Chapter 6:
Principles of Architecture Design

Modularity and Layering

Book By Musacchio

Lecture 12
TIM 50  Autumn 2012

Tuesday  November 5, 2012
Announcement

1. Assignments submitted later than deadline will have 50% points at most.
2. HW 3 and DB assignment 2 will be released on Tuesday (11/6/2012).

HW 3 is due one week later (11/13/2012), and

DB assignment is due two weeks later (11/20/2012).

4. The grades for every assignment will be given in eCommons.
5. It's important to check webpage to get the latest information and assignments changes.
   Submit your business proposal through e-Common

Check class webpage Frequently!
Review

Information Technology

Business Information Systems

Computer System Architecture, Functions

Network Computing Architectures

Data Processing through Network Systems

H/W
S/W
Binary System
Data Formats
Boolean Logic, Gate
Machine Language
Hi-level Language
Operating System
Window, Mac, Linux

Peer to peer
Same Potential/Fn
Client Server
Functional Asymmetry
ATM, Internet terminal
M-phone
Client, Server, Data Server
Middle Ware

ALU
Memory
RAM, ROM
Mag/Flash/USB
I/O
Data Bus
Clock Pulse, Speed

Data representation
Data Copy
Data Transfer
Data communications
Protocols

Data representation
Data Copy
Data Transfer
Data communications
Protocols
Contents are re-edited using materials from Internet Sites
Modularity and Layering

Business IT System

Computer H/W, S/W
Communication H/W, S/W

Architecture, Structures

Sub-Systems
Components
Modules

Hierarchy

Couplings
Control

System H/W, S/W
Functionality
Efficiency
Flexibility
Interoperability
Reviews for Computer

What is the Computer Systems?
Data Processing System
Data used in the Computer
Binary(Number) System; 0, 1,
what about base 10, base 60 or Base 3 Systems?

H/W: Number of Gates are connected,
and Gate works on Binary Logic Signals
On/off, true/false. 0,1,
Bit: Byte; Words(8bit machine, 16, 32, 64, 128 bit Machine)
Data Bus = Width of Words Size
Basic Computer System diagram

Von Neumann and Harvard Architectures

Central Processing Unit (CPU)

Main Memory (M)

Arithmetic-Logic Unit (CA)

Program Control Unit (CC)

I/O Equipment (I, O)
Data Bus is Information Highway

Diagram:

- CPU
- Main memory
- Disk controller
- Communication ports
- Graphic controller

Bus
What is Computer S/W??
Integration of Machine Language(string of 0, 1,) which control computer operations
  Assembler, Compiler
  High Level Language( C++, C, JaVa...)
  >> Operation system(O/S), Applications

What is functions? :
operation, amount of Data, Memory, Storage

What is Computer S/W Engineering?
Phonon, problems
System Analysis; what is the problem
How to solve; Algorithms
Actions; Program source(Code)
Business and IT Systems

Business Expansion, Computer Tech. Progress
System Price, Complexity grow
Regional/ Time Zone Difference

Networked Computing Ideas/ Data Storage Devices
Computer, Communications, Business Process
Computer Networks

Communications: Regeneration, Recovering of sent Information
Time Difference: storage
Regional Differences: Networks of Computers
Data Transfer

H/L Lang >> Encoding >> Packets >> Frame >> Communication >> Frame >> Packets >> Decoding >> Information

Peer to Peer (point to Point); functional equality
Client and Server Systems
    Tier 2;
    Tier 3; Flexibility, Security
Business IT Systems

TPS, DSS, MIS, ESS
ES, ERP, SCM, CRM, KMS

Huge Data Collections: bit > byte > word > File > Record
Data Base

Client – Serve Systems
Communication Protocols
Different Computer (PC, MacOS, Linux, Mainframe,..)
Application and Middleware
Thin, Fat, Ultra-Thin Client
DBMS
Software of Complexity

Importance of Design/Architecture (Structure)

Buying a House
Marriage
Get a Job
Making a Friends

Very Hard to Alter?

Why S/W Complexity? Complex Systems!

S/W

Computer Systems
Communications
Human Systems
Society

Solutions
Results

Garbage in Garbage Out (GIGO)
Capacity of Individual Human Beings makes the System
Software Design

– General definition of design
  • “... the process of applying various techniques and principles for the purpose of defining a device, a process, or a system in sufficient detail to permit its physical realization.”

