Chapter 4 — Design

May 13, 2009

Outline

1. Introduction
2. Architectural Design
   - System Organization
   - Distributed Systems Architecture
3. Application Architectures
   - Types of AAs
4. Object-Oriented Design
   - Object-Oriented Design Process
5. Interface Design

Introduction

- Logical organization of software

Introduction

- Logical organization of software
  - UML model

Introduction

- Logical organization of software
  - UML model
  - sketches, drawings, notes

Introduction

- Logical organization of software
  - UML model
  - sketches, drawings, notes
- *creative process*: no real “method”

Introduction

- Logical organization of software
  - UML model
  - sketches, drawings, notes
- *creative process*: no real “method”
- learn from previous design experiences

In This Chapter

- Learn about software design perspectives and ideas

In This Chapter

- Learn about software design perspectives and ideas
- **Architecture**: abstract formulas and patterns

In This Chapter

- Learn about software design perspectives and ideas
- **Architecture**: abstract formulas and patterns
- **Specific Issues**: common to most software designs

In This Chapter

- Learn about software design perspectives and ideas
- **Architecture**: abstract formulas and patterns
- **Specific Issues**: common to most software designs
  - object-oriented design
  - user-interface design

Outline

1. Introduction
2. Architectural Design
   - System Organization
   - Distributed Systems Architecture
3. Application Architectures
   - Types of AAs
4. Object-Oriented Design
   - Object-Oriented Design Process
5. Interface Design

Introduction

- System is composed of sub-systems

Introduction

- System is composed of sub-systems
- *Architectural Design*:
    - identify sub-systems
    - framework for sub-system control and communication

Introduction

- System is composed of sub-systems
- *Architectural Design*:
  - identify sub-systems
  - framework for sub-system control and communication
- affects *performance*, *robustness*, *availability*, *maintainability*

Outline

1. Introduction

2. Architectural Design
   - System Organization
   - Distributed Systems Architecture

3. Application Architectures
   - Types of AAs

4. Object-Oriented Design
   - Object-Oriented Design Process

5. Interface Design

Organization

- Reflects basic strategy for the structure

Organization

- Reflects basic strategy for the structure
- Common organization styles:
  - repository
  - client-server
  - layered

Repository Model

- If large amounts of data: central, shared *repository* or DB

Repository Model

- If large amounts of data: central, shared *repository* or DB
  - otherwise: DBs for each subsystem

Repository Model

- If large amounts of data: central, shared *repository* or DB
  - otherwise: DBs for each subsystem
- Advantages / Disadvantages:

Repository Model

- If large amounts of data: central, shared *repository* or DB
  - otherwise: DBs for each subsystem
- Advantages / Disadvantages:
  - efficient for sharing (*all sub-systems must agree on data model*)

Repository Model

- If large amounts of data: central, shared *repository* or DB
    - otherwise: DBs for each subsystem
- Advantages / Disadvantages:
    - efficient for sharing (*all sub-systems must agree on data model*)
    - backup, security, access control are centralized (*agreement*)

Client-Server Architecture Model

- System as a set of services

Client-Server Architecture Model

- System as a set of services
- Components:

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*
  - *clients*: consumer services

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*
  - *clients*: consumer services
  - *network*: connects the two

Client-Server Architecture Model

- System as a set of services
- Components:
    - *servers*: offer services.
        - e.g., *web*, *print*, *file*, *compile*
    - *clients*: consumer services
    - *network*: connects the two
        - not always necessary if not distributed

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*
  - *clients*: consumer services
  - *network*: connects the two
    - not always necessary if not distributed
- Clients must know *name*, *services* of available servers

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*
  - *clients*: consumer services
  - *network*: connects the two
    - not always necessary if not distributed
- Clients must know *name*, *services* of available servers
  - e.g., DNS

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*
  - *clients*: consumer services
  - *network*: connects the two
    - not always necessary if not distributed
- Clients must know *name*, *services* of available servers
  - e.g., DNS
  - not vice-versa

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*
  - *clients*: consumer services
  - *network*: connects the two
    - not always necessary if not distributed
- Clients must know *name*, *services* of available servers
  - e.g., DNS
  - not vice-versa
- Clients access services using RPC with a request-reply protocol

Client-Server Architecture Model

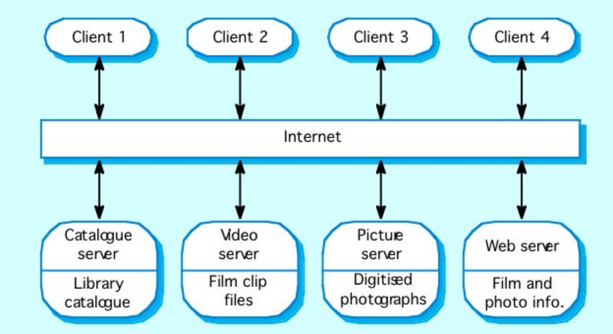
- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*
  - *clients*: consumer services
  - *network*: connects the two
    - not always necessary if not distributed
- Clients must know *name*, *services* of available servers
  - e.g., DNS
  - not vice-versa
- Clients access services using RPC with a request-reply protocol
  - e.g., `http` protocol for WWW

Client-Server Architecture Model

- System as a set of services
- Components:
  - *servers*: offer services.
    - e.g., *web*, *print*, *file*, *compile*
  - *clients*: consumer services
  - *network*: connects the two
    - not always necessary if not distributed
- Clients must know *name*, *services* of available servers
  - e.g., DNS
  - not vice-versa
- Clients access services using RPC with a request-reply protocol
  - e.g., `http` protocol for WWW
- Client *waits* for a reply

Client-Server Architecture Model

- E.g.: Video and Photo Library system's architecture



Client-Server Architecture Model

- Advantages:

Client-Server Architecture Model

- Advantages:
    - Distributed architecture: processing, storage

Client-Server Architecture Model

- Advantages:
    - Distributed architecture: processing, storage
    - make changes transparently

Client-Server Architecture Model

- Advantages:
  - Distributed architecture: processing, storage
  - make changes transparently
- Best used if using shared data model

Client-Server Architecture Model

- Advantages:
  - Distributed architecture: processing, storage
  - make changes transparently
- Best used if using shared data model
  - otherwise may need to translate for each sub-system

Client-Server Architecture Model

- Advantages:
  - Distributed architecture: processing, storage
  - make changes transparently
- Best used if using shared data model
  - otherwise may need to translate for each sub-system
  - XML can be used, but slow

Layered Architecture Model

- Organize sub-systems in layers

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine
- Contrast this with client-server model

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine
- Contrast this with client-server model
    - hierarchy vs. network

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine
- Contrast this with client-server model
  - hierarchy vs. network
- Advantages:

