Architectural Design

CSCE 740 - Lecture 14 - 10/19/2015

Architectural Styles

Today's Goals

- Define what "architecture" means when discussing software development.
- Discuss methods of documenting and planning software architecture (and why this is a good practice).
- Discuss common architectural models.

What is Software Architecture?

"Software architecture is the **fundamental organization** of a system, embodied in its **components**, their **relationships** to one another and the environment, and the **principles** governing its design and evolution."

- IEEE Definition

Architecture Parallels

- Architectural plans are the technical interface between the customer and the contractor building the building.
  - (and the software)
- A bad architectural design for a building cannot be rescued by good construction.
  - (same for software)
- There are specialist types of building architects and architecture styles.
  - (you get the point)

Why Explicitly Plan Architecture?

- Enable stakeholder communication
  - High-level presentation of the system.
- Enables system analysis
  - Can look for problems before coding.
- Enables large-scale reuse
  - Planning subsystems as independent entities allows their reuse in other systems.
- Bad architectural design means bad security
  - Controlling access is the first line of defense.

How We Partition a System

- System Structuring
  - The system is decomposed into several subsystems and communications between those subsystems are identified.
- Control Modeling
  - A model of the control relationships between the different parts of the system is established.
- Modular Decomposition
  - The subsystems are decomposed into modules to structure the implementation.

Architectural Qualities

- Performance
  - Minimize communication using fewer, larger components, stored on a local machine. Consider opportunities for parallel execution.
- Security
  - Layer the architecture, with the most critical components protected in the innermost layers.
- Safety
  - Encapsulate safety-related operations within a small number of local components.

Architectural Qualities

- Availability
  - Include redundant components so that they can be replaced or updated without stopping operation.
- Maintainability
  - Design system with large number of self-contained components that may readily be changed. Separate data from consumers, avoid shared data structures.

Architectural Qualities Conflict

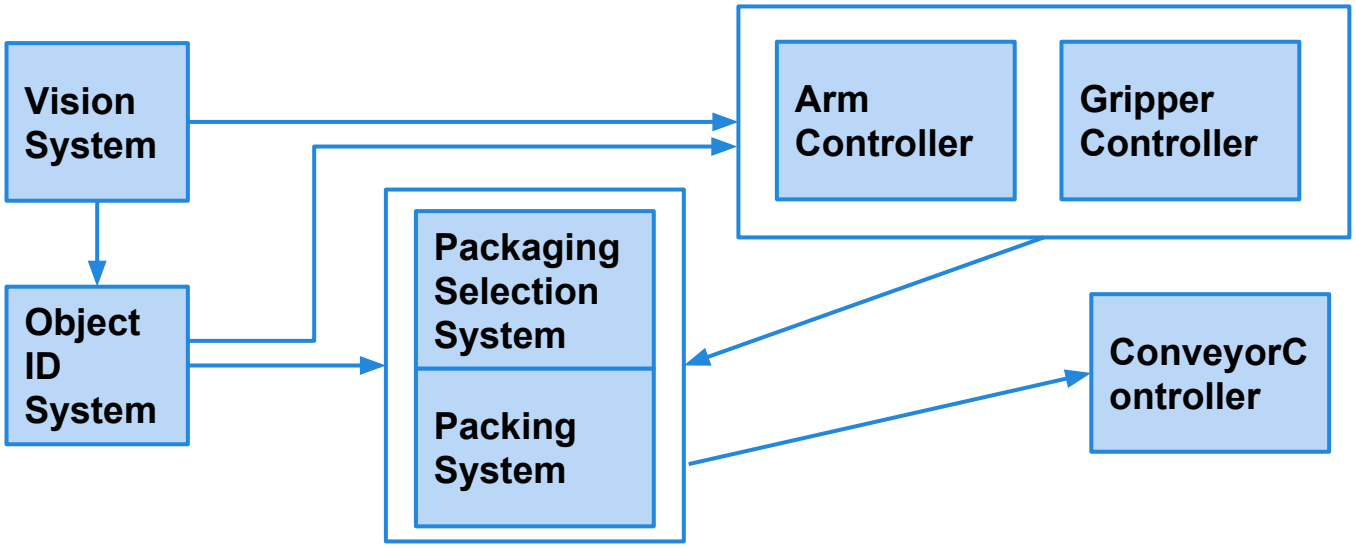
These qualities often conflict. It is hard to achieve multiple qualities at once.

- Using fewer subsystems improve performance, but hurts maintainability.
- Introducing redundant data improves availability, but makes security more difficult.
- Localizing safety-related features usually introduces more communication between subsystems, degrading performance.

System Structuring

System Structuring

- How we decompose the system into interacting subsystems.
- Can be visualized as block diagrams presenting an overview of the system structure.



Structuring Views

When structuring the system, consider:

- **Static View**
  - Logical view - given the services we want to offer, how does it make sense to delegate responsibility? Relate requirements to entities in the system.
- **Dynamic View**
  - Visualize entities communicating during runtime execution. Useful for judging performance, security, availability.
- **Physical View**
  - How hardware and software communicate and how software is distributed across processors.

