture, we reviewed the activities of system design :

- ♦ Concurrency identification
- ♦ Hardware/Software mapping
- ♦ Persistent data management
- ♦ Global resource handling
- ♦ Software control selection
- ♦ Boundary conditions

Each of these activities revises the subsystem decomposition to address a specific issue. Once these activities are completed, the interface of the subsystems can be defined.

Exercises

6.4 Consider a legacy, fax-based, problem-reporting system for an aircraft manufacturer. You are part of a reengineering project replacing the core of the system by a computer-based system, which includes a database and a notification system. The client requires the fax to remain an entry point for problem reports. You propose an E-mail entry point.

Describe a subsystem decomposition, and possibly a design pattern, which would allow both interfaces.

Exercises (cont’d)

6.5 You are designing the access control policies for a Web-based retail store:

- **Customers access the store via the Web, browse product information, input their address and payment information, and purchase products.**
- **Suppliers can add new products, update product information, and receive orders.**
- **The store owner sets the retail prices, makes tailored offers to customers based on their purchasing profiles, and provides marketing services.**

You have to deal with three actors: StoreAdministrator, Supplier, and Customer. Design an access control policy for all three actors. Customers can be created via the Web, whereas Suppliers are created by the StoreAdministrator.