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine
- Contrast this with client-server model
    - hierarchy vs. network
- Advantages:
    - supports incremental development

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine
- Contrast this with client-server model
    - hierarchy vs. network
- Advantages:
    - supports incremental development
    - localize machine dependence in lower layers

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine
- Contrast this with client-server model
  - hierarchy vs. network
- Advantages:
  - supports incremental development
  - localize machine dependence in lower layers
- Disadvantages:

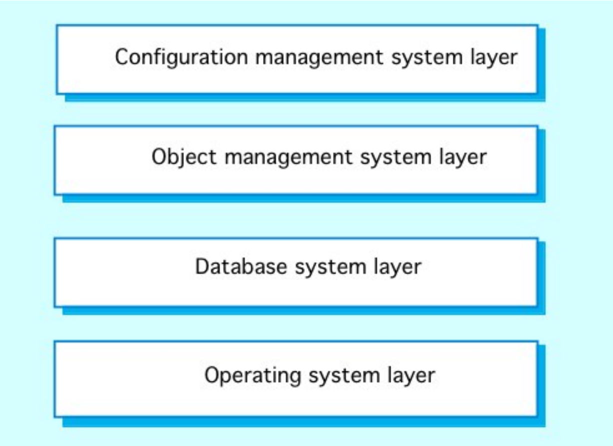
Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine
- Contrast this with client-server model
  - hierarchy vs. network
- Advantages:
  - supports incremental development
  - localize machine dependence in lower layers
- Disadvantages:
  - may have to “punch through” from top to bottom

Layered Architecture Model

- Organize sub-systems in layers
- Also “abstract machine model”
- Each layer ≈ an abstract machine
- Contrast this with client-server model
  - hierarchy vs. network
- Advantages:
  - supports incremental development
  - localize machine dependence in lower layers
- Disadvantages:
  - may have to “punch through” from top to bottom
  - performance may suffer

Layered Architecture Model



Configuration management system layer

Object management system layer

Database system layer

Operating system layer

Figure: Version Control System

Reference architectures

- Represents architecture for a particular domain

Reference architectures

- Represents architecture for a particular domain
    - several systems will share the architecture

Reference architectures

- Represents architecture for a particular domain
  - several systems will share the architecture
  - Can be resued

Reference architectures

- Represents architecture for a particular domain
  - several systems will share the architecture
  - Can be resued
- Not for direct implementation

Reference architectures

- Represents architecture for a particular domain
  - several systems will share the architecture
  - Can be resued
- Not for direct implementation
  - communicate domain concepts

Reference architectures

- Represents architecture for a particular domain
  - several systems will share the architecture
  - Can be resued
- Not for direct implementation
  - communicate domain concepts
  - evaluate possible architectures

Example Reference Architecture

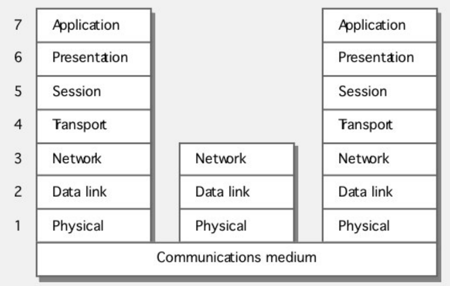


Figure: The OSI Reference Architecture for Computer Networks

Outline

1 Introduction

2 Architectural Design
- System Organization
- Distributed Systems Architecture

3 Application Architectures
- Types of AAs

4 Object-Oriented Design
- Object-Oriented Design Process

5 Interface Design

Introduction

- Information processing is distributed over several computers

Introduction

- Information processing is distributed over several computers
- Advantages:

Introduction

- Information processing is distributed over several computers
- Advantages:
  - Resource sharing

Introduction

- Information processing is distributed over several computers
- Advantages:
  - Resource sharing
  - Openness

Introduction

- Information processing is distributed over several computers
- Advantages:
  - Resource sharing
  - Openness
  - Concurrency

Introduction

- Information processing is distributed over several computers
- Advantages:
  - Resource sharing
  - Openness
  - Concurrency
  - Scalability

Introduction

- Information processing is distributed over several computers
- Advantages:
    - Resource sharing
    - Openness
    - Concurrency
    - Scalability
    - Fault Tolerance

Introduction

- Information processing is distributed over several computers
- Advantages:
    - Resource sharing
    - Openness
    - Concurrency
    - Scalability
    - Fault Tolerance
- Disadvantages:

Introduction

- Information processing is distributed over several computers
- Advantages:
    - Resource sharing
    - Openness
    - Concurrency
    - Scalability
    - Fault Tolerance
- Disadvantages:
    - Complexity

Introduction

- Information processing is distributed over several computers
- Advantages:
  - Resource sharing
  - Openness
  - Concurrency
  - Scalability
  - Fault Tolerance
- Disadvantages:
  - Complexity
  - Security

Introduction

- Information processing is distributed over several computers
- Advantages:
    - Resource sharing
    - Openness
    - Concurrency
    - Scalability
    - Fault Tolerance
- Disadvantages:
    - Complexity
    - Security
    - Manageability

Introduction

- Information processing is distributed over several computers
- Advantages:
  - Resource sharing
  - Openness
  - Concurrency
  - Scalability
  - Fault Tolerance
- Disadvantages:
  - Complexity
  - Security
  - Manageability
  - Unpredictability

Multi-processor Architecture

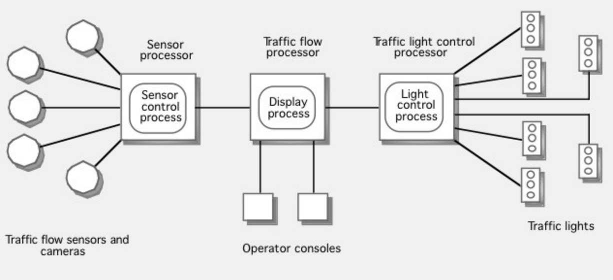


Figure: A multiprocessor traffic control system

Client-Server Architecture (CSA)