Example: The ASW

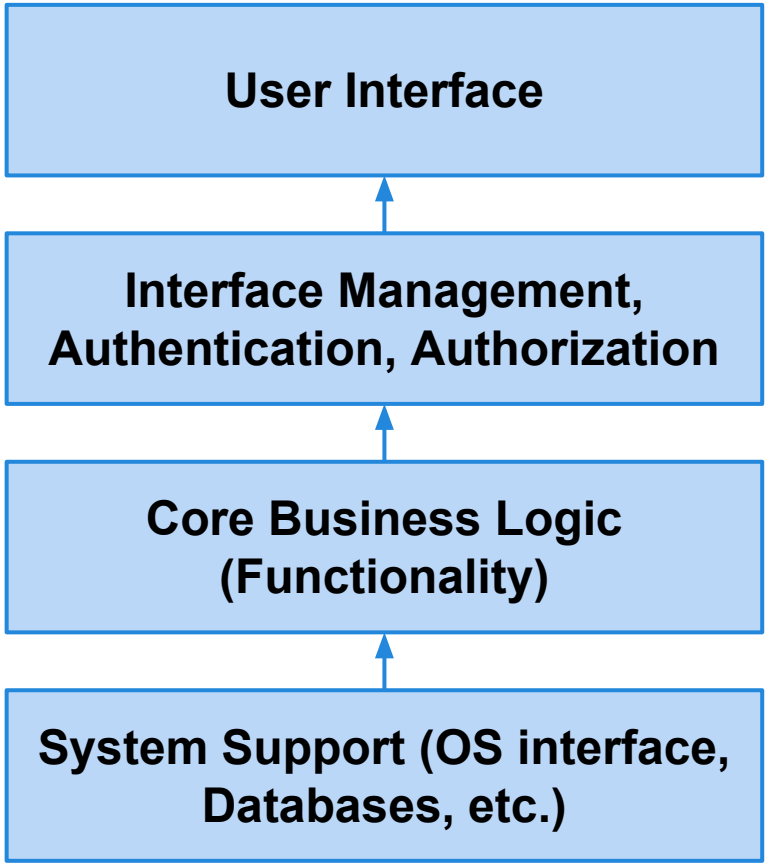
You are designing control software for an aircraft. In such software, multiple behaviors are based on altitude. The software interfaces with one of more altimeters, makes autopilot decisions based on this information, and outputs status information to a monitor that is viewed by the pilot. If altitude drops below certain thresholds, the system will send warnings to that monitor and, if autopilot is engaged, will attempt to correct the plane's orientation.

How would you architect the system?

Architectural Models

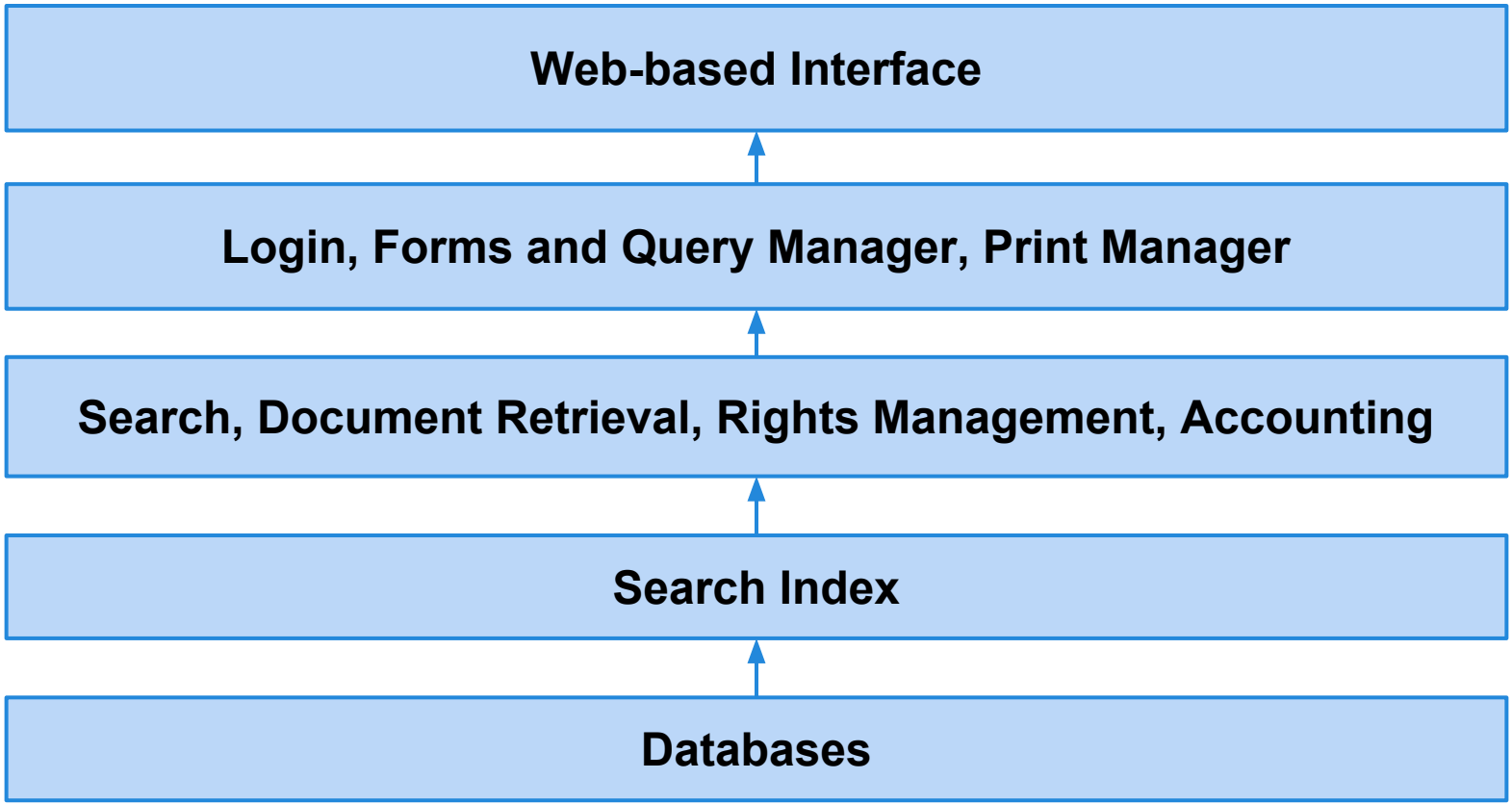
- Four common models: layered, shared repository, client/server, pipe & filter
- The model used affects the performance, robustness, availability, maintainability, etc. of the system.
- Complex systems might not follow a single model - mix and match.

