COMP3027J
Software Architecture

Software Architectural Styles (Call/Return Style)

DENG, YONGJIAN

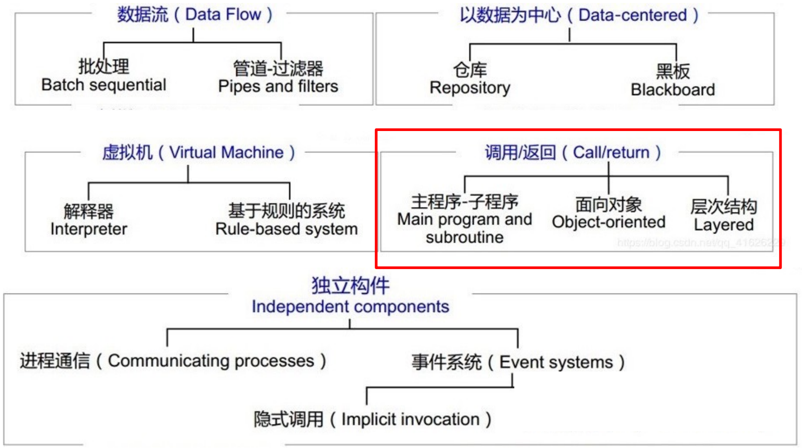
Faculty of Computer Science, BJUT

Data Mining & Security Lab (DMS Lab)

Outline

Outline

Taxonomy of styles



数据流（Data Flow）
批处理 Batch sequential
管道-过滤器 Pipes and filters
以数据为中心（Data-centered）
仓库 Repository
黑板 Blackboard
虚拟机（Virtual Machine）
解释器 Interpreter
基于规则的系统 Rule-based system
调用/返回（Call/return）
主程序-子程序 Main program and subroutine
面向对象 Object-oriented
层次结构 Layered
独立构件 Independent components
进程通信（Communicating processes）
事件系统（Event systems）
隐式调用（Implicit invocation）

Call/Return Architectural Style

Main Program & Subroutine

- Classic programming paradigm: functional decomposition.

Object Oriented

- info (representation, function) hide.

Layered

- Each layer can only communicate with its immediate neighbors.

C/S, B/S, etc.

Call/Return Architectural Style

Main Program & Subroutine

- Single thread of control, divided into several processing steps.
- Functional modules: integrate steps into modules.

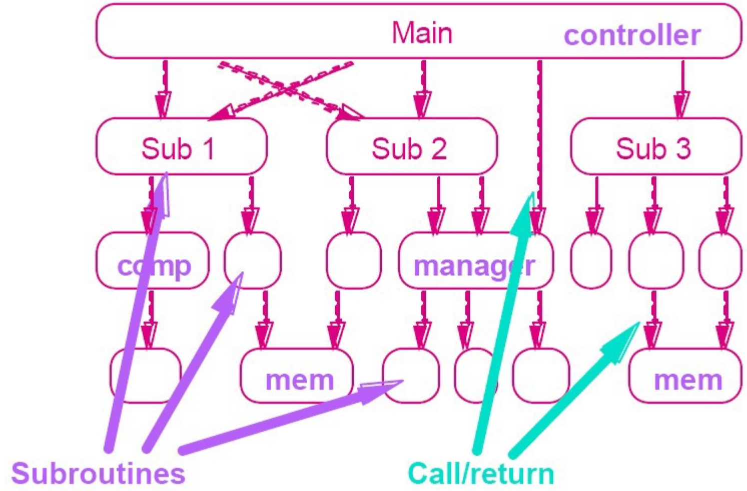
Object Oriented

- Methods (dynamic binding), polymorphism (subclassing), reuse (inheritance).
- Objects are active in different processes/threads (distributed objects).

| C/S, Layered style

Outline

Main Program & Subroutine



Main Program & Subroutine

Problem: This pattern is suitable for applications in which the computation can appropriately be defined via a hierarchy of procedure definitions.

Context: Many programming languages provide natural support for defining nested collections of procedures and for calling them hierarchically. These languages often allow collections of procedures to be grouped into modules, thereby introducing name-space locality. The execution environment usually provides a single thread of control in a single namespace.

Main Program & Subroutine

Solution:

- System model: call and definition hierarchy, subsystems often defined via modularity.
- Components: procedures and explicitly visible data.
- Connectors: procedure calls and explicit data sharing.
- Control structure: single thread.