Difference between server *process* and server *computer*

CSA Example

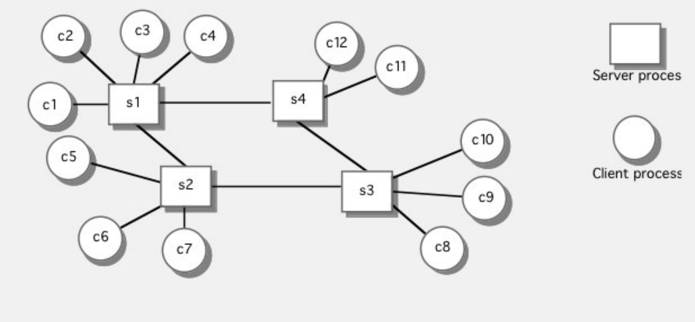


Figure: A client-server system

CSA Example

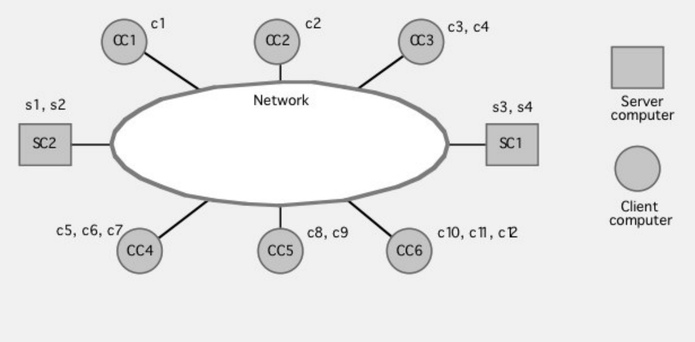
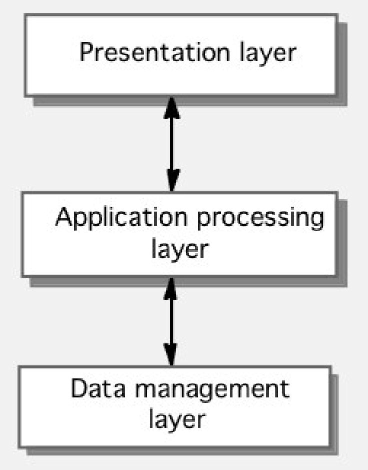


Figure: Computers in a client-server network

How to CSA-ize Your System

Distribute some/all of the MVC layers

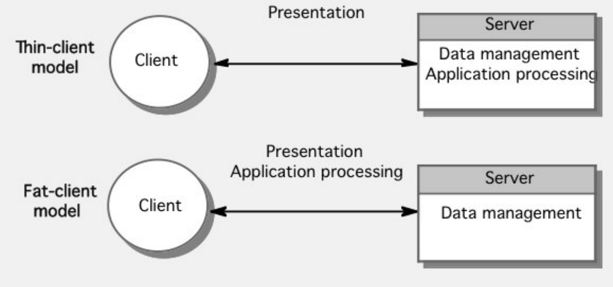


Two-tier CSA

- Simplest CSA: a server(s) and clients

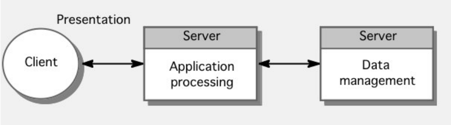
Two-tier CSA

- Simplest CSA: a server(s) and clients
- *Thin* and *flat* client models



Three-tier CSA

- Each MVC layer on a separate computer



Other Distributed Architectures

- Peer-to-peer

Other Distributed Architectures

- Peer-to-peer
  - centralized

Other Distributed Architectures

- Peer-to-peer
  - centralized
  - semi-centralized

Other Distributed Architectures

- Peer-to-peer
    - centralized
    - semi-centralized
- Service-oriented Architecture (e.g., web services)

Other Distributed Architectures

- Peer-to-peer
  - centralized
  - semi-centralized
- Service-oriented Architecture (e.g., web services)
  - web beyond browsers

Other Distributed Architectures

- Peer-to-peer
  - centralized
  - semi-centralized
- Service-oriented Architecture (e.g., web services)
  - web beyond browsers
  - standards (based on XML)
    - SOAP
    - WSDL
    - UDDI

Outline

1. Introduction
2. Architectural Design
   - System Organization
   - Distributed Systems Architecture
3. Application Architectures
   - Types of AAs
4. Object-Oriented Design
   - Object-Oriented Design Process
5. Interface Design

Introduction

- Look at system architecture from application's perspective

Introduction

- Look at system architecture from application's perspective
  - previous perspective: control, distribution, structure

Introduction

- Look at system architecture from application's perspective
  - previous perspective: control, distribution, structure
- Issues to common to applications of a certain kind

Use of AA

- As s/w developer, AAs are useful as
  - starting point for design process

Use of AA

- As s/w developer, AAs are useful as
    - starting point for design process
    - design checklist

Use of AA

- As s/w developer, AAs are useful as
  - starting point for design process
  - design checklist
  - organize team-specific work

Use of AA

- As s/w developer, AAs are useful as
  - starting point for design process
  - design checklist
  - organize team-specific work
  - assessing components for reuse

Outline

1 Introduction

2 Architectural Design
- System Organization
- Distributed Systems Architecture

3 Application Architectures
- Types of AAs

4 Object-Oriented Design
- Object-Oriented Design Process

5 Interface Design

Data-Processing Systems

- Batch-processing of data

Data-Processing Systems

- Batch-processing of data
    - input and output: sizable databases / data stores

Data-Processing Systems

- Batch-processing of data
  - input and output: sizable databases / data stores

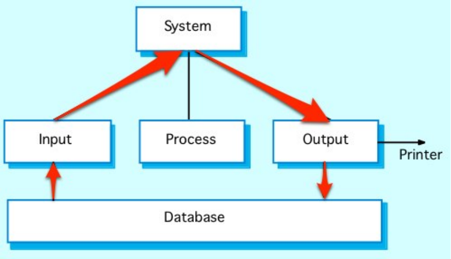


Figure: Model of data-processing applications

Data-Processing Systems

- Batch-processing of data
    - input and output: sizable databases / data stores

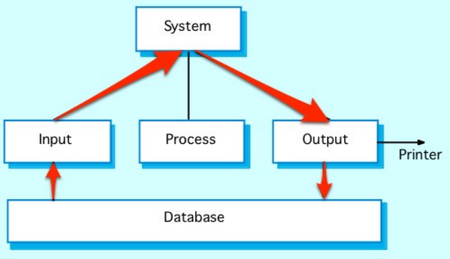


Figure: Model of data-processing applications

- Do not need to save state across transactions

Data-Processing Systems

- Batch-processing of data
    - input and output: sizable databases / data stores

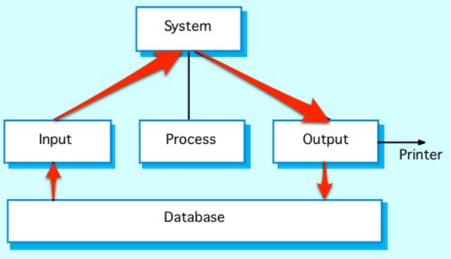


Figure: Model of data-processing applications

- Do not need to save state across transactions
    - ∴, function-oriented rather than OO

