CSE 503: Software Engineering

Software Architecture

Jonathan Aldrich
University of Washington
Department of Computer Science and Engineering
Winter 2002

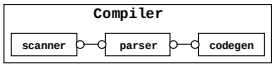
Programming in the Large vs. Programming in the Small

- Large systems bring different challenges
- What problems have you experienced?
  - Where/how do I extend the system?
  - What invariants hold of a large data structure?
  - What are a module's clients/what does it use?
  - Tight coupling
  - Large interfaces

Programming in the Large vs. Programming in the Small

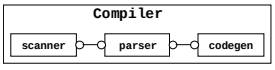
- Large systems bring different challenges
- How do you deal with them?
  - Imitate examples
  - Have someone explain/draw a picture
  - Trace through code
  - Grep or other tools
  - Read documentation

Software Architecture



- The highest level of design
  - The gross organization of a software system
  - Issues: decomposition, control flow, communication, concurrency, distribution
- A set of *components*, *connections* between the components, and *constraints* on how they interact

Benefits of Architecture

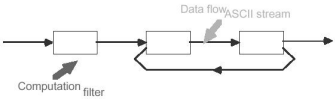


- Specification of high-level system design
  - Program understanding
  - Analysis
  - Language and tool support
- Taxonomy of system design
  - Advantages/disadvantages
  - Capturing design experience
  - Relationship between systems

Architectural Styles

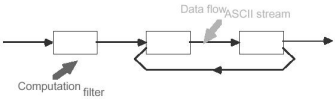
- System-level design patterns
  - Client-server or 3-tier
  - Layered system
  - Pipeline architecture
- Represent design knowledge
  - Vocabulary of concepts
  - Constraints on implementation
  - Benefits/drawbacks of alternatives

Pipe and Filter Style



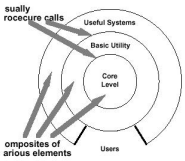
- Filters process data, streams connect filters
  - No shared state
  - No knowledge of other filters
- Design choices
  - Linear (pipeline) vs. connection graph
  - Incremental vs. batch processing
- Canonical examples: compilers, unix tools
  - Others?

Pipe and Filter Style



- Advantages
  - Easy to change: add/remove/replace filters
  - Filters need only agree on data (e.g., XML)
  - Inherent scalability and concurrency
  - Analysis for throughput, deadlock
- Drawbacks?

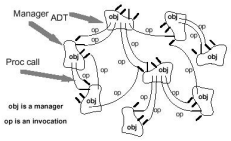
Layered Style



- As discussed before!
- Invariant
  - Each layer uses only the layer (or layers) below
- Advantages
  - Easy to add new functionality in a new layer
  - Limited dependencies ease change
  - Can swap in different layer implementations
- Drawbacks?

Object-Oriented/ADT Style

- A set of communicating objects
  - Each is responsible for its (hidden) representation

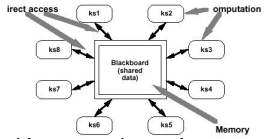


- What do you think of this style?

Implicit Invocation Style

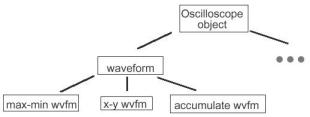
- As discussed before!
- Invariant
  - Announcers don't know about listeners
- Advantages
  - Easy to add/remove/replace components
- Drawbacks
  - Hard to reason about system

Repository Style



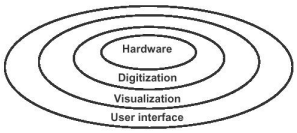
- Data stored in a central repository
  - Independent components operate on the store
    - Pure form: may not communicate in any other way
  - Components may be triggered by data
- Advantages?
- Drawbacks?

Oscilloscope Case Study



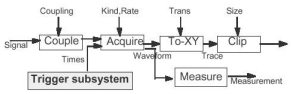
- Initial OO architecture
  - Showed data structures
  - Little organizational guidance

Oscilloscope Case Study



- Second architecture: layers
  - Partitions functionality
  - But, UI has to touch all layers

Oscilloscope Case Study



- Final architecture: modified pipe & filter
  - Matches data flow intuition
  - UI has control connection to each filter
  - Many pipes don't copy data

Architectural Analysis

- Example: Wright
  - An Architecture Description Language (ADL)
- Models computation and communication
  - Finite state machines with event transitions
  - Use CSP notation and semantics
- Analysis
  - Find deadlock
  - See if components are compatible

Recap

- Architectural Styles
  - Vocabulary for design
  - Encapsulate design knowledge
  - Constrain design
  - Have specific advantages and disadvantages
- Analysis can enable more effective design
- But what about the implementation?