ICS 52: Introduction to Software Engineering

Winter Quarter 2004
Professor Richard N. Taylor
Lecture Notes
Week 3: Architectures

http://www.ics.uci.edu/~taylor/ICS_52_WQ04/syllabus.html

Copyright 2004, Richard N. Taylor. Duplication of course material for any commercial purpose without written permission is prohibited.
"Magician Coder" View of Development

Requirements

(Here a Miracle happens)

Code
A Professional View

Requirements

Architecture

Code

University of California, Irvine
ICS 52 Life Cycle

- **Requirements phase**
  - Verify

- **Architecture/Design phase**
  - Verify

- **Implementation phase**
  - Test

- **Testing phase**
  - Verify
ICS 52 Software Life Cycle

- Requirements specification
  - Interview customer (TA)
  - Focus on “what”, not “how”
- Architectural and module design
  - Based on provided “official” requirements specification
  - Focus on system structure and interfaces
- Implementation
  - Based on provided “official” design
  - Focus on good implementation techniques
- Testing
  - Based on provided “official” implementation
  - Focus on fault scenarios and discovery
Bridges...
Architecture of Buildings

- **Types** (Domains): office building, shepherd's shelter, detached home, apartment building, aircraft hanger
  - Domain-specific software architectures
- **Styles**: colonial, Victorian, Greek revival, Mediterranean, Bauhaus
  - Software system organization paradigms
- **Building codes**: electrical, structural, ...
  - Constraints on how the building can be legally built
- **Blueprints and drawings**
  - Formal specification of supporting details
Architectural Design

Buildings

Elements
- Floors
- Walls
- Rooms

Types
- Office building
- Villa
- Aircraft hanger

Styles
- Colonial
- Victorian
- Southwestern

Rules and regulations
- Electrical
- Structural

Software

Elements
- Components
- Interfaces
- Connections

Types
- Office automation
- Game
- Space shuttle control

Styles
- Pipe and filter
- Layered
- Implicit invocation

Rules and regulations
- Use of interfaces
- Methods of change
Design

◆ Architectural (software system) design
  – High-level partitioning of a software system into separate modules *(components)*
  – Focus on the interactions among parts *(connections)*
  – Focus on structural properties *(architecture)*
    » “How does it all fit together?”
◆ Module design
  – Detailed design of a component
  – Focus on the internals of a component
  – Focus on computational properties
    » “How does it work?”
### Comparison to Programming (of Modules)

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>• interactions <strong>among</strong> parts</td>
<td>• implementations <strong>of</strong> parts</td>
</tr>
<tr>
<td>• structural properties</td>
<td>• computational properties</td>
</tr>
<tr>
<td>• system-level performance</td>
<td>• algorithmic performance</td>
</tr>
<tr>
<td>• outside module boundary</td>
<td>• inside module boundary</td>
</tr>
</tbody>
</table>
Software Architecture Topics

- Essential elements
- Repertoire of architectural styles
- Choosing and/or modifying a style
- Designing within a style
- Architecture in support of application families
Software Architecture: Essentials

◆ Components
  – What are the elements?
  – What aspects of the requirements do they correspond to? Where did they come from?
  – Examples: filters, databases, objects, ADTs

◆ Connections
  – How do components communicate?
  – Examples: procedure calls, messages, pipes, event broadcast

◆ Topology
  – How are the components and connections organized topologically?

◆ Constraints (including constraints on change)
We Can Do Anything…
...But Style Has Proven to Help

- Architectural styles restrict the way in which components can be connected
  - Prescribe patterns of interaction
  - Promote fundamental principles
    » Rigor, separation of concerns, anticipation of change, generality, incrementality
    » Low coupling
    » High cohesion
- Architectural styles are based on success stories
  - For many years most compilers were built as “pipe-and-filter”
  - Almost all network protocols are built as “layers”
  - Many business systems are built as “three-tier” systems
Common Architectural Idioms

◆ Data flow systems
  – (1) Batch sequential
  – (2) pipe-and-filter
◆ (3) Data and/or service-centric systems: the Client-Server style
  – The (pre-1994) WWW
  – Database servers
◆ (4) Hierarchical systems
  – Main program and subroutines;
◆ (5) Data abstraction/OO systems
◆ (6) Peer-to-Peer
◆ (7) Layered systems
◆ (8) Interpreters
◆ (9) Implicit invocation (event-based)
◆ (10) Three-level architectures

Note: not all of these are of equal value, current use, or intellectual depth

Many of the following slides are from David Garlan, Mary Shaw, and Jose Galmes: Experience with a Course on Architectures for Software Systems, Part II: Educational Materials
Style 1: Batch Sequential

Data Flow: Validate → Sort → Update → Report

Data transformation: tape → Validate, tape → Sort, tape → Update, tape → Report

University of California, Irvine
Batch Sequential

- Components
  - components are independent programs
  - each component runs to completion before next step starts
- Connections
  - Data transmitted as a whole between components
- Topology
  - Connectors define data flow graph
- Typical application: classical data processing
Style 2: Pipe and filter