Layered Model



- System functionality organized into layers, with each layer only dependent on the previous layer.
- Allows elements to change independently.
- Supports incremental development.

Copyright Management Example



Layered Model Characteristics

Advantages

- Allows replacement of entire layers as long as interface is maintained.
- When changes occur, only the adjacent layer is impacted.
- Redundant features (authentication) in each layer can enhance security and dependability.

Disadvantages

- Clean separation between layers is often difficult.
- Performance can be a problem because of multiple layers of processing between call and return.

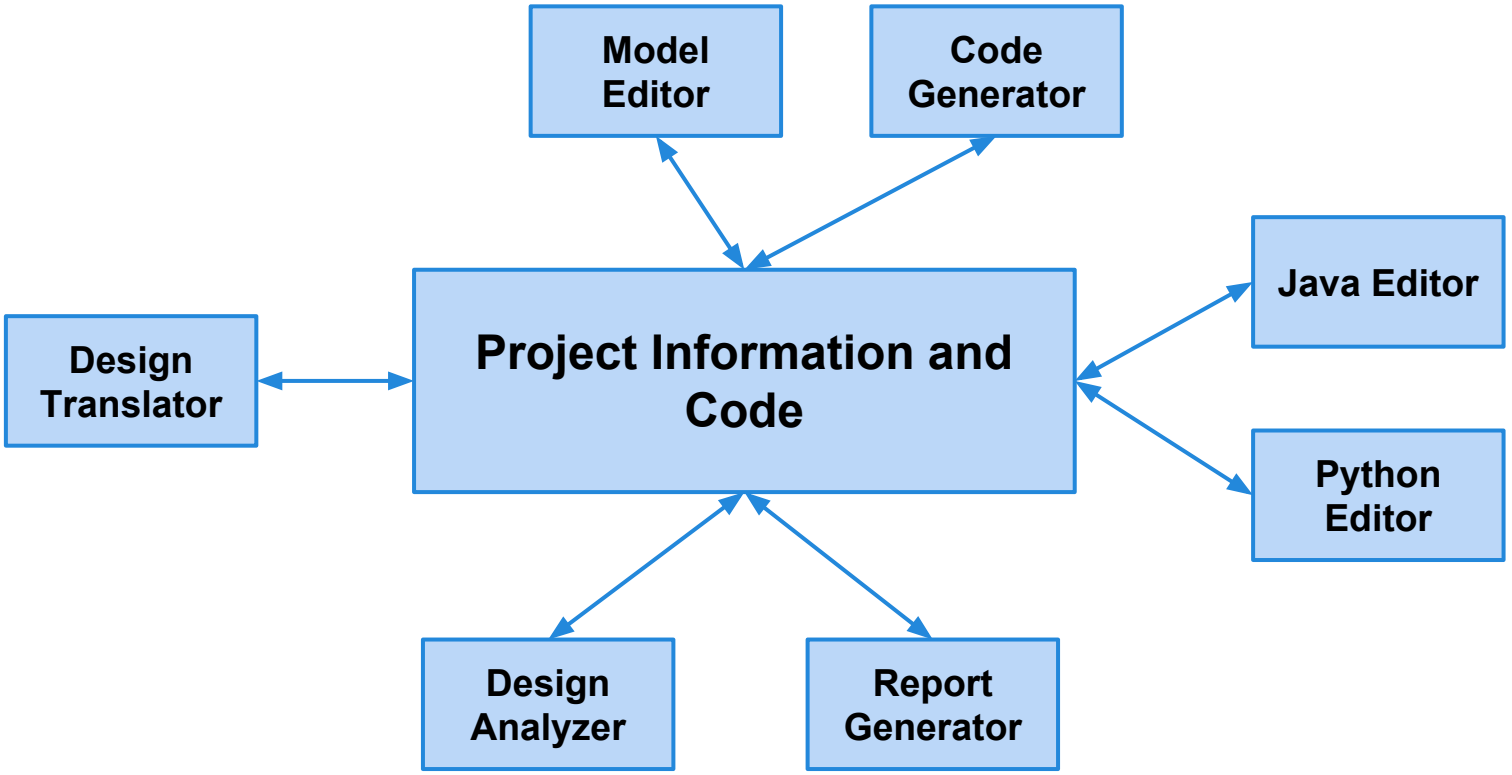
The Repository Model

Subsystems often exchange and work with the same data. This can be done in two ways:

- Each subsystem maintains its own database and passes data explicitly to other subsystems.
- **Shared data is held in a central repository and may be accessed by all subsystems.**

Repository model is structured around the latter.

IDE Example


Model Editor
Code Generator
Design Translator
Project Information and Code
Java Editor
Python Editor
Design Analyzer
Report Generator

Repository Model Characteristics

Advantages

- Efficient way to share large amounts of data.
- Components can be independent.
  - May be more secure.
- All data can be managed consistently (centralized backup, security, etc)

Disadvantages

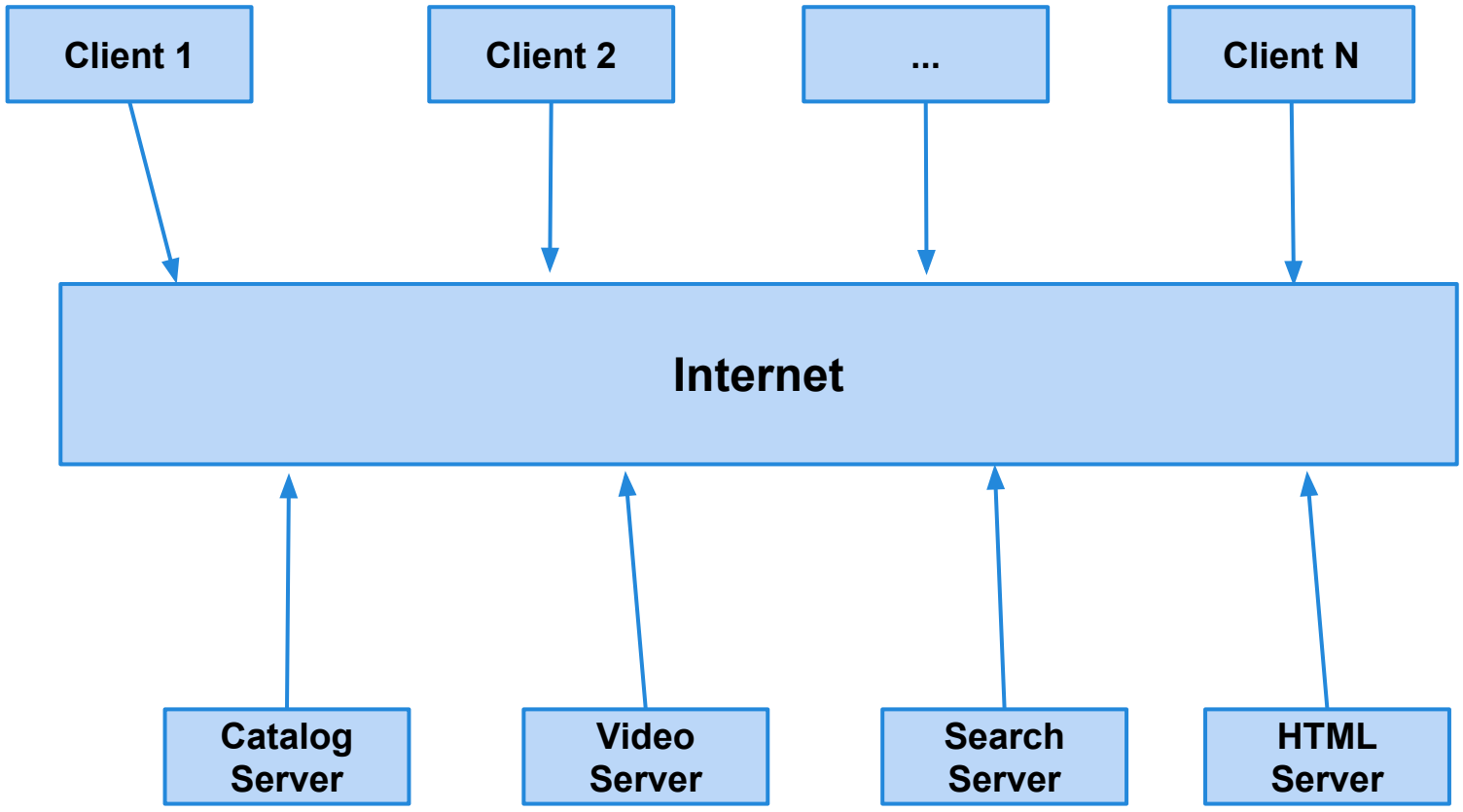
- Single point of failure.
- Subsystems must agree on a data model (inevitably a compromise).
- Data evolution is difficult and expensive.
- Communication may be inefficient.