– Goal:
  • To produce a model or representation depending on the SRS document, that will later be built

– Engineering or Art?
Software Design

- Design activities having two important parts
  - High-level design
    - Outcome is program structure or software architecture
  - Detailed design
    - The data structure and the algorithms of different modules are designed

Question

How to distinguish a good design from a bad design?
Software Design

– Good design is not accomplished by chance

“The beginning of wisdom for a computer programmer is to recognize the difference between getting a program to work, and getting it right.” [Jackson]

– Fundamental concepts provide the framework for “getting it right”
Design Fundamentals

- Abstraction
- Refinement
- Modularity
- Software Architecture
- Control Hierarchy
- Data Structure
- Software Procedure
- Information Hiding
Modularity
System Structure; Architecture

Integration of Sub-Systems(System of Systems)

Component

Module

Reasons to decompose

Functionality
Hierarchy System
Reusability
Interoperability
Modularity

• A complex system may be divided into simpler pieces called *modules*
• A system that is composed of modules is called *modular*
• Supports application of separation of concerns
  – when dealing with a module we can ignore details of other modules
Modularity

• Take the advantage of divide and conquer principle
• Problem is decomposed into modules, and make as much as possible functionally independent to each other
• Reduces complexity of design solution
Top-down Design

• In principle, top-down design involves starting at the uppermost components in the hierarchy and working down the hierarchy level by level.

• In practice, large systems design is never truly top-down. Some branches are designed before others. Designers reuse experience (and sometimes components) during the design process.
Hierarchical Design Structure

System level

Sub-system level
Clean decomposition

• Modules should be less independent of each other satisfying the property like high *cohesion* and low coupling

Modular design

Poor design
Functionally independence

• Performs a single task or function
• Advantages:
  – Error Isolation: Reduce error propagation
  – Scope for reuse: reuse possible due to less dependency
  – Understandability:
    • Complexity is reduced and can be understood in isolation
Functional Independence

- **COHESION**
  - the degree to which a module performs one and only one function

- **COUPLING**
  - the degree to which a module is “connected to” other modules in the system
Cohesion

• Represents how tightly bound the internal elements of the module are to one another
• Tries to capture the *intra-module* characteristics
• Gives the designer an idea about whether the different elements of a module belong together in the same module
• **High cohesion** of a module is desired
A visual representation

(a) high coupling

(b) low coupling
Coupling

- All the modules in a system can not be independent of each other
- The more connection between the modules more connected they are
- The fewer and simpler the connections – easier to understand and modifiable
- *Coupling* determines the strength of interconnections between modules or the measure of interdependence among modules
- *Low coupling* is desired within the modules
Coupling

- Degree of coupling between two modules depends on their interface complexity
  - Interface complexity is determined by the number of types of parameters that are interchanged during invocation of function

<table>
<thead>
<tr>
<th>Interface Complexity</th>
<th>Type of Connection</th>
<th>Type of Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Simple</td>
<td>To module by name</td>
</tr>
<tr>
<td></td>
<td>Obvious</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>Complicated obscure</td>
<td>To internal elements</td>
</tr>
</tbody>
</table>
Neat Arrangement

- Represents the control hierarchy
- Characterized by the following:
  - Layered solution
    - Modules are arranged in layers
    - Superordinate/subordinate modules
    - Visibility: if a module A directly/indirectly calls another module B then module B is visible to module A
  - Control abstraction
    - Should invoke the functions of the modules in the layer immediate below to it
  - Depth and width
    - Provide an indication of the number of levels of control and overall span of control respectively
  - Fan-out
    - Measures the number of modules directly controlled by a given module
  - Fan-in
    - Number of modules directly invoking a given module.
    - High fan-in represents code-reuse and is encouraged
Granularity and Hierarchy

Granularity
The number of modules, the range of functionality

Fine Granularity or a Coarse Granularity
Compromise Functionality and Granularity

Hierarchy

Small modules inside Modules
Information Hiding

- algorithm
- data structure
- details of external interface
- resource allocation policy

"secret"

module

controlled interface

clients

a specific design decision
Why Information Hiding?

• reduces the likelihood of “side effects”
• limits the global impact of local design decisions
• emphasizes communication through controlled interfaces
• discourages the use of global data
• leads to encapsulation—an attribute of high quality design
• results in higher quality software
Interface

The Interactions between Modules
H/W Interfaces
S/W Interfaces
Human Interfaces
Message

Ingredients of Interface

Actions
Parameters and Returns
Action Menus
Data Types
Protocols
Interfaces Are Designed

• intermodular interface design
  – driven by data flow between modules
• external interface design
  – driven by interface between applications
  – driven by interface between software and non-human producers and/or consumers of information
• human-computer interface design
  – driven by the communication between human and machine
Interface Design

Easy to learn?
Easy to use?
Easy to understand?
Interface Design

Typical Design Errors

lack of consistency
too much memorization
no guidance / help
no context sensitivity
poor response
Arcane/unfriendly
Golden Rules

- Place the user in control
- Reduce the user’s memory load
- Make the interface consistent

Place the User in Control

- Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
- Provide for flexible interaction.
- Allow user interaction to be interruptible and undoable.
- Streamline interaction as skill levels advance and allow the interaction to be customized.
- Hide technical internals from the casual user.
- Design for direct interaction with objects that appear on the screen.
Golden Rules

• Reduce the User’s Memory Load
  – Reduce demand on short-term memory.
  – Establish meaningful defaults.
  – Define shortcuts that are intuitive.
  – The visual layout of the interface should be based on a real world metaphor.
  – Disclose information in a progressive fashion.

• Make the Interface Consistent
  – Allow the user to put the current task into a meaningful context.
  – Maintain consistency across a family of applications.
  – If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.
Data Used in the Computers

Integer
Float/Floating Point
Character
String(String of Character)

Data is expressed Name, Type and Value
Layered Infrastructure Software
(Infrastructure Software Layering)

Business Enterprise Infrastructure (Infra)
- Division and Factory facility
- Quality of Employers
- IT System Networking
- Benefits
- Housing/ Environment
- Q of L

University Infrastructure (Infra)
- Good professor
- Building and Lab.
- Student Academic Guidance System
- Student Life Supporting
- Cultural Sites
Good Architecture

Layering Principles
  Special form of Modularity

Application
Middleware
Operating System
Network

Building Construction
  Building House
  Assembly
  Inventory
  Supply
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 *specialized types* of building and software architects
- There are *schools or styles* of building and software architecture
Architectural Styles

An architectural style defines a family of systems in terms of a pattern of structural organization.

More specifically, an architectural style defines a vocabulary of components, connector types, and a set of constraints on how they can be combined.

— Shaw and Garlan
Layered Architectures

A **layered architecture** organises a system into a set of layers each of which provide a set of services to the layer “above”.

- Normally layers are *constrained* so elements only see
  - other elements in the same layer, or
  - elements of the layer below
- **Callbacks** may be used to communicate to higher layers
- Supports the *incremental development* of sub-systems in different layers.
  - When a layer interface changes, *only the adjacent layer is affected*
Definitions

- **Software Architectures** – describe the organization and interaction of software components; focuses on logical organization of software (component interaction, etc.)

- **System Architectures** – describe the placement of software components on physical machines
  - The realization of an architecture may be centralized (most components located on a single machine), decentralized (most machines have approximately the same functionality), or hybrid (some combination).
Architectural Styles

(a) The layered architectural style

(b) The object-based architectural style.
Architectural Styles

- Communication via event propagation, in dist. systems seen often in Publish/Subscribe; e.g., register interest in market info; get email updates
- Decouples sender & receiver; asynchronous communication

(a) The event-based architectural style
Data Centric Architecture; e.g., shared distributed file systems or Web-based distributed systems

Combination of data-centered and event based architectures

Processes communicate asynchronously

The shared data-space architectural style.
The Layers in a Computing Infrastructure

- Application
- Application Components
- Middleware
- Operating System (O/S)
- Network

Client A

Client B

Application Middleware Operating System

Networks Bus
Data and Information in Layers

Network Infrastructure (H/W and S/W)

Data, Information Storage + Data Communication

Data Processing

Information (Letter, number, Words) → Encoding (Mapping) → Data (String Of 0, 1 + Standard)

Communication Protocol → Packet, frame, Encapsulates → Modem, Gate, Router

Data → Decoding → Information, Applications
A Package of Data

Data is stored in the format of the File

Data is communicated as a message (Packets, Frames)

Example of Web browser

Web Server

File System

File

Message

Operating System

Fragmentation

Packets, Frame

Collection of Packets

Assembles

Screen

Web Browser

HTML

Client
Responding to Platform Heterogeneity

In general, Heterogeneous Platforms (Different) (Computer, Data Processing Systems)

IBM, MacOS, Main Frame, Unix

Infrastructures can understand;
The Structures + Interpretation of the Data from Different Systems

1. Functions in a Individual Systems
2. Convert to Standard Data Type, then Regenerate Originals
Middleware, Very Useful Tools with Functions

Data Integrations across the Enterprise Systems

One Key to Enterprise Application incorporating Divisional Application and Heterogeneous Platform
Middleware

• Software that manages and supports the different components of a distributed system. In essence, it sits in the *middle* of the system.
• Middleware is usually off-the-shelf rather than specially written software.
• Examples
  – Transaction processing monitors
  – Data converters
  – Communication controllers
More Layering Principles

Principle #1: Data Structure and ==
Principle #1: Additional structure and Instruction are added
Principle #1: Consistency///
Principle #1; In fraction maintain integrity
Layered application architecture

• Presentation layer
  – Concerned with presenting the results of a computation to system users and with collecting user inputs

• Application processing layer
  – Concerned with providing application specific functionality e.g., in a banking system, banking functions such as open account, close account, etc.

• Data management layer
  – Concerned with managing the system databases
Application layers

- Presentation layer
- Application processing layer
- Data management layer
Thin and fat clients

• **Thin-client model**
  – In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.

• **Fat-client model**
  – In this model, the server is only responsible for data management. The software on the client implements the application logic and the interactions with the system user.
Thin and fat clients

Thin-client model

Client

Presentation

Server

Data management
Application processing

Fat-client model

Client

Presentation Application processing

Server

Data management
A client-server ATM system

Account server
- Tele-processing monitor
- Customer account database

ATMs connected to the account server.
Three-tier architectures

• In a three-tier architecture, each of the application architecture layers may execute on a separate processor
• Allows for better performance than a thin-client approach and is simpler to manage than a fat-client approach
• A more scalable architecture - as demands increase, extra servers can be added
A 3-tier C/S architecture

Client

Presentation

Server

Application processing

Server

Data management
An internet banking system

Client

HTTP interaction

Web server

Account service provision

SQL query

Database server

SQL

Customer account database

Client

Client

Client

Client
A data mining system
The Horizontal Layer Interface
Partitioning the Architecture

• “horizontal” and “vertical” partitioning are required
Horizontal Partitioning

- define separate branches of the module hierarchy for each major function
- use control modules to coordinate communication between functions
Vertical Partitioning: Factoring

- design so that decision making and work are stratified
- decision making modules should reside at the top of the architecture
The Spanning Layer
More on Good Architecture

Abstraction
Encapsulations
Flexibility
Encapsulation

A typical frame

Preamble  Destination Address  Source Address  Data  Padding  CRC
Design Fundamentals

– Abstraction
  • Levels of detail/language used to describe a problem

...notion of “abstraction” permits one to concentrate on a problem at some level of generalization without regard to irrelevant low level details; use of abstraction also permits one to work with concepts and terms that are familiar in the problem environment...

[Wasserman]

• Types of abstractions
  – Procedural abstraction
  – Data abstraction
Procedural Abstraction

• A procedure maps from input to output parameters, may modify its parameters, may have side effects, may return a result

• Procedural abstraction describes what a procedure does, ignore how to does it.
  – Different implementations of the abstraction can differ over details; one implementation can substitute others
Data abstraction

• Defining data abstractions:
  – Name the data type
  – List the operations
  – Give a procedural abstraction for each abstraction
Design Fundamentals (cont.)

– Refinement
  • Top-down strategy

In each step, one or several instructions of the given program are decomposed into more detailed instructions. This successive decomposition or refinement of specification terminates when all instructions are expressed in terms of any underlying computer or programming language. [Wirth]

– Modularity
  • Divide software into separate components that are integrated to solve problem requirements
Design Fundamentals (cont.)

– Software Architecture
  • The hierarchical structure of procedural components & the structure of data
  • Transition between analysis and design

– Control Hierarchy/Program Structure
  • Organization of modules that implies a hierarchy of control
  • Metrics - depth, width, fan-out, fan-in
  • Visibility & connectivity
Control Hierarchy
Design Fundamentals cont..

– Data Structure
  • Logical representation of the relationship among individual data elements
  • Scalar, sequential vector, array, linked list, hierarchical data structure

– Software Procedure
  • Processing details of each module
  • Precise specification includes sequence of events, decision points, repetitive operations, data organization