Data Flow as ASCII Stream

Computation Filter

ls → grep -e exp → sort → lpr
Pipe and filter

- **Components**
  - Like batch sequential, but components (filters) incrementally transform some amount of the data at their inputs to data at outputs
  - Little local context used in processing input stream
  - No state preserved between instantiations

- **Connections**
  - Pipes move data from a filter output to a filter input
  - Data is a stream of ASCII characters

- **Topology**
  - Connectors define data flow graph

- Pipes and filters run (non-deterministically) until no more computation possible

- Typical applications: many Unix applications
Style 3: Client-Server

Connections are remote procedure calls or remote method invocations

University of California, Irvine
Client-Server Systems

- Components
  - 2 distinguished kinds
    » Clients: towards the user; little persistent state; active (request services)
    » Servers: “in the back office”; maintains persistent state and offers services; passive

- Connectors
  - Remote procedure calls or network protocols

- Topology
  - Clients surround the server
The pre-1994 WWW as a Client-Server Architecture

- Browsers are clients
- Web servers maintain state
- Connections by HTTP/1.0 protocol
Database Centered Systems

- Components
  - Central data repository
  - Schema (how the data is organized) designed for application
  - Independent operators
    » Operations on database implemented independently, one per transaction type
    » interact with database by queries and updates
- Connections
  - Transaction stream drives operation
  - Operations selected on basis of transaction type
  - May be direct access to data; may be encapsulated
Style 4: Hierarchy: Main Program and Subroutines

Connections are function or method calls
Main Program and Subroutines

- Components
  - Computational elements as provided by programming language
  - Typically single thread
- Connections
  - Call/return as provided by programming language
  - Shared memory
- Topology
  - Hierarchical decomposition as provided by language
  - Interaction topologies can vary arbitrarily
Style 5: Data Abstraction/OO Systems

Data Abstraction or Object-Oriented

Manager ADT
Proc call

obj is a manager
op is an invocation

Software Architectures
Data Abstraction/OO Systems

- **Components**
  - Components maintain encapsulated state, with public interface
  - Typically single threaded, though not logical

- **Connections**
  - Procedure calls ("method invocations") between components
  - Various degrees of polymorphism and dynamic binding
  - Shared memory a common assumption

- **Topology**
  - Components may share data and interface functions through inheritance hierarchies
  - Interaction topologies can vary arbitrarily
Style 6: Peer-to-Peer

Connections are remote procedure calls or remote method invocations
Peer-to-Peer Architectures

- **Components**
  - Autonomous
  - Act as both clients and servers

- **Connectors**
  - Asynchronous and synchronous message passing ("remote procedure calls")
  - By protocols atop TCP/IP
  - No shared memory (except as an optimization when the configuration allows)

- **Topology**
  - Interaction topologies can vary arbitrarily and dynamically
Connections are function or method calls + “something in between”
Layered Systems,  Take 2

Inter-level interfaces usually procedure calls

Users

Core Level

Basic Utility

Useful Systems
Layered Systems

- Components
  - Each layer provides a set of services

- Connections
  - Typically procedure calls
  - A layer typically hides the interfaces of all layers below, but others use "translucent" layers

- Topology
  - Nested

- Typical applications: support for portability, systems with many variations ("core features" v. extended capabilities)
Style 8: Implicit Invocation

Connections are events on the software bus

University of California, Irvine
Style 9: Interpreters

- Program Being Interpreted
- State of Program Being Interpreted
- Simulated Interpretation Engine
- Output of P
  - Data upon which P is to be executed
  - Output of interpreter
    (e.g. error messages)
- Internal Interpreter
  State (the interpreter’s local variables)
Interpreters

- **Components**
  - Execution engine simulated in software (with its internal data)
  - Program being interpreted
  - State of program being interpreted

- **Connections**
  - Program being interpreted determines sequence of actions by interpreter
  - Shared memory

- **Topology**

- **Typical applications:** end-user customization; dynamically changing set of capabilities (e.g. HotJava)
Style 10: “Three Level Architectures”

- User interface
- Application Logic
- Database (server)
Where do Architectures and Components Come From?

- **Architectures**: typically driven by kind of application
  - Often possible to solve one problem many different ways
- **Components**: many design strategies
  - **ICS 52 component strategy**:
    » Component design by information hiding
    » Designing systems for ease of extension and contraction
    » An OO design approach
  - **Rationale**: design systems that have a long, useful lifetime
Choosing the Right Style

- Ask questions on whether a certain style makes sense
  - The Internet as a blackboard
    » Does that scale?
  - Stock exchange as a layers
    » How to deal with the continuous change?
  - Math as hierarchy
    » How to properly call different modules for different functions?

- Draw a picture of the major entities
- Look for the natural paradigm
- Look for what “feels right”