Module decomposition

Parnas

- Hide secrets. OK, what’s a “secret”?
  - Representation of data
  - Properties of a device, other than required properties
  - Mechanisms that support policies
- Try to localize future change.
  - Hide system details likely to change independently
  - Expose in interfaces assumptions unlikely to change
- Use functions to allow for change.
  - They’re easier to change than a visible representation

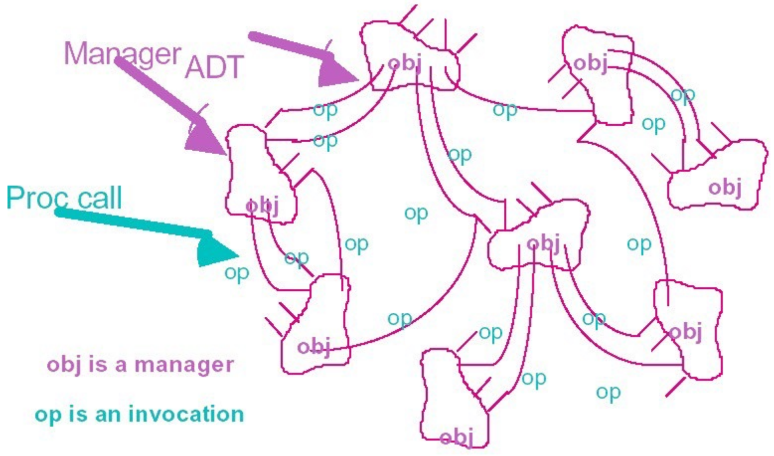
Outline

Object Oriented Style

Target

- Parnas: Hide secrets (not just representations)
- Booch: Object's behavior is characterized by actions that it suffers and that it requires
- Practically speaking:
  - Object has state and operations, but also has responsibility for the integrity of its state
  - Object is known by its interface
  - Object is probably instantiated from a template
  - Object has operations to access and alter state and perhaps generator
  - There are different kinds of objects (e.g., actor, agent, server)

Object Oriented Style



Object Oriented Style

Problem: This pattern is suitable for applications in which a central issue is identifying and protecting related bodies of information, especially representation information.

Context: Numerous design methods provide strategies for identifying natural objects. Newer programming languages support various variations on the theme, so if the language choice or the methodology is fixed, that will strongly influence the flavor of the decomposition.

Object Oriented Style

Solution:

- System model: localized state maintenance
- Components: managers (e.g., servers, objects, abstract data types)
- Connectors: procedure call
- Control structure: decentralized, usually single thread

Charecteristics of Object Oriented

- **Encapsulation**: Restrict access to certain information 封装：限制对某些信息的访问
- **Interaction**: Via procedure calls or similar protocol 交互：通过过程调用或类似的协议
- **Polymorphism**: Choose the method at run-time 多态：在运行时选择具体的操作
- **Inheritance**: Shared definitions of functionality 继承：对共享的功能保持唯一的接口
- **Reuse and maintenance**: Exploit encapsulation and locality to increase productivity 复用和维护：利用封装和聚合提高生产力

Problems of Object Oriented

Managing many objects

- Vast sea of objects requires additional structuring
- Hierarchical design suggested by Booch and Parnas

Managing many interactions

- Single interface can be limiting & unwieldy (hence, “friends”)
- Some languages/systems permit multiple interfaces (inner class, interface, multiple inheritances)

Distributed responsibility for behavior

- Makes system hard to understand
- Interaction diagrams now used in the design

Problems of Object Oriented

Capturing families of related designs

- Types/classes are often not enough
- Design patterns as an emerging off-shoot

Pure O-O design leads to large flat systems with many objects

- Same old problems can reappear
- Hundreds of modules -> hard to find things
- Need a way to impose structure

Need additional structure and discipline

Solutions

Client-server

- Objects are processes 进程就是对象
- Asymmetric: client knows about servers, but not vice versa 不对称：客户端知道服务器，反之则不然

Layered

- Elaboration on client-server C/S 模式的扩展
- Aggregation into run-time strata 运行时层的结合
- Usually small number of layers 通常只有少量的层

Outline

Layered Style

Problem: This style is suitable for applications that involve distinct classes of services that can be arranged hierarchically. There are often layers for basic system-level services, utilities appropriate to many applications, and specific application tasks.

Context: Frequently, each class of service is assigned to a layer and several different patterns are used to refine the various layers. Layers are most often used at the higher levels of design, using different patterns to refine the layers.

Layered Style

Solution:

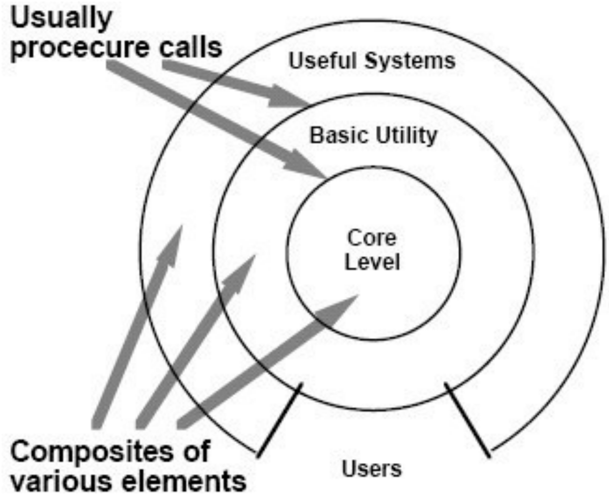
| System model: hierarchy of opaque layers

| Components: usually composites; composites are most often collections of procedures

| Connectors: depends on the structure of components; often procedure calls under restricted visibility, might also be client/server

| Control structure: a single thread

Layered Style



Advantages of Layered System Style

1. Each layer is selected to be a set of related services; thus the architecture provides a high degree of cohesion within the layer.

