


Center
For
Software
Engineering


Next week: Guest Lecture by Eric Dashofy (UCI)

Attendance is mandatory (DEN
students are not required to come
to campus)



Center
For
Software
Engineering

Architectural Styles




Center
For
Software
Engineering

Definitions of Architectural Style

- Recurring organizational patterns & idioms
 - Established, shared understanding of common design forms
 - Mark of mature engineering field.
 - Shaw & Garlan
- Abstraction of recurring composition & interaction characteristics in a set of architectures
 - Taylor

3




Center
For
Software
Engineering

Definitions of Architectural Style (cont.)

- Styles are key design idioms that enable
 - Exploitation of suitable structural & evolution patterns
 - Facilitate component, connector, & process reuse
 - Medvidovic


4



Categories of Styles

- Idioms & patterns
 - Deal with global organizational structures
 - Possibly application-domain independent
 - To be discussed today
- Pipe and filter
- Client-server
- Blackboard
- Layered
- Reference models
 - Specific configurations for certain application areas
 - May be effective outside their initial domains
 - Discussed previously
- Canonical compiler architecture
- Other DSSAs


5



Basic Properties of Styles

- A vocabulary of design elements
 - Component and connector types
 - e.g., pipes, filters, objects, servers
- A set of configuration rules
 - Topological constraints that determine allowed compositions of elements
 - e.g., a component may be connected to at most two other components
- A semantic interpretation
 - Compositions of design elements have well-defined meanings
- Possible analyses of systems built in a style
 - Code generation is a special kind of analysis


6



Benefits of Using Styles

- Design reuse
 - Well-understood solutions applied to new problems
- Code reuse
 - Shared implementations of invariant aspects of a style
- Understandability of system organization
 - A phrase such as "client-server" conveys a lot of information
- Interoperability
 - Supported by style standardization
 - e.g., CORBA, JavaBeans
- Style-specific analyses
 - Enabled by the constrained design space
- Visualizations
 - Style-specific depictions matching engineers' mental models

7



Style Analysis Dimensions

- What is the design vocabulary?
 - Component and connector types
- What are the allowable structural patterns?
- What is the underlying computational model?
- What are the essential invariants of the style?
- What are common examples of its use?
- What are the (dis)advantages of using the style?
- What are the style's specializations?

8

Center For Software Engineering

Some Common Architectural Styles

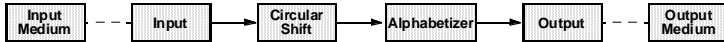
- ❑ “Basic” styles
 - Pipe and filter
 - Object-oriented
 - Layered
 - Blackboard
 - State transition
 - Client-server
 - Many flavors [fielding99]
 - Peer-to-peer
 - Event-based (a.k.a. Implicit invocation)
 - Push-based
- ❑ “Derived” styles
 - GenVoca
 - C2

9

Center For Software Engineering

Pipe and Filter Style

- ❑ Components are filters
 - Transform input data streams into output data streams
 - Possibly incremental production of output
- ❑ Connectors are pipes
 - Conduits for data streams



```

graph LR
  IM[Input Medium] -.-> I[Input]
  I --> CS[Circular Shift]
  CS --> A[Alphabetizer]
  A --> O[Output]
  O -.-> OM[Output Medium]
  
```

- ❑ Style invariants
 - Filters are independent (no shared state)
 - Filter has no knowledge of up- and down-stream filters
- ❑ Examples

UNIX shell	signal processing
Distributed systems	parallel programming

10

Center For Software Engineering

Pipe and Filter (cont.)

- ❑ Variations
 - Pipelines — linear sequences of filters
 - Bounded pipes — limited amount of data on a pipe
 - Typed pipes — data strongly typed
 - Batch sequential — data streams are not incremental
- ❑ Advantages
 - $\text{System.Behavior} = \sum \text{Component.Behavior}$
 - Filter addition, replacement, and reuse
 - Possible to hook any two filters together
 - Certain analyses
- ❑ Throughput, latency, deadlock
 - Concurrent execution
- ❑ Disadvantages
 - Batch organization of processing
 - Interactive applications
 - Lowest common denominator on data transmission

11

Center For Software Engineering

Object-Oriented Style


- ❑ Components are objects
 - Data and associated operations
- ❑ Connectors are messages and method invocations
- ❑ Style invariants
 - Objects are responsible for their internal representation integrity
 - Internal representation is hidden from other objects
- ❑ Advantages
 - “Infinite malleability” of object internals
 - System decomposition into sets of interacting agents
- ❑ Disadvantages
 - Objects must know identities of servers
 - Side effects in object method invocations

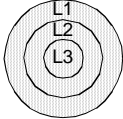
12

Center For Software Engineering