Client-Server Architecture

Functionality organized into services, distributed across a range of components:

- A set of servers that offer services.
  - Print server, file server, code compilation server, etc..
- Set of clients that call on these services.
  - Through locally-installed front-end.
- Network that allows clients to access these services.
  - Distributed systems connected across the internet.

Film Library Example



Client-Server Model Characteristics

Advantages

- Distributed architecture.
- Failure in one server does not impact others.
- Makes effective use of networked systems and their CPUs. May allow cheaper hardware.
- Easy to add new servers or upgrade existing servers.

Disadvantages

- Performance is unpredictable (depends on system and network).
- Each service is a point of failure.
- Data exchange may be inefficient (server -> client -> server).
- Management problems if servers owned by others.

Interactions Between Clients/Servers

- REST is a simple architecture for managing interactions between clients and servers.
- Allows clients and servers to pass resources around through requests and responses.
- Simple API that allows interactions tailored to clients as diverse as phone apps and websites.
  - Same API, up to client to present information.

HTTP

- Protocol used to send documents back and forth on the internet.
- Clients initiate conversation, servers reply.
- Messages composed of header (metadata) and body (data).
- The header is the most important part.

VERB resource HTTP/1.1

Host: example.com

...

HTTP Requests

Resources are URLs.

- Should be described using nouns.
  - Good: `/clients/rbob`
  - Bad: `/clients/remove`
- Everything needed to identify a resource should be in the URL.

Actions described through HTTP verbs: GET, DELETE, PUT, and POST.

HTTP Verbs

GET

- `GET /clients/rbob`
- Transmit the resource to the client.

PUT

- `PUT /clients/rbob`
- Creates a resource on the server.

DELETE

- `DELETE /clients/rbob`
- Remove a resource from the server.

POST

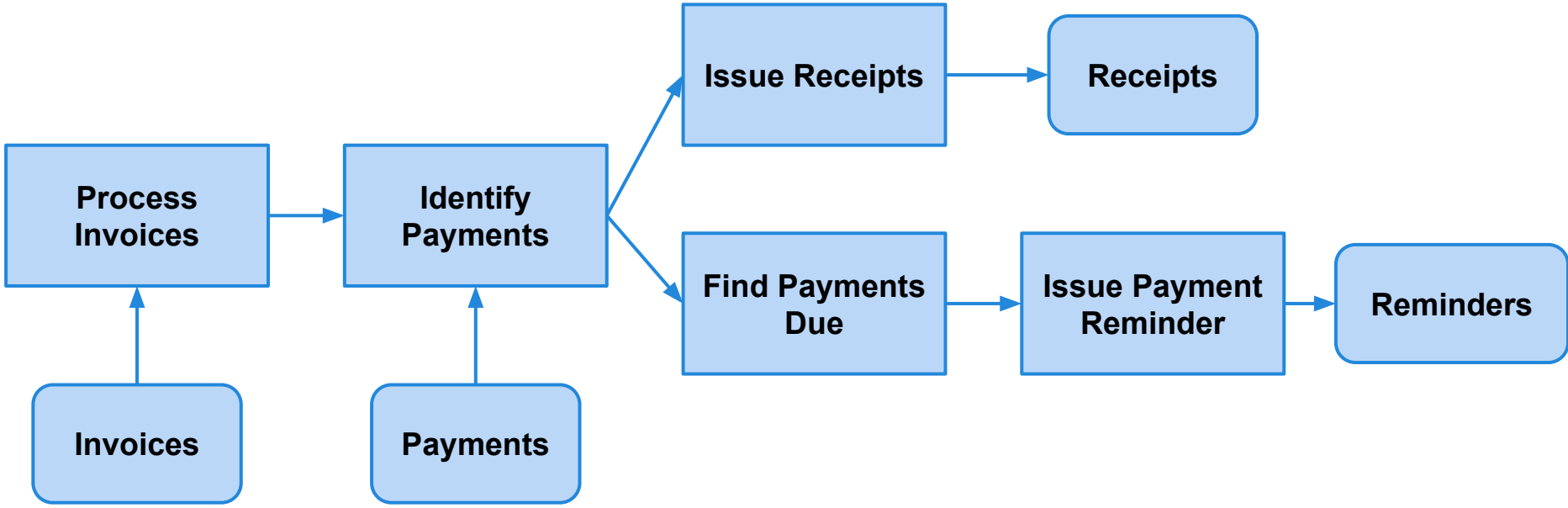
- `POST /clients/rbob`
- Trigger processing on the server.
  - Sometimes used like PUT: POST for creation, PUT for updates
  - Sometimes used to trigger pre-set operations on resources.