2. Each layer may hide private information from other layers

3. Layers may use only lower layers, constraining the amount of coupling.

| Easy to add and/or modify a current layer

| Changes in a layer affect at most the adjacent two layers

4.Each layer, being cohesive and coupled only to lower layers, makes it easier for reuse by others and easier to be replaced or interchanged.

| change of DB touches only the data store/access layer, change of browser only changes the presentation layer.

Disadvantages of Layered System Style

1. Strict Layered Style may cause performance problems depend- ing on the number of layers.

| If any layer only uses the layer directly below it, then it is a Strict Layered Style.

| If a layer may use any of the layers below it, then it is a Relaxed Layer Style。

2. Not always easy to structure in clean layers.

3. Layers call each other, affecting performance.

Outline

Client/Server (C/S) Style

Components are clients and servers.

- Client: an application that makes requests (to the servers) and handles input/output with the system environment.
- Server: an application that responses requests from clients.

Connectors are RPC-based network interaction protocols.

- 连接器是基于 RPC 的网络交互协议
- RPC（Remote Procedure Call）远程过程调用

Client-Server Style

Why Client / Server?.

- The C/S software architecture is based on unequal resources and was proposed for sharing. It matured in the 1990s.
- Multiple users want to share and exchange data (Network applica- tions)
- Typical application area: distributed multi-user (business) informa- tion systems

Characteristics of C/S Style

1. **Dependencies: client depends on the server.**
   - If any layer only uses the layer directly below it, then it is a Strict Layered Style.
   - If a layer may use any of the layers below it, then it is a Relaxed Layer Style.
2. **Topology.**
   - one or more clients may be connected to a server.
   - there are no connections between clients.
3. **Mobility: easily supports client mobility.**
4. **Security: typically controlled at the server, also possible at the application/business layer**

Advantages of Client/Server Architectures

1. Makes effective use of network systems. May require cheaper hardware.

2. Easy to add new servers or upgrade existing servers.

3. Allows sharing of data between multiple users.

4. Scalable: add new clients.

Disadvantages of Client/Server Architectures

1. **Redundant management in each server.**

2. **Hard to find out what servers and services are available. No central register of names and services.**

3. **Difficult to change functionalities of server and client.**

   | Changing application logic is difficult if it is distributed over C & S.

4. **Scalability of applications is limited by server & network capacity.**

Two Tier Client/Server Architectures

Processing management split between user system interface environment and database management server environment.

| Tier 1: user system interface: In user’s desktop environment.

| Tier 2: database management services: In a server.

The server is responsible for data management, and the client completes the interactive tasks with users. ”Thick Client, Thin Server”.

Two Tier Client/Server Architectures

Limitations.

| High requirements for client software and hardware configuration, bloated client

| Complicated client program design

| Data security is bad. Client programs can directly access the database server.

| Single information content and form

| Different styles of the user interface, complex use, no use of promo- tional use

| Software maintenance and upgrades are difficult. Software on each client requires maintenance

Three Tier Client/Server Architectures

Compared with the two-tier C/S structure, an application server is added.

The entire application logic resides on the application server, only the presentation layer exists on the client: “thin client”.

The application function is divided into three layers: Presentation tier, Application (Logic) tier and Data tier.

- The presentation tier is the user interface portion of the application. Usually using a graphical user interface.
- The application tier is the main body of the application, implementing specific business processing logic.
- The data tier is the database management system.
- The above three layers are logically independent.
- Usually only the presentation tier is configured in the client.

Browser/Server Architectures

B/S architecture is a special case of three-tier C/S architecture. The client has an http browser.

- In order to enhance the function, it is often necessary to install flash, jvm and some special plug-ins.

Use standard http/https protocol, save a lot of trouble.

Can only ”pull”, not ”push”.

Communication between clients can only be relayed through the server.

Browser/Server Architectures

Limited utilization of client resources and other network resources.

The security of B/S structure is difficult to control (SQL injection attack...).

The application system of B/S structure is much lower than the C/S architecture in terms of data query and other corresponding speeds.

The load on the server is heavy, and the resources of the client are wasted.

- Use jvm, flash, ActiveX and other client computing technologies to solve.

Thank you!