Fundamentals of Design

Deriving a solution which satisfies the software requirements

Hamlet Chapter 11
Additional Reading

Objectives

- To define design
- To introduce the design process
- To preview two design strategies
  - Functional decomposition
  - Object Oriented design
- To give a short overview of software architecture

What is Design?

- Design
  - The creative process of transforming a problem into a solution
  - In our case, transforming a requirements specification into a detailed description of the software

- Design
  - The description of the solution
  - In our case, we will develop a software design

General Design Process

Stages of Design

- Problem understanding
  - Look at the problem from different angles to discover the design requirements
- Identify one or more solutions
  - Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources
- Describe solution abstractions
  - Use graphical, formal or other descriptive notations to describe the components of the design
- Repeat process for each identified abstraction until the design is expressed in primitive terms

From Informal to Formal Design
**Phases in the Design Process**

- **Requirements specification**
- **Architectural design**
- **Abstract specification**
- **Interface design**
- **Component design**
- **Data design**
- **Algorithm design**

**Design Activities**

- System architecture
- Software specification
- Interface specification
- Component specification
- Data specification
- Algorithm specification

**Design Products**

- Architectural design
- Abstract specification
- Interface design
- Component design
- Data design
- Algorithm design

**Design Phases**

- **Architectural design**
  - Identify sub-systems
- **Abstract specification**
  - Specify sub-systems
- **Interface design**
  - Describe sub-system interfaces
- **Component design**
  - Decompose sub-systems into components
- **Data structure design**
  - Design data structures to hold problem data
- **Algorithm design**
  - Design algorithms for problem functions

**Design Strategies**

- **Functional design**
  - The system is designed from a functional viewpoint
  - The system state is centralized and shared between the functions operating on that state
- **Object-oriented design**
  - The system is viewed as a collection of interacting objects
  - The system state is decentralized and each object manages its own state
  - Objects may be instances of an object class and communicate by exchanging messages

**Functional View of a Compiler**

- **Source Program**
  - Scan
  - Build Symbol Table
- **Token Stream**
  - Analyze Syntax Tree
  - Generate Code
- **Symbol Table**
  - Check Symbols
  - Error Indicator
- **Grammar**
  - Print Error Message

**Object-Oriented View of a Compiler**

- **Source Program**
  - Scan
  - Add Symbol Table
- **Token Stream**
  - Check Symbols
  - Get
- **Grammar**
  - Build Abstract Code
  - Generate Object Code

**Key Points**

- Design is a creative process
- Design activities include architectural design, system specification, component design, data structure design and algorithm design
- Functional decomposition considers the system as a set of functional units
- Object-oriented decomposition considers the system as a set of objects
Architectural Design

The High-Level Structure of a Software Intensive System

Architectural Parallels

- Architects are the technical interface between the customer and the contractor building the system
- A bad architectural design for a building cannot be rescued by good construction
  - The same is true for software
- There are specialist types of building and software architects
- There are schools or styles of building and software architecture

Architectural Design Process

- System structuring
  - The system is decomposed into several principal sub-systems and communications between these sub-systems are identified
- Control modeling
  - A model of the control relationships between the different parts of the system is established
- Modular decomposition
  - The identified sub-systems are decomposed into modules

Architectural Models

- Structure, control and modular decomposition may be based on a particular model or architectural style
- However, most systems are heterogeneous in that different parts of the system are based on different models
- The architectural model used affects the performance, robustness, distributability and maintainability of the system

System Structuring

- Concerned with decomposing the system into interacting sub-systems
- The architectural design is normally expressed as a block diagram presenting an overview of the system structure
- More specific models showing how sub-systems share data, are distributed, and interface with each other may also be developed

Packing Robot Control System

Vision System

Object Identification System

Packaging System

Arm Controller

Gripper Controller

Conveyor Controller
The Repository Model

- Sub-systems must exchange data
- This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems.
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.

Repository Characteristics Model

- Advantages
  - Efficient way to share large amounts of data.
  - Sub-systems need not be concerned with how data is produced.
  - Centralized management e.g., backup, security, etc.
  - Sharing model is published as the repository schema.
- Disadvantages
  - Sub-systems must agree on a repository data model.
  - Inevitably a compromise.
  - Data evolution is difficult and expensive.
  - No scope for specific management policies.
  - Difficult to distribute efficiently.