Pipe and Filter Architecture

Input is taken in by one component, processed, and the output serves as input to the next component.

- Each processing step is a data transformation.
- Transformations may execute sequentially or in parallel.
- Data can be processed by item or in batches.
- From Unix command line:
  - `cat file.txt | cut -d, -f 2 | sort -n | uniq -c`

Customer Invoicing Example



Pipe and Filter Characteristics

Advantages

- Easy to understand communication between components.
- Supports subsystem reuse.
- Can add features by adding new subsystems to the sequence.

Disadvantages

- Format for data communication must be agreed on. Each transformation needs to accept and output the right format.
- Increases system overhead.
- Can hurt reuse if code doesn’t accept right data structure.

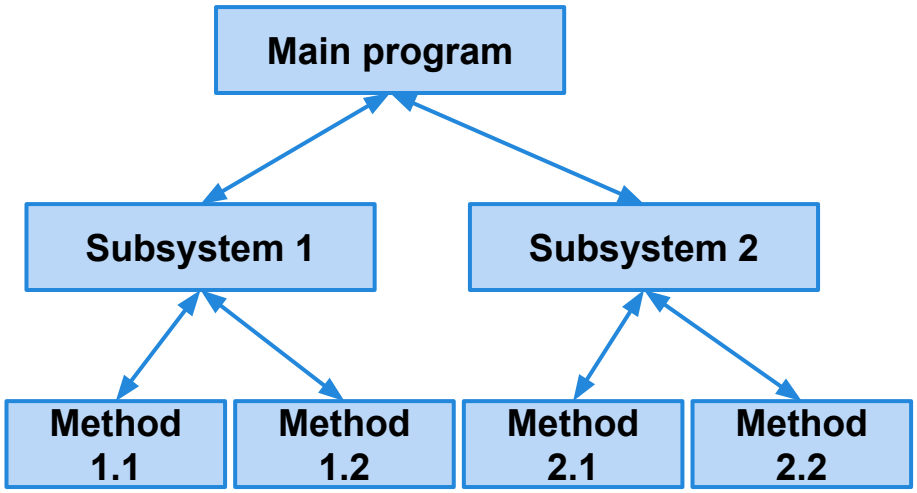
Control Modeling

Control Models

- A model of the control relationships between the different parts of the system is established.
- During execution, how do the subsystems work together to respond to requests?
  - **Centralized Control:**
    - One subsystem has overall responsibility for control and stops/starts other subsystems.
  - **Event-Based Control:**
    - Each subsystem can respond to events generated by other subsystems or the environment.

Centralized Control: Call-Return

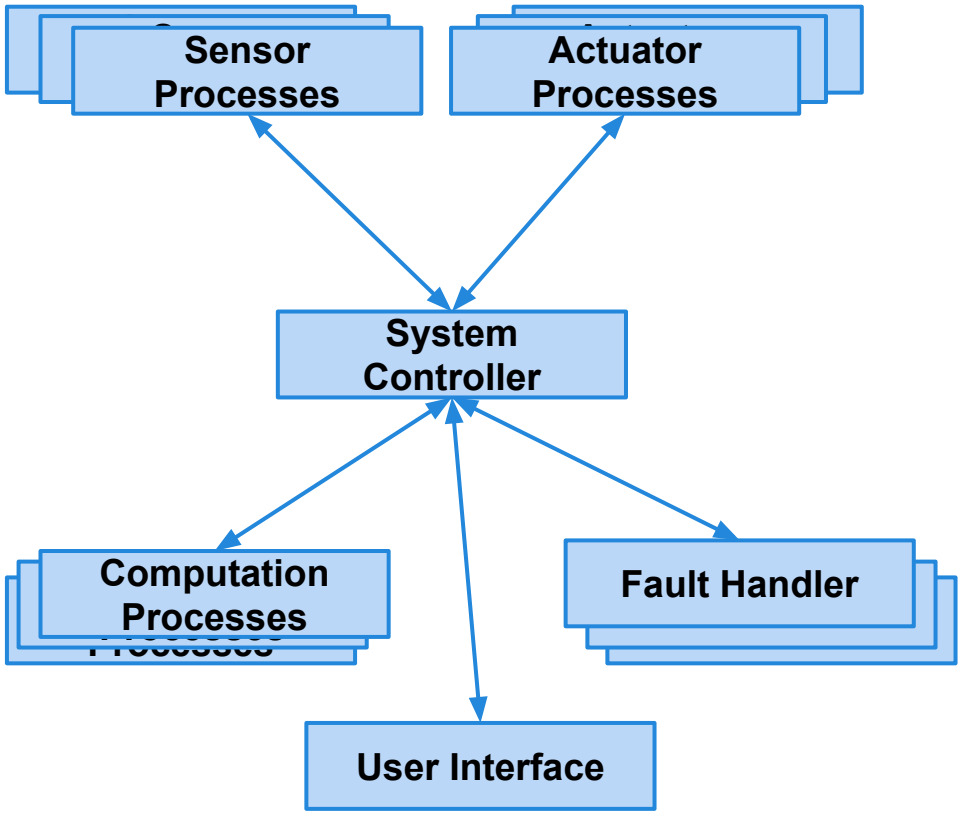
A central piece of code (Main) takes responsibility for managing the execution of other subsystems.



