Architecture

- How do you begin to architect a solution for a problem like this?
- Break it into modules!
Properties of Modularity

- **Functionality**: chosen as distinct and natural functional groupings, e.g., executive, legislative, etc.

- **Hierarchy**: itself be a system, internally decomposed into modules, not visible externally, e.g., agencies, departments

- **Separation of concerns**: strongly associated within each module, weakly to other modules, e.g., law, legislative

- **Interoperability**: successfully interact to realize higher level purposes of the system, e.g., law enforcement, judiciary

- **Reusability**: defined, documented, implemented independently so their designs can be used by other systems, e.g., US government, adopted by other countries
Granularity tradeoff.

- How big should we make the modules
  - Many simple small ones (fine: divide and conquer)
  - Or a few complicated big ones... (easy to understand)

- This aspect of modularity is called **granularity**.

- Which is better? Typically “decomposition within modules” allows the system to view different levels of granularity.
Simple modularity in a computer system

- Processor
- Arithmetic unit
- Address unit
- Memory
- Storage
- Network adaptor
- Network
Simple modularity in a computer system

Processor

Arithmetic unit

Address unit

Processor

interface

Bus

Memory

Storage

Network adaptor

Network
Interface

- An external view of a module, specifying what functions it performs and instructing other modules to invoke those functions.
- Determined at time of architecture was designed.
- Could be with different versions of a module share a common interface or module changes over time.
  - leads to “separation of concerns”, e.g., equipment module: printers (not specify how printer does its job).
Interface

- Specified at time of the architecture is designed \textit{w/o assumption} of how the module will be implemented.
- Serve as a \textit{starting point} when the implementation of a module is turnovered to other people
- Hardware interface: power outlets, USB
- Software interface: boundary between two software programs, i.e., think each software is a server, “interface” provides way that two servers can communicate
Interfaces: three properties of data passed through an interface

Name, Type, and Value

PARAMETERS

N numbers of Float type

INTERFACE

Computation of key statistics

RETURNS

Compute Mean and Variance

2 Numbers of float type that signify: Mean, Variance
Class Announcements


- Business paper draft due in 1 week!
  - Due Thursday 3/3

- Database Assignment 2 posted
  - Due Today 2/25

- Midterm return
Data: Type, Name and Value

- **Name**: meaningful to users, e.g., “dailyrain”, “income”

- **Type**: structure & interpretations, passing an interface is often specified in terms of a limited number of standard data types,
  - e.g., integer, float \((m,n)\), characters, and string (range of values and allowable manipulation)

- **Value**: changes over time due to manipulations
Example data types

**Integer**
- “natural number between, limited in size by the number of bits”
- Could be represented (in many ways) by 16 bits, e.g., since $2^n = 65,536$ (n=16)
  - E.g., The number of letters in the Roman alphabet is 26.

**Float**
- Real number approximated by a finite number of digits in scientific notation.
- In form of $m \times 10^n$, where size of n and m is limited by the number of bits
More data types

Character
- “values assuming a-z and A-Z plus space and punctuation marks”
  - could be represented by 8 bits (ASCII) or Unicode (16 bits)

Character string
- “collection of $n$ characters, where $n$ is customizable”
Compound data types

Programmer-defined composition of basic data types

Example:

```java
Employee {
    String name;
    String address;
    Integer year_of_birth;
    etc.
}
```
**Actions & Protocols Parameters and Return**

- **Action**: something specifics that a module does

- **Protocol**: A sequence of “actions” leads to an protocol
  - Typically a menu that tells clients what it prepares to do, e.g., withdraw, deposit, balance check, transfer or $+, -, \times, \div$.
  - A pull-down or display menu
  - protocol “+”: 1) enter: “-1” & “-2” → display  2) add “-1+ (-2)” → display (-3)
  - “-1”, “-2” are **parameters** and “-3” is **return**.

- **Message**: send a packet of data w/o return
In the simplest case, the infrastructure deals with a package of data (non-standard terminology)
- collection of bits
- specified number and ordering

The objective of the infrastructure is to store and communicate packages while maintaining data integrity

File for storage, message for communication
Client-Server Interactions

1) Decides it needs to invoke an action of a server module
2) Invokes the action by name
3) Passes data to server
4) Processes parameters in accordance with the specified action; generates return values
5) Passes the return values to the client
6) Processes the return values to complete the interaction

Client Module

Server Module
Human Interfaces

- Interface applies to interaction between a user and a computer application

- Graphic user interface (GUI): use comfortable elements to human
  - E.g., graphics, images, animation, sound, pointing devices, keyboards
  - Pull down menus
Interface Example: Automatic teller machine (ATM)

What is the interface between this machine and the customer?
Steps

Define available actions
  e.g., Authentication, specify_account, amount, cash_withdrawal, and etc.

Define, for each higher level function, a protocol
  - Single action or a finite sequence of actions
Interface building blocks

Message on screen or printed
- Menu of actions or returns from an action
- Touch selection of action

Keypad
- Input parameters to an action

Card reader
- Authentication, input parameters

Money output slot
- Returns money
Action: authentication

Parameters
Internal functionality

Returns
Action: authentication

Parameters
- Identity (card in slot)
- Institution (card in slot)
- PIN (typed on keypad)

Internally, it contacts institution and matches against its database, institution noted for all subsequent actions (example of state)

Returns
- Screen message (“Invalid PIN” or menu of available actions)
Action: specify_account

Parameters
Internal functionality
Returns
Action: specify_account

Parameters

- Account (touch screen from menu of choices)

Internally, choice noted for all subsequent actions (another example of state)

Returns

- None
Action: amount

Parameters
- Dollars_and_cents (typed on keypad)

Internally, amount noted (another example of state)

Returns
- Success or failure (state dependent, for example for a withdraw failure when dollars_and_cents exceeds balance)
Protocol: cash_withdrawal

What is the sequence of actions?
Protocol: cash_withdrawal

- authentication → failure
- choose objective → other objectives
- account → no accounts
- amount → balance exceeded!
Well-principled architecture

Goal:
- Minimize cost & maximize performance
- Minimize effort to maintain & develop new applications
- Provide capability to support operations and reliability

Rely on layering principle:
- To contain complexity
- To coordinate suppliers of modules
- To allow new capabilities to be added incrementally
Layering builds capability incrementally by adding to what exists.
Layering builds capability incrementally by adding to what exists.

Layering builds capability incrementally by adding to what exists.

Layering builds capability incrementally by adding to what exists.

Layering builds capability incrementally by adding to what exists.

Layering builds capability incrementally by adding to what exists.

Layering builds capability incrementally by adding to what exists.

Layering builds capability incrementally by adding to what exists.

Layering builds capability incrementally by adding to what exists.

Layering builds capability incrementally by adding to what exists.
Interaction of layers

“Here is the actions I can perform to support you, but don’t ask how I do them”

Layer above is a client of the layer below

Each layer provides services to the layer above....w/ revealing it is done

....by utilizing the services of the layer below and adding capability

Layer below as a server to the layer above
Three types of software

Application

• Components and frameworks:
  What is in common among applications

• Infrastructure:
  Basic services (communication, storage, concurrency, presentation, etc.)
Major layers

Network Computing Infrastructure

Applications

Application components

Middleware

Operating system

Network

Support interaction at different locations

Professional services, e.g., accounting, law

Resource management, e.g., janitorial, gardening

Auto manufacture and devices process of assembly lines

All auto companies components, e.g., batteries

Analogy

Those are specialized services not typically provided within the company
Summary of Major layers Principles

1) Each layer is a server to the layer above, providing a set of actions but not revealing how they are implemented, e.g., a new networking technology can be introduced without modifying middleware or application.

2) Each layer is a client to the layer below, utilizing its available actions in the course of providing services to the layer above.

3) Each layer is permitted to interact with the layers immediately above and below.

Functionally, provide increasingly elaborated or specialized services at each higher layer, based on services provided by lower layers!!
Infrastructure needs to “treat” each host “identically” in a sense that it needs to worry less about “types” or “heterogeneity” of applications (e.g., windows words, Mac words or other text compatible editors).

This is done by having “middleware” that shields application from the rest of the infrastructure!
Layers on Data and Information

- Applications
- Middleware
- Operating system
- Network

Additional interpretation

Additional structure, e.g. files (storage) or message (communications)

Packets of bits, with 1) known order 2) unknown structure & interpretation

No interpretation attached until application → Separation of concerns
Example 1

Bob sends a letter to Alice

US Postal Service

Shipping Container

UK Royal Mail

Shipping Container

ABC Airlines
Example 2

Web server ➔ Web page ➔ Web browser

Application

Operating system

File (retrieve by browser)

File system

Message

Network

Fragmentation ➔ Collection of packets ➔ Assembly

Application

Operating system

Network