ADVANCED SOFTWARE DESIGN
LECTURE 5
SOFTWARE ARCHITECTURE

Dave Clarke
OVERVIEW

What **software architecture** is and why it is interesting

Who are the **stakeholders**

What **software qualities** does software architecture concern

**UML diagrams** expressing aspects of software architecture

**Architectural styles** or software architectural design patterns
What is the purpose of software architecture?
SOFTWARE ARCHITECTURE

A **Software Architecture** defines:

- the components of the software system
- how the components use each other’s functionality and data
- how control is managed between the components

The highest level of design – large-scale structure of solution
Which stakeholders have an interest in a software development effort?
SOFTWARE QUALITIES

What are the various software qualities (= non-functional requirements) that software architecture is concerned with?

Which software qualities are of interest to which stakeholders?

Which software qualities are conflicting?
UML DIAGRAMS
**COMPONENT DIAGRAMS**

**Component** – a set of related operations that share a common purpose

**Interface** – the set of operations available to other sub-systems

![Diagram showing component interaction](attachment:image.png)
PACKAGE DIAGRAMS

Packages
Hierarchy
Dependency

Useful for expressing the dependencies between major elements of a system.
DEPLOYMENT DIAGRAMS

Express the physical deployment of software artefacts to hardware nodes – static view of run-time configuration

Use when application spans several machines.

Nodes correspond to

- devices (e.g., servers, mobile devices)
- specific execution environments (application servers, rule engines, operating system, virtual machines, database engines, web browser).

Nodes connected by communication paths (middleware, protocol)
DEPLOYMENT DIAGRAMS
GUIDELINES

1. **Identify the scope of the model** – single application, group of applications?

2. **Consider fundamental technical constraints** – existing systems, robustness, who/how users connect, middleware, hardware, software, ...

3. **Identify distribution architecture** – Fat client, thin client, 3-tier?

4. **Identify nodes and their connections** – What kinds of nodes do you have? How connected?

5. **Distribute software to nodes** – indicate also critical information for whoever implements or deploys system
SOFTWARE QUALITIES

What are ways of addressing various software qualities?
CONFLICTING QUALITIES

Which software qualities are conflicting?

Why?

How can these conflicts be resolved?
WORDFEUD

- What are the key architectural concerns for Wordfeud?
- How could these be addressed?
ARCHITECTURAL STYLES
(= SA DESIGN PATTERNS)
ARCHITECTURAL STYLES

Object-oriented

Client server; object broker; peer to peer

Pipe and filter

Event based – publish/subscriber

Layered – Three-tier, Four-tier

Repositories: blackboard, Model/View/Controller (MVC)

Process control

The diagrams that follow capture only high level of abstraction — still need to place components.
MODEL-VIEW-CONTROLLER

Properties

- One central model, many views (viewers)
- Each view has an associated controller
- The controller handles updates from the user of the view
- Changes to the model are propagated to all the views
MODEL-VIEW-CONTROLLER (MVC)

- Model contains domain knowledge
- Views only display data
- Controllers only manage interaction sequences

Model does not depend on views or controllers

Subscribe/notify mechanism
EXAMPLE
MODEL-VIEW-CONTROLLER

Navigability arrows show the directions in which messages will be sent.
MVC INTERACTION
OBJECT-ORIENTED ARCHITECTURES

- Example
  - Abstract data types (modules)
- Interesting properties
  - data hiding (internal data representations are not visible to clients)
  - can decompose problems into sets of interacting agents
  - can be multi-threaded or single thread
- Disadvantages
  - objects must know the identity of objects they wish to interact with
VARIANT: CLIENT-SERVER

- Interesting properties
  - Clients do not need to know about one another
  - Breaks the system into manageable components
  - Independent flow of control
  - Server generally responsible for persistence and consistency of data

- Disadvantages
  - Client objects must know the identity of the server
CLIENT-SERVER

Client/Server communication via remote procedure call or common object broker (e.g. CORBA, Java RMI, or HTTP)

Distributed systems. e.g., web services, system with central DBMS

Variants

– thick clients have their own services

– thin ones get everything from servers
VARIANT: OBJECT BROKER

• Interesting Properties
  • Adds a broker between the clients and servers
  • Clients no longer need to know which server they are using
  • Can have many brokers, many servers.