Data-Flow Diagrams

- DFDs are useful to describe data-processing applications

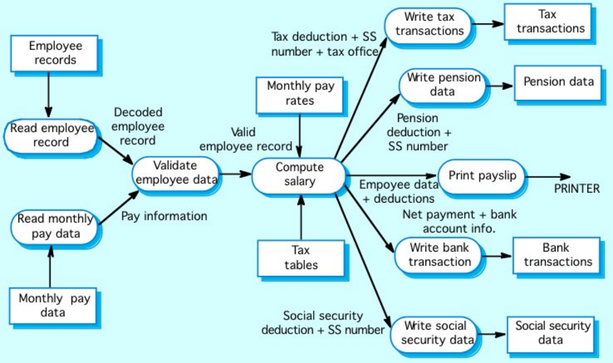


Figure: DFD for a payroll system

Transaction-Processing Systems

- Process user requests for DB read / update

Transaction-Processing Systems

- Process user requests for DB read / update
- Could be event-driven (interactive) or procedural (non-interactive)

Transaction-Processing Systems

- Process user requests for DB read / update
- Could be event-driven (interactive) or procedural (non-interactive)
- E.g.: medical records software, ATM

Transaction-Processing Systems

- Process user requests for DB read / update
- Could be event-driven (interactive) or procedural (non-interactive)
- E.g.: medical records software, ATM

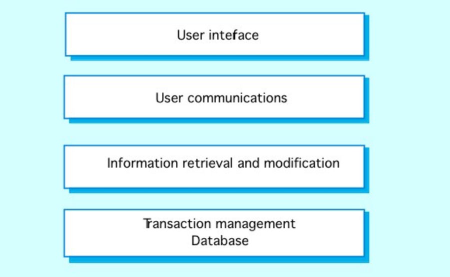


Figure: Layered architecture of a transaction processing system

Event Processing Systems

- Respond to user or system events

Event Processing Systems

- Respond to user or system events
    - e.g. of such events?

Event Processing Systems

- Respond to user or system events
    - e.g. of such events?

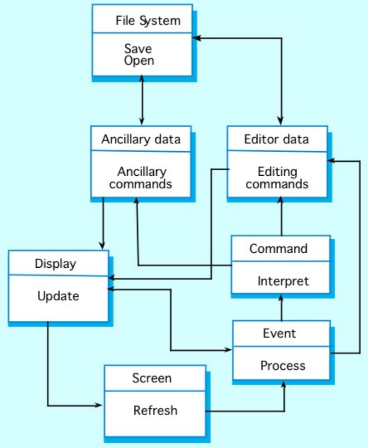


Figure: Architecture of Event-driven system

Outline

1 Introduction

2 Architectural Design
- System Organization
- Distributed Systems Architecture

3 Application Architectures
- Types of AAs

4 Object-Oriented Design
- Object-Oriented Design Process

5 Interface Design

Introduction

Object-oriented system: interacting objects that maintain their own state and provide operations on those states

Introduction

Object-oriented system: interacting objects that maintain their own state and provide operations on those states
Object-oriented design: system designing with object classes and with relationships between these classes

- classes are related to problem

Introduction

Object-oriented system: interacting objects that maintain their own state and provide operations on those states
Object-oriented design: system designing with object classes and with relationships between these classes

- classes are related to problem
- state representation is private

Introduction

Object-oriented system: interacting objects that maintain their own state and provide operations on those states
Object-oriented design: system designing with object classes and with relationships between these classes

- classes are related to problem
- state representation is private
- system is easy to modify → objects (classes) since they are independent
- objects are reusable

Objects and classes

- **Object**: entity{state, operations}

Objects and classes

- **Object**: entity{state, operations}
- *state* provides object information

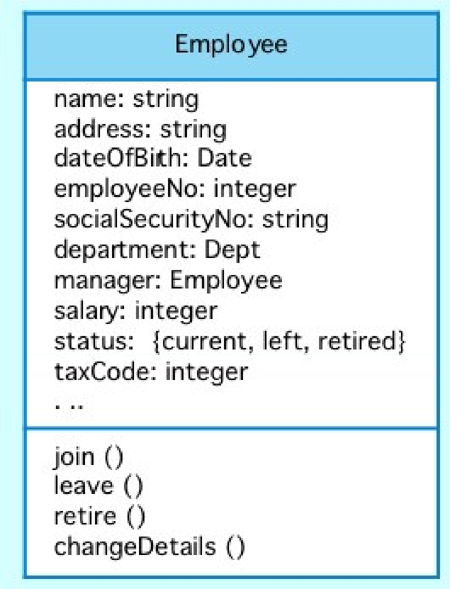
Objects and classes

- **Object**: entity{state, operations}
- *state* provides object information
- *operations* provide services to other objects

Objects and classes

- **Object**: entity{state, operations}
- *state* provides object information
- *operations* provide services to other objects
- Objects created from a *class*: definition of template

An Employee object class (UML)

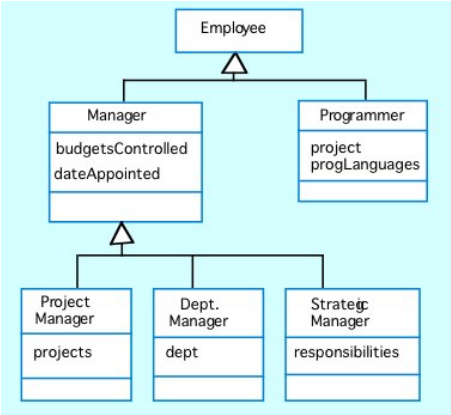


Employee

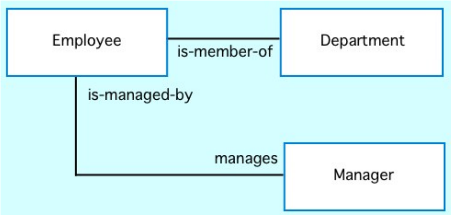
name: string
address: string
dateOfBirth: Date
employeeNo: integer
socialSecurityNo: string
department: Dept
manager: Employee
salary: integer
status: {current, left, retired}
taxCode: integer
. ..

join ()
leave ()
retire ()
changeDetails ()

Example Class Hierarchy (Generalization)



Association Model



Concurrent Objects

- Objects execute concurrently

Concurrent Objects

- Objects execute concurrently
- However, service requests are *procedural*

Concurrent Objects

- Objects execute concurrently
- However, service requests are *procedural*
- Threads allow for full concurrency even with service requests

Outline

1. Introduction
2. Architectural Design
   - System Organization
   - Distributed Systems Architecture
3. Application Architectures
   - Types of AAs
4. Object-Oriented Design
   - Object-Oriented Design Process
5. Interface Design

The Process

The Process

- Steps in the process:
  - Understand context and use
  - Design system architecture
  - Identify main objects
  - Design system models
  - Specify object interfaces

The Process

- Steps in the process:
  - Understand context and use
  - Design system architecture
  - Identify main objects
  - Design system models
  - Specify object interfaces
- Usually an iterative, interleaved process

Example: a weather mapping system

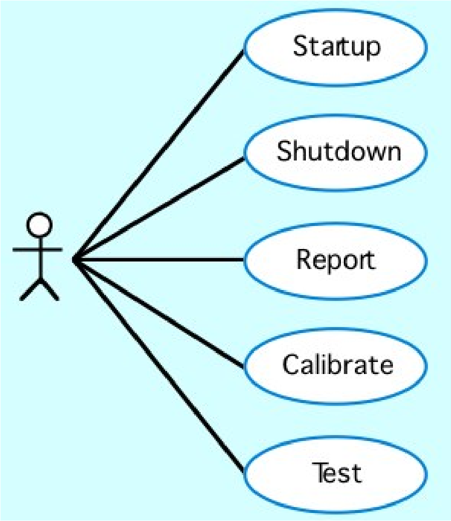
Basics

- System description
  - generates weather maps from data collected from several sources
  - collect and integrate data into an archive
  - use archive and digitized map to display/print weather map

Use Case and Context

- Context: how the system is connected in its environment
- Use case: ways in which system can be used

Use-cases for weather station

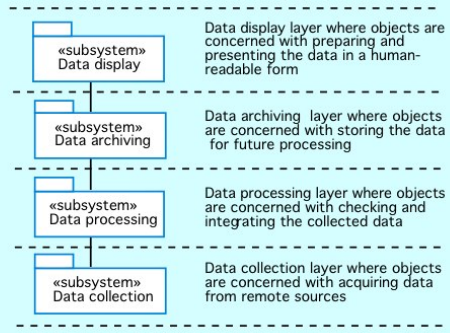


Compltete System Architecture

- Layered architecture: each step only depends on previous step

Compltete System Architecture

- Layered architecture: each step only depends on previous step



«subsystem» Data display

Data display layer where objects are concerned with preparing and presenting the data in a human-readable form

«subsystem» Data archiving

Data archiving layer where objects are concerned with storing the data for future processing

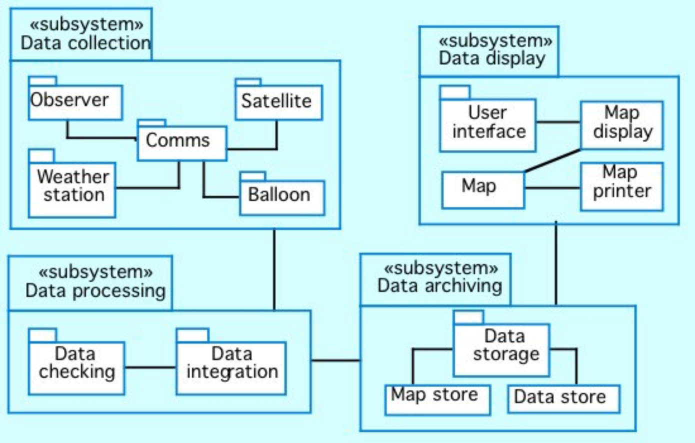
«subsystem» Data processing

Data processing layer where objects are concerned with checking and integrating the collected data

«subsystem» Data collection

Data collection layer where objects are concerned with acquiring data from remote sources

Subsystems in the Architecture



System Architecture of Weather Station

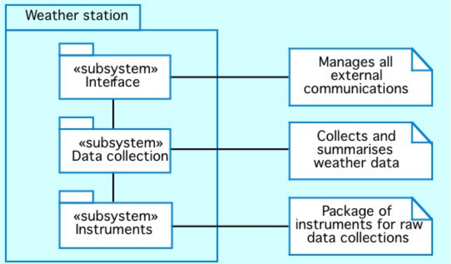
- Decomposing the system

System Architecture of Weather Station

- Decomposing the system
- Weather Station architecture:

System Architecture of Weather Station

- Decomposing the system
- Weather Station architecture:


Weather station
«subsystem» Interface
«subsystem» Data collection
«subsystem» Instruments
Manages all external communications
Collects and summarises weather data
Package of instruments for raw data collections

Object Identification

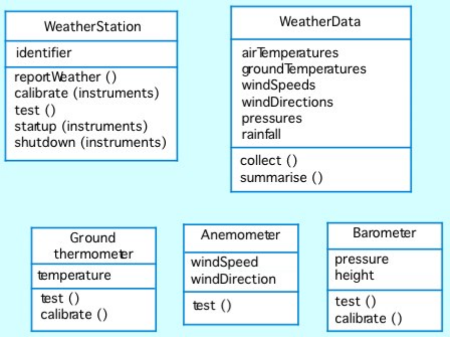
- Figure out what objects (classes) for each system/susbsystem

Object Identification

- Figure out what objects (classes) for each system/susbsystem
- Use application domain knowledge for attributes and services

Object Identification

- Figure out what objects (classes) for each system/susbsystem
- Use application domain knowledge for attributes and services
- For the weather station subsystem:



WeatherStation
identifier
reportWeather ()
calibrate (instruments)
test ()
startup (instruments)
shutdown (instruments)

WeatherData
airTemperatures
groundTemperatures
windSpeeds
windDirections
pressures
rainfall
collect ()
summarise ()

Ground thermometer
temperature
test ()
calibrate ()

Anemometer
windSpeed
windDirection
test ()

Barometer
pressure
height
test ()
calibrate ()

Design Models

- Graphical model of system to be implemented

Design Models

- Graphical model of system to be implemented
- Helps you to program your classes later
- Bridge between *requirements* and *implementation*

Design Models

- Graphical model of system to be implemented
- Helps you to program your classes later
- Bridge between *requirements* and *implementation*
  - can create conflicts for level of detail

Design Models

- Graphical model of system to be implemented
- Helps you to program your classes later
- Bridge between *requirements* and *implementation*
    - can create conflicts for level of detail
- *create several models with varying detail*

Design Models

- Graphical model of system to be implemented
- Helps you to program your classes later
- Bridge between *requirements* and *implementation*
    - can create conflicts for level of detail
- *create several models with varying detail*
- or *choose certain level of detail in single model*

Static and Dynamic models

- **Static Models**

Static and Dynamic models

- **Static Models**
  - describe system structure with classes and relationships

Static and Dynamic models

- **Static Models**
  - describe system structure with classes and relationships
  - relationships: *generalization*, *used/used-by*, *composition*

Static and Dynamic models

- **Static Models**
  - describe system structure with classes and relationships
  - relationships: *generalization*, *used/used-by*, *composition*
- **Dynamic Models**

Static and Dynamic models

- **Static Models**
  - describe system structure with classes and relationships
  - relationships: *generalization*, *used/used-by*, *composition*
- **Dynamic Models**
  - show interactions between system objects (not classes)

Static and Dynamic models

- **Static Models**
  - describe system structure with classes and relationships
  - relationships: *generalization*, *used/used-by*, *composition*
- **Dynamic Models**
  - show interactions between system objects (not classes)
  - *service requests*, *state chages*

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages
    - static models

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages
    - static models
- *Sequence models*: show sequence of object interactions

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
  - show as packages
  - static models
- *Sequence models*: show sequence of object interactions
  - UML sequence or collaboration diagram.

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages
    - static models
- *Sequence models*: show sequence of object interactions
    - UML sequence or collaboration diagram.
    - dynamic models

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages
    - static models
- *Sequence models*: show sequence of object interactions
    - UML sequence or collaboration diagram.
    - dynamic models
- *State machine models*: show state changes for individual objects

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages
    - static models
- *Sequence models*: show sequence of object interactions
    - UML sequence or collaboration diagram.
    - dynamic models
- *State machine models*: show state changes for individual objects
    - events and responses

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages
    - static models
- *Sequence models*: show sequence of object interactions
    - UML sequence or collaboration diagram.
    - dynamic models
- *State machine models*: show state changes for individual objects
    - events and responses
    - state chart diagrams

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages
    - static models
- *Sequence models*: show sequence of object interactions
    - UML sequence or collaboration diagram.
    - dynamic models
- *State machine models*: show state changes for individual objects
    - events and responses
    - state chart diagrams
    - dynamic models

UML for Modeling

- 12 types of graphical models to document the static and dynamic system design
- *Subsystem models*: show logical grouping of classes in sub-systems
    - show as packages
    - static models
- *Sequence models*: show sequence of object interactions
    - UML sequence or collaboration diagram.
    - dynamic models
- *State machine models*: show state changes for individual objects
    - events and responses
    - state chart diagrams
    - dynamic models
- Other model types in UML: *use case*, *object models*, *generalization*, etc.

Subsystem Models

- Objects in weather station package:

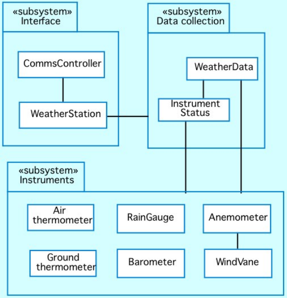


Figure: Weather station packages

Subsystem Models

- simple associations as well

Subsystem Models

- simple associations as well
- sub-system model + class model = describe logical grouping

Subsystem Models

- simple associations as well
- sub-system model + class model = describe logical grouping
- usually relate to Java packages/libraries

Sequence Diagrams

- Document interactions

Sequence Diagrams

- Document interactions
- For each interaction: sequence of object interactions

Sequence Diagrams

- Document interactions
- For each interaction: sequence of object interactions

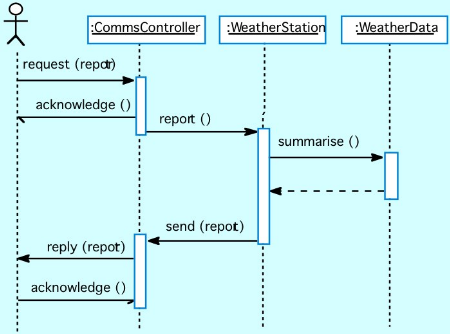


Figure: Sequence of operations in data collection

Sequence Diagrams

- Objects shown horizontally, time vertically

Sequence Diagrams

- Objects shown horizontally, time vertically
- Labeled arrows show atomic interactions between objects

Sequence Diagrams

- Objects shown horizontally, time vertically
- Labeled arrows show atomic interactions between objects
  - represent messages
  - *not* data flows

Sequence Diagrams

- Objects shown horizontally, time vertically
- Labeled arrows show atomic interactions between objects
  - represent messages
  - *not* data flows
- thin rectangles: when the object is the controlling object

Sequence Diagrams

- Objects shown horizontally, time vertically
- Labeled arrows show atomic interactions between objects
    - represent messages
    - *not* data flows
- thin rectangles: when the object is the controlling object
    - in a hierarchy, control is not relinquished until original object is replied to

Sequence Diagrams

- Objects shown horizontally, time vertically
- Labeled arrows show atomic interactions between objects
  - represent messages
  - *not* data flows
- thin rectangles: when the object is the controlling object
  - in a hierarchy, control is not relinquished until original object is replied to
- Sequence Diagrams also for single objects

State Diagrams

- For important objects: show their lifetimes and event responses

State Diagrams

- For important objects: show their lifetimes and event responses

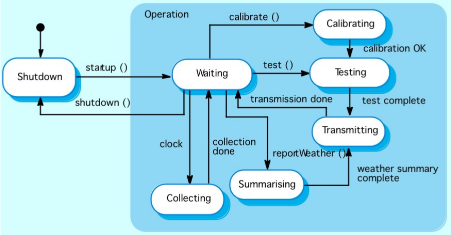


Figure: State diagram for weather station

State Diagrams

- For important objects: show their lifetimes and event responses

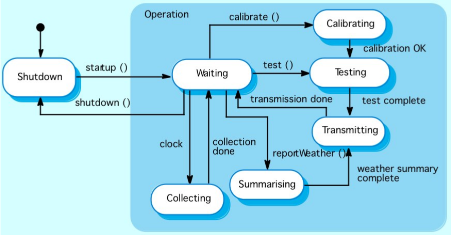


Figure: State diagram for weather station

- States can be helpful when implementing the class

Object-Interface Specification

- Only the interface

Object-Interface Specification

- Only the interface
    - not *implementation* or *data items*

Object-Interface Specification

- Only the interface
    - not *implementation* or *data items*
    - no internal details (private methos etc.)

Object-Interface Specification

- Only the interface
  - not *implementation* or *data items*
  - no internal details (private methos etc.)
- helps concurrent development

Object-Interface Specification

- Only the interface
    - not *implementation* or *data items*
    - no internal details (private methos etc.)
- helps concurrent development
- more maintainable than full specification of classes

Object-Interface Specification

- Only the interface
  - not *implementation* or *data items*
  - no internal details (private methos etc.)
- helps concurrent development
- more maintainable than full specification of classes
- Two approaches:

Object-Interface Specification

- Only the interface
  - not *implementation* or *data items*
  - no internal details (private methos etc.)
- helps concurrent development
- more maintainable than full specification of classes
- Two approaches:
  - each interface is a class in Java

Object-Interface Specification

- Only the interface
  - not *implementation* or *data items*
  - no internal details (private methos etc.)
- helps concurrent development
- more maintainable than full specification of classes
- Two approaches:
  - each interface is a class in Java
  - declare interfaces separately from classes

Object-Interface Specification

- Only the interface
  - not *implementation* or *data items*
  - no internal details (private methos etc.)
- helps concurrent development
- more maintainable than full specification of classes
- Two approaches:
  - each interface is a class in Java
  - declare interfaces separately from classes
    - use `interface` and let classes implement an interface
- Simply use Java (or another OO PL) to define interfaces