Client-Server Architecture

- Distributed system model which shows how data and processing is distributed across a range of components.
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

Client-Server Characteristics

- Advantages
  - Distribution of data is straightforward.
  - Makes effective use of networked systems.
  - May require cheaper hardware.
  - Easy to add new servers or upgrade existing servers.
- Disadvantages
  - No shared data model so sub-systems use different data organization.
  - Data interchange may be inefficient.
  - Redundant management in each server.
  - No central register of names and services.
  - It may be hard to find out what servers and services are available.
Abstract Machine Model

- Used to model the interfacing of sub-systems
- Organizes the system into a set of layers (or abstract machines) each of which provide a set of services
- Supports the incremental development of sub-systems in different layers
  - When a layer interface changes, only the adjacent layer is affected
- However, often difficult to structure systems in this way

Control Models

- Are concerned with the control flow between sub-systems
  - Distinct from the system decomposition model
- Centralized control
  - One sub-system has overall responsibility for control and starts and stops other sub-systems
- Event-based control
  - Each sub-system can respond to externally generated events from other sub-systems or the system’s environment

Call-Return Model

- A control sub-system takes responsibility for managing the execution of other sub-systems
- Call-return model
  - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards
  - Applicable to sequential systems
- Manager model
  - Applicable to concurrent systems
  - One system component controls the stopping, starting and coordination of other system processes

Version Management System

- Version Manager
- Object Manager
- Database System
- Operating System
- Hardware

Real-Time System Control

- System Controller
- Sensor Processes
- Actuator Processes
- Control Processes
- User Interface
- Fault Handler
Event-Driven Systems

- Driven by externally generated events where the timing of the event is out with the control of the sub-systems which process the event
- Two principal event-driven models
  - Broadcast models
    - An event is broadcast to all sub-systems
    - Any sub-system which can handle the event may do so
  - Interrupt-driven models
    - Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing
- Other event driven models include, for example, spreadsheets

Broadcast Model

- Effective in integrating sub-systems on different computers in a network
- Sub-systems register an interest in specific events
  - When these occur, control is transferred to the sub-system which can handle the event
- Control policy is not embedded in the event and message handler
  - Sub-systems decide on events of interest to them
- However, sub-systems don’t know if or when an event will be handled

Selective Broadcasting

Subsystem 1

Event and Message Handler

Interrupt-Driven Systems

- Used in real-time systems where fast response to an event is essential
- There are known interrupt types with a handler defined for each type
- Each type is associated with a memory location and a hardware switch causes transfer to its handler
- Allows fast response but complex to program and difficult to validate

Interrupt-Driven Control

Interrupt Vector

Handler 1
Handler 2
Handler 3
Handler 4
Process 1
Process 2
Process 3
Process 4

Domain-Specific Architectures

- Architectural models which are specific to some application domain
- Two types of domain-specific model
  - Generic models which are abstractions from a number of real systems and which encapsulate the principal characteristics of these systems
  - Reference models which are more abstract, idealized model
    - Provide a means of information about that class of system and of comparing different architectures
**Generic Models**

- Compiler model is a well-known example although other models exist in more specialized application domains
  - Lexical analyzer
  - Symbol table
  - Syntax analyzer
  - Syntax tree
  - Semantic analyzer
  - Code generator
- Generic compiler model may be organized according to different architectural models

**Compiler Model**

![Diagram of compiler model]

**Language Processing System**

![Diagram of language processing system]

**Modular Decomposition**

- Another structural level where sub-systems are decomposed into modules
- Two modular decomposition models covered in this class
  - An object model where the system is decomposed into interacting objects
  - A data-flow model where the system is decomposed into functional modules which transform inputs to outputs

**Key Points**

- The software architect is responsible for deriving a structural system model, a control model and a sub-system decomposition model
- Large systems rarely conform to a single architectural model
- System decomposition models include repository models, client-server models and abstract machine models
- Control models include centralized control and event-driven models

**Key Points**

- Modular decomposition models include data-flow and object models
- Domain specific architectural models are abstractions over an application domain
  - They may be constructed by abstracting from existing systems or may be idealized reference models