• Disadvantages
  • Broker can become a bottleneck
  • Degraded performance
BROKER ARCHITECTURE EXAMPLE
VARIANT: PEER-TO-PEER

• Interesting Properties
  • Find peers via server or broadcast.
  • Interact subsequently with peers.
  • Reduces bottleneck. Robust to peer failure.

• Disadvantages
  • Server can become a bottleneck
  • Peers have only incomplete picture – synchronisation is (virtually) impossible
**PIPE AND FILTER**

- Examples
  - Unix command shell
  - compiler chain: lexical analysis → parsing → semantic analysis → code generation
  - signal processing

- Interesting properties:
  - filters don't need to know anything about what they are connected to
  - filters can be implemented in parallel
  - behaviour of the system is the composition of behaviour of the filters
  - specialised analysis such as throughput and deadlock analysis is possible

Example:

```
grep gustav < foo.txt | sort | cut -f2-3
```
EVENT-BASED (IMPLICIT INVOCATION)

- Examples
  - debugging systems (listen for particular breakpoints)
  - database management systems (for data integrity checking)
  - graphical user interfaces

- Interesting properties
  - announcers of events don’t need to know who will handle the event
  - supports re-use, and evolution of systems (add new agents easily)

- Disadvantages
  - Components have no control over ordering of computations
LAYERED SYSTEMS

• Examples
  • Operating Systems
  • communication protocols
• Interesting properties
  • Support increasing levels of abstraction during design
  • Support enhancement (add functionality) and re-use
  • can define standard layer interfaces
• Disadvantages
  • May not be able to identify (clean) layers
LAYERS AND PARTITIONS

- Layering
  - Hierarchical decomposition \(\rightarrow\) tree of sub-systems
  - “Horizontal division”
  - Open vs closed layers
- Partitioning
  - “Vertical division”: Each set handles a function
  - Extreme form of decoupling: Semi-independent
EXAMPLE: 3-LAYER DATA ACCESS
OPEN VS CLOSED ARCHITECTURE

closed architecture
• each layer only uses services of the layer immediately below;
• minimises dependencies between layers and reduces the impact of a change.

open architecture
• a layer can use services from any lower layer.
• More compact code, as the services of lower layers can be accessed directly
• Breaks the encapsulation of layers, so increase dependencies between layers
HOW MANY LAYERS?

2 layers:
- application layer
- database layer
- e.g., simple client-server model

3 layers (three tier):
- separate out the business logic
- helps make both user interface and database layers modifiable

4 layers (four tier):
- separate applications from the domain entities that they use
- boundary classes in presentation layer
- control classes in application layer
- entity classes in domain layer

Partitioned 4 layers:
- identify separated applications
REPOSITORIES

- Examples
- databases
- blackboard expert systems
- programming environments

- Interesting properties
  - can choose where the locus of control is (agents, blackboard, both)
  - reduce the need to duplicate complex data

- Disadvantages
  - blackboard becomes a bottleneck
REPOSITORY

- Sub-systems access and modify a single data structure
- Complex, changing data
- Concurrency & data consistency
- Control by sub-systems or by repository (blackboard)
- Disadvantage: Possibly performance bottlenecks and reduced modifiability
- E.g. databases, IDE’s, tuple spaces
PROCESS CONTROL

Examples

- aircraft/spacecraft flight control systems
- controllers for industrial production lines, power stations, etc.
- chemical engineering

Interesting properties

- separates control policy from the controlled process
- handles real-time, reactive computations

Disadvantages

- difficult to specify the timing characteristics and response to disturbances
CONCLUDING REMARKS
CONCLUDING REMARKS

What **software architecture** is and why it is interesting

Who are the **stakeholders**

What **software qualities** does software architecture concern

**UML diagrams** expressing aspects of software architecture

**Architectural styles** or software architectural design patterns