Design Evolution

- OO approach allows for easy changes to design

Design Evolution

- OO approach allows for easy changes to design
    - state representation does not affect interface

Design Evolution

- OO approach allows for easy changes to design
  - state representation does not affect interface
  - objects are loosely coupled ⇒ easy to introduce new ones

Design Evolution

- OO approach allows for easy changes to design
    - state representation does not affect interface
    - objects are loosely coupled ⇒ easy to introduce new ones
- E.g.: add pollution-monitoring to weather station

Design Evolution

- OO approach allows for easy changes to design
  - state representation does not affect interface
  - objects are loosely coupled ⇒ easy to introduce new ones
- E.g.: add pollution-monitoring to weather station
  - introduce `Air quality` class at `Weather Data` level

Design Evolution

- OO approach allows for easy changes to design
  - state representation does not affect interface
  - objects are loosely coupled ⇒ easy to introduce new ones
- E.g.: add pollution-monitoring to weather station
  - introduce `Air quality` class at `Weather Data` level
  - add `reportAirQuality()` operation to `WeatherStation`

Design Evolution

- OO approach allows for easy changes to design
  - state representation does not affect interface
  - objects are loosely coupled ⇒ easy to introduce new ones
- E.g.: add pollution-monitoring to weather station
  - introduce `Air quality` class at `Weather Data` level
  - add `reportAirQuality()` operation to `WeatherStation`
  - add classes for pollution monitoring

Outline

1. Introduction
2. Architectural Design
   - System Organization
   - Distributed Systems Architecture
3. Application Architectures
   - Types of AAs
4. Object-Oriented Design
   - Object-Oriented Design Process
5. Interface Design

Introduction

- “Design interface with user’s experience and interaction in mind”

Introduction

- “Design interface with user’s experience and interaction in mind”
- Field of **Human Computer Interaction**

Introduction

- “Design interface with user’s experience and interaction in mind”
- Field of **Human Computer Interaction**
- Examples:

Introduction

- "Design interface with user's experience and interaction in mind"
- Field of **Human Computer Interaction**
- Examples:
  - Command line interface
  - XHTML/CSS
  - Swing
  - sometimes combined with hardware design

Introduction

- “Design interface with user’s experience and interaction in mind”
- Field of **Human Computer Interaction**
- Examples:
  - Command line interface
  - XHTML/CSS
  - Swing
  - sometimes combined with hardware design
- Software developer == Interface designer

Introduction

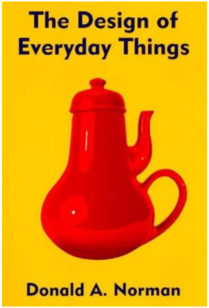
- Good user experience is crucial to product success

Introduction

- Good user experience is crucial to product success
- User errors are often a result of bad interface design

Introduction

- Good user experience is crucial to product success
- User errors are often a result of bad interface design



Introduction

- Bad interface leads to
  - frustration

Introduction

- Bad interface leads to
  - frustration
  - inaccessible / difficult to use features

Introduction

- Bad interface leads to
  - frustration
  - inaccessible / difficult to use features
  - mistakes

Introduction

- Bad interface leads to
  - frustration
  - inaccessible / difficult to use features
  - mistakes
- Some (random) guides:

Introduction

- Bad interface leads to
    - frustration
    - inaccessible / difficult to use features
    - mistakes
- Some (random) guides:
    - make user interaction simple and efficient (Quicksilver, TaDa List, Ubiquity, !gpsGuide, !Password-engine)

Introduction

- Bad interface leads to
  - frustration
  - inaccessible / difficult to use features
  - mistakes
- Some (random) guides:
  - make user interaction simple and efficient (Quicksilver, TaDa List, Ubiquity, !gpsGuide, !Password-engine)
  - balance meaning with action (icons in any app, Reason, !Office 2010)

Introduction

- Bad interface leads to
    - frustration
    - inaccessible / difficult to use features
    - mistakes
- Some (random) guides:
    - make user interaction simple and efficient (Quicksilver, TaDa List, Ubiquity, !gpsGuide, !Password-engine)
    - balance meaning with action (icons in any app, Reason, !Office 2010)
    - understand and aid human memory (!16tone, !SonicMood)

Introduction

- Bad interface leads to
  - frustration
  - inaccessible / difficult to use features
  - mistakes
- Some (random) guides:
  - make user interaction simple and efficient (Quicksilver, TaDa List, Ubiquity, !gpsGuide, !Password-engine)
  - balance meaning with action (icons in any app, Reason, !Office 2010)
  - understand and aid human memory (!16tone, !SonicMood)
  - don't terrorize user with errors (http://adobegripes.tumblr.com)

Introduction

- Bad interface leads to
  - frustration
  - inaccessible / difficult to use features
  - mistakes
- Some (random) guides:
  - make user interaction simple and efficient (Quicksilver, TaDa List, Ubiquity, !gpsGuide, !Password-engine)
  - balance meaning with action (icons in any app, Reason, !Office 2010)
  - understand and aid human memory (!16tone, !SonicMood)
  - don't terrorize user with errors (http://adobegripes.tumblr.com)
  - work with user's capabilities (accessibility, Frenzic, Mac toolbar, !Any (old) Linux GUI interface)

Formal principles

- User familiarity

Formal principles

- User familiarity
- Consistency

Formal principles

- User familiarity
- Consistency
- Minimal surprise

Formal principles

- User familiarity
- Consistency
- Minimal surprise
- Recoverability

Formal principles

- User familiarity
- Consistency
- Minimal surprise
- Recoverability
- User guidance

Formal principles

- User familiarity
- Consistency
- Minimal surprise
- Recoverability
- User guidance
- User diversity

Issues of Interface Design

Answer two questions:

- How should user interact with system?
- How should information be presented to user?

User Interaction

- Advanced from interfaces designed for experts

User Interaction

- Advanced from interfaces designed for experts
- Current interaction styles:

User Interaction

- Advanced from interfaces designed for experts
- Current interaction styles:
  - *direct manipulation*

User Interaction

- Advanced from interfaces designed for experts
- Current interaction styles:
  - *direct manipulation*
  - *menu selection*

User Interaction

- Advanced from interfaces designed for experts
- Current interaction styles:
  - *direct manipulation*
  - *menu selection*
  - *form fill-in*

User Interaction

- Advanced from interfaces designed for experts
- Current interaction styles:
  - *direct manipulation*
  - *menu selection*
  - *form fill-in*
  - *command language*

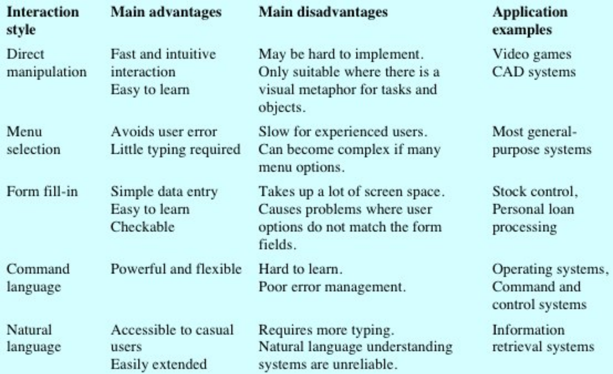
User Interaction

- Advanced from interfaces designed for experts
- Current interaction styles:
  - *direct manipulation*
  - *menu selection*
  - *form fill-in*
  - *command language*
  - *natural language*

User Interaction

- Advanced from interfaces designed for experts
- Current interaction styles:
  - *direct manipulation*
  - *menu selection*
  - *form fill-in*
  - *command language*
  - *natural language*

Comaprisons of Interface Styles



Interaction style	Main advantages	Main disadvantages	Application examples
Direct manipulation	Fast and intuitive interaction Easy to learn	May be hard to implement. Only suitable where there is a visual metaphor for tasks and objects.	Video games CAD systems
Menu selection	Avoids user error Little typing required	Slow for experienced users. Can become complex if many menu options.	Most general-purpose systems
Form fill-in	Simple data entry Easy to learn Checkable	Takes up a lot of screen space. Causes problems where user options do not match the form fields.	Stock control, Personal loan processing
Command language	Powerful and flexible	Hard to learn. Poor error management.	Operating systems, Command and control systems
Natural language	Accessible to casual users Easily extended	Requires more typing. Natural language understanding systems are unreliable.	Information retrieval systems

Figure: Interaction Styles Merits/Demerits

Interaction Issues

- single application may have *mixed styles*

Interaction Issues

- single application may have *mixed styles*
  - e.g., Linux: direct manipulation, menu selection, command language

Interaction Issues

- single application may have *mixed styles*
  - e.g., Linux: direct manipulation, menu selection, command language
- Web interfaces

Interaction Issues

- single application may have *mixed styles*
    - e.g., Linux: direct manipulation, menu selection, command language
- Web interfaces
    - mostly *forms based*

Interaction Issues

- single application may have *mixed styles*
  - e.g., Linux: direct manipulation, menu selection, command language
- Web interfaces
  - mostly *forms based*
  - *direct manipulation* ?

Interaction Issues

- single application may have *mixed styles*
  - e.g., Linux: direct manipulation, menu selection, command language
- Web interfaces
  - mostly *forms based*
  - *direct manipulation* ?
  - *command language* ?

Information Presentation

- Use the MVC approach for focusing on UI

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
    - precise vs relationships (tables/text/graphs, sparklines)

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
  - precise vs relationships (tables/text/graphs, sparklines)
  - speed of change in values (tables/graphs, animation)

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
  - precise vs relationships (tables/text/graphs, sparklines)
  - speed of change in values (tables/graphs, animation)
- Visualizations for large, changing, interactive data

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
  - precise vs relationships (tables/text/graphs, sparklines)
  - speed of change in values (tables/graphs, animation)
- Visualizations for large, changing, interactive data
  - molecule models, network graphs

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
  - precise vs relationships (tables/text/graphs, sparklines)
  - speed of change in values (tables/graphs, animation)
- Visualizations for large, changing, interactive data
  - molecule models, network graphs
- Color usage:

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
    - precise vs relationships (tables/text/graphs, sparklines)
    - speed of change in values (tables/graphs, animation)
- Visualizations for large, changing, interactive data
    - molecule models, network graphs
- Color usage:
    - few (4-5)

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
  - precise vs relationships (tables/text/graphs, sparklines)
  - speed of change in values (tables/graphs, animation)
- Visualizations for large, changing, interactive data
  - molecule models, network graphs
- Color usage:
  - few (4-5)
  - system change == color change

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
  - precise vs relationships (tables/text/graphs, sparklines)
  - speed of change in values (tables/graphs, animation)
- Visualizations for large, changing, interactive data
  - molecule models, network graphs
- Color usage:
  - few (4-5)
  - system change == color change
  - display anomalies, similarities

Information Presentation

- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
  - precise vs relationships (tables/text/graphs, sparklines)
  - speed of change in values (tables/graphs, animation)
- Visualizations for large, changing, interactive data
  - molecule models, network graphs
- Color usage:
  - few (4-5)
  - system change == color change
  - display anomalies, similarities
  - consistency

Information Presentation

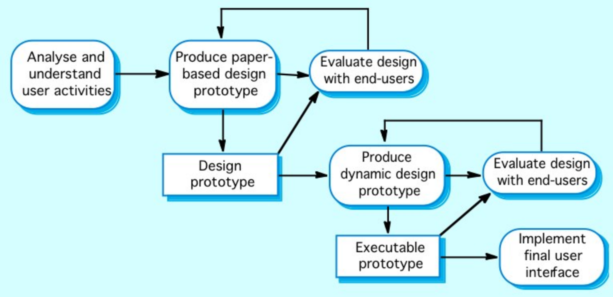
- Use the MVC approach for focusing on UI
- Kinds of presentation elements depend on many factors:
  - precise vs relationships (tables/text/graphs, sparklines)
  - speed of change in values (tables/graphs, animation)
- Visualizations for large, changing, interactive data
  - molecule models, network graphs
- Color usage:
  - few (4-5)
  - system change == color change
  - display anomalies, similarities
  - consistency
  - eye strain (red-on-blue)

UI Design Process

- 3-step process

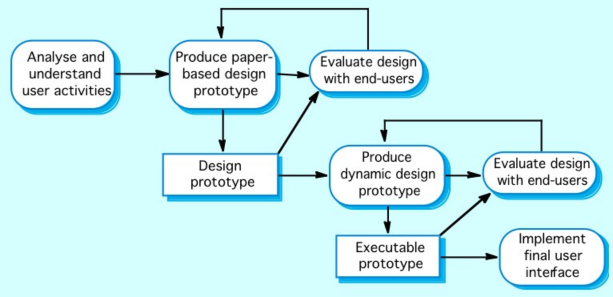
UI Design Process

- 3-step process


Analyse and understand user activities
Produce paper-based design prototype
Evaluate design with end-users
Design prototype
Produce dynamic design prototype
Evaluate design with end-users
Executable prototype
Implement final user interface

UI Design Process

- 3-step process


Analyse and understand user activities
Produce paper-based design prototype
Evaluate design with end-users
Design prototype
Produce dynamic design prototype
Evaluate design with end-users
Executable prototype
Implement final user interface

- User analysis, System prototyping, Interface evaluation