Call-Return Model

- Applicable to sequential systems.
- Top-down model where control starts at the top of a subroutine and moves downwards.

Centralized Control: Manager Model



Manager Model

- Applicable to concurrent systems.
- One system component controls the stopping, starting, and coordination of other system processes.

Decentralized Control: Event-Driven Systems

Control is driven by externally-generated events where the timing of the event is out of control of subsystems that process the event.

- Broadcast Model
  - An event is broadcast to all subsystems.
  - Any subsystem that needs to respond to the event does do.
- Interrupt-Driven Model
  - Events processed by interrupt handler and passed to proper component for processing.

Broadcast Model

An event is broadcast to all subsystems, and any that can handle it respond.

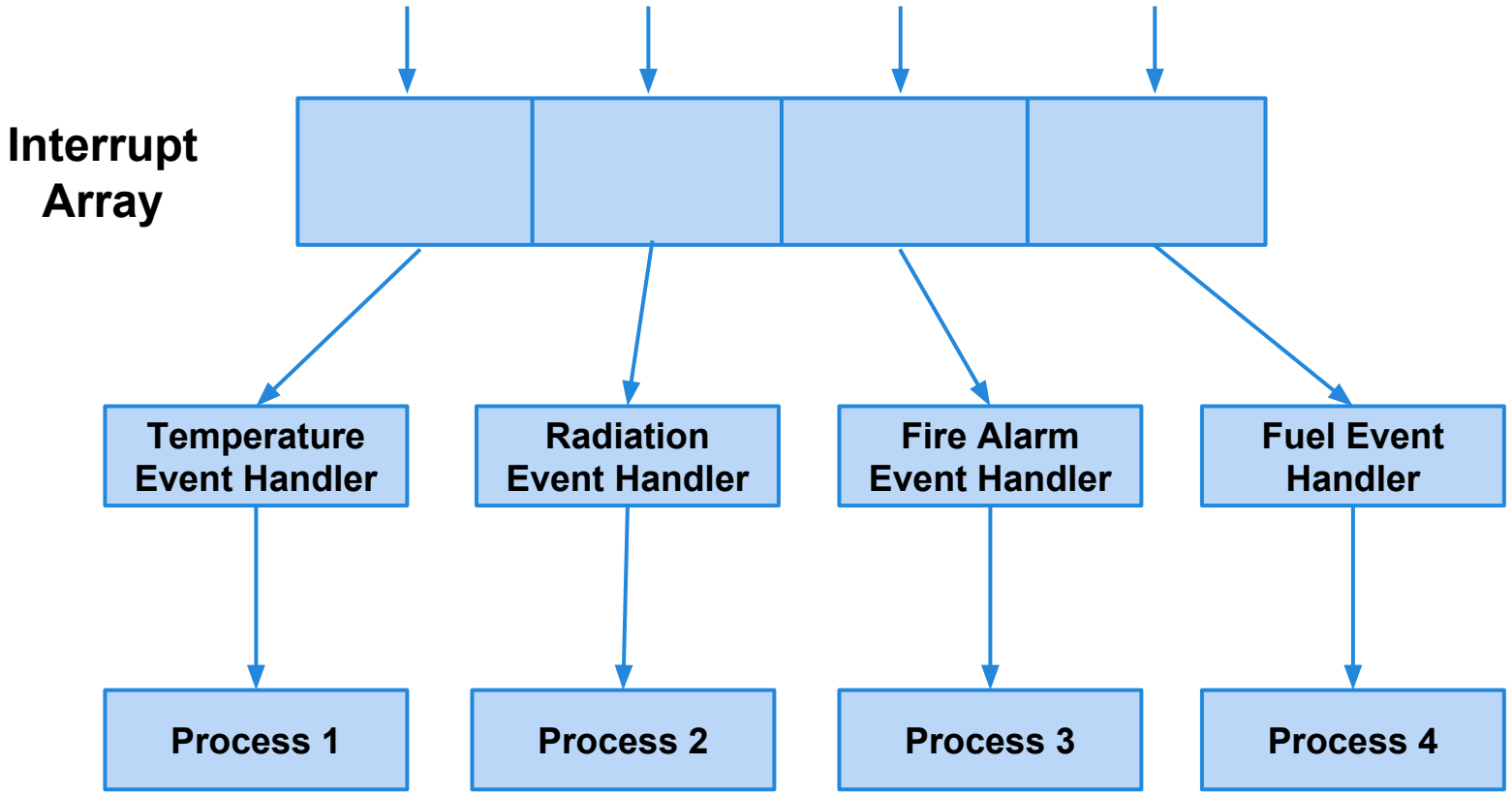
- Subsystems can register interest in specific events. When these occur, control is transferred to the registered subsystems.
- Effective for distributed systems. When one component fails, others can potentially respond.
- However, subsystems don't know when or if an event will be handled.

Interrupt-Driven Model

Events processed by interrupt handler and passed to proper component for processing.

- For each type of interrupt, define a handler that listens for the event and coordinates response.
- Each interrupt type associated with a memory location. Handlers watch that address.
- Used to ensure fast response to an event.
- Complex to program and hard to validate.

Nuclear Plant Interrupt Example



Example: The ASW

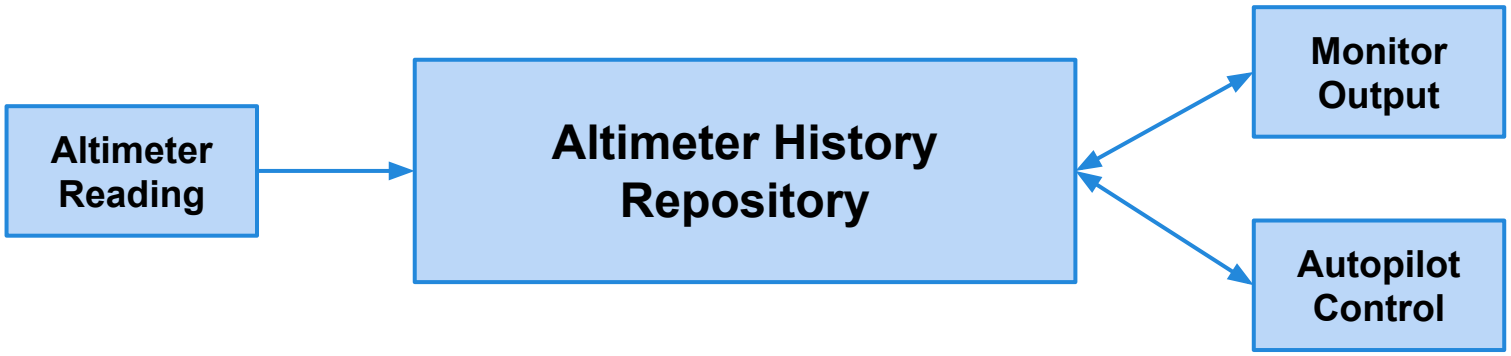
You are designing control software for an aircraft. In such software, multiple behaviors are based on altitude. The software interfaces with one of more altimeters, makes autopilot decisions based on this information, and outputs status information to a monitor that is viewed by the pilot. If altitude drops below certain thresholds, the system will send warnings to that monitor and, if autopilot is engaged, will attempt to correct the plane's orientation.

- **Perform system structuring. Try to use one or more of the models covered.**
- **Perform control modeling. How should events be handled?**

ASW Solution

- **Perform system structuring. Try to use one or more of the models covered.**

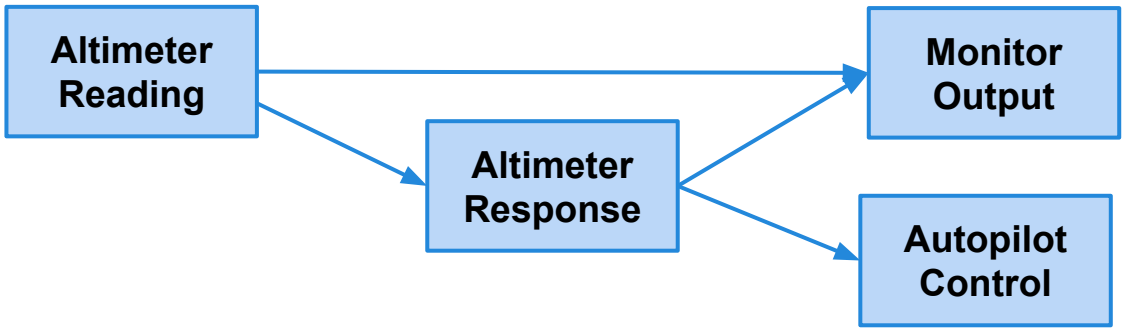
Option 1: Repository Model



ASW Solution

- **Perform system structuring. Try to use one or more of the models covered.**

Option 2: Pipe and Filter



ASW Solution

- **Perform control modeling. How should events be handled?**

Depends on how you answered the previous question, but a natural option would be an Interrupt-Driven Model.

Handlers for new altimeter readings, for error flags triggered by altimeter processing code.

Modular Decomposition

The rest of design - subsystems need to be decomposed into modules.

- How we get from a “system” to classes and methods.
- We’ll start to talk about this next time.

Key Points

- The software architect is responsible for deriving a system structure, a control model, and a modular decomposition.
- Architectural models can help organize a system.
  - But, Large systems rarely conform to one model.
- Models include layered, repository, client-server, and pipe and filter models.
- Control models include centralized control and event-driven models.

Next Time

- Object-oriented design and class diagrams
- Reading
  - Sommerville, chapter 6
  - Fowler UML, chapter 3
    - (or any resource on class diagrams)

- Homework: Project 3 is up.