Outline

- Dimensions of design
- Object-Oriented Model
- Subsystems
- System Architecture
- System Control
- Progression of design

Design

- A design captures understanding of how system should be constructed
- Process of design explores alternatives and develops design of system
- Goal is to have enough detail thought out to avoid problems when implementing – want to be able to give subsystems to others and have result work together.

Dimensions of Design

- Can view design in multiple ways
  - Structural
  - Behavioral
  - Information
- Use different representations for procedural and object-oriented programming

Object-Oriented Dimensions

- Information – classes and implementing data structures
- Behavioral – descriptions of scenarios in system
- Structural – next slide…

O-O Design Structures

- Object Model – classes and relationships
- Subsystems – division of system
- System Architecture – how subsystems compose together
Concepts

• Conception – idea or understanding of an individual person about the world
• Concept – common idea or understanding
  • Concept may be
    – concrete: dog, desk, factory, etc.
    – intangible: time, quality, etc.
    – relational: ownership, partnership, marriage, etc.

Object-Oriented Model

• Goal of analysis is to capture concepts involved in system
• Use object-oriented structures:
  – class
  – composition (association and aggregation)
  – polymorphism (generalization/specialization)

Subsystems

• A major component of a system
• Not necessarily a single object, although sometimes single class may provide interface
• Identified with a domain
• Domain – a “world” of related objects and operational rules.

Example Domains

• User-Interface domain – windows, pulldown menus, dialog boxes, icons along with behavior
• University registration domain – objects related to students, courses, instructors, classrooms, class resources, etc.
• Payroll domain – objects related to employees, pay scales, payment records, etc.

Properties of Domains

1. An object is defined in one domain.
2. The existences of an object in a domain may require the existence of other objects in the domain.
3. The objects in one domain do not require the existence of objects in a different domain.

Example Domain

• Airline management system
  – Air route in Airline Management domain
  – Air route requires airplanes and airports
  – Air routes and airplanes can exist without windows, icons or dialog boxes.
Objects in Domains

• May be useful to have correspondence between objects in different domains
• Classical examples are between application domain objects and user-interface objects
• For instance:
  An airplane object may correspond to an icon in the user-interface.

Classification of Domains

• Application domain – environment in which system is used
• Service domain – objects and mechanisms used to support the application domain
• Architectural domain – generic mechanisms for system control and data management
• Implementation domain – programming language, operating system, networks, class libraries

Domain Bridge

• Relationship between domains, where one uses objects/services of the other
• Client/Server – client domain uses services of server domain
  Ex. Airline Management domain uses User Interface domain
• Peer-to-peer – both domains use each other

Domains and Subsystems

• Subsystem corresponds directly to domain
• Idea is that reuse of domains is better than reuse of individual classes
  – If we implement a set of domain classes, they can be used in later applications
  – An infrastructure for building future systems
• Can use domain classes implemented by others

Organizing Subsystems

• Horizontal (layered) structure
  – Higher layer built on top of lower layer
  – Uses services of lower layer objects
  – May be correspondence between objects in layers
• Vertical structure
  – Division into weakly coupled subsystems
  – Each provides unique service
• May be mixed

Ex. Internet Subsystems

Note: general network code knows nothing about application
Ex. X Windows Application

- Application
- Widget Set
- Xt Intrinsics
- Xlib
- X Protocol
- X Server
- Device Drivers

System Architectures

- Specify structure and control of subsystems
- Similar idea to design patterns
- Common structures used in design

Kinds of Architectures

- **Batch** – get data, run, dump results
- **Continuous** – get data throughout execution
- **Interactive** – interactions dominate
- **Transaction** – store and update data
- **Rule-based** – use rules to determine what to do
- **Simulation** – simulate evolving, real-world objects
- **Real-time** – run within time constraints

System Control

- **External** – control of events and interactions among objects
  - Procedure-driven
  - Event-driven
  - Concurrent
- **Internal** – flow of control within methods
  - Procedure calls
  - Intertask calls (concurrency)

Event-Driven Control

- Dispatcher provided by OS or language runtime system
  - When event occurs, dispatcher calls functions attached to event

Progression of Design

- Object model – classes, composition, polymorphism
- System design – subsystems, architecture/control, data storage
- Detailed design – architecture and implementation classes, representations of classes, optimizations
System Design

- Concerned with global properties of system
- Identifying subsystems, architecture, control
- Use of concurrency and multiple processors
- Startup, shutdown, and failure actions

Detailed Design

- Add classes to object model for implementation and architecture domains
- Design complex algorithms
- Determine implementations of relationships and representations of classes
- Optimizations

Optimizations

- Typical optimizations are to
  - decrease execution time
  - reduce memory usage
  - minimize disk I/O access
- Cannot achieve all at once
- Ex. Trade-off between memory usage and execution time (subject of CS 2604)

About Optimization

- There is a trade-off between software engineering goals and efficiency
- For example,
  - encapsulation makes a design more flexible, but possibly less efficient; and
  - not using encapsulation makes the design less flexible, but possibly more efficient

Design Overview

- Think about structure of system with respect to classes and relationships identified
- Postpone committing to details of class implementation until certain of structure
- Think through scenarios to “test” design(s)
- Easier to change design than change code

Remember

- You likely have not seen the most challenging program you will see in college
- Design is critical to
  - Saving time – more time on design can avoid debugging problems
  - Getting programs done
- Design is not just something a perverse professor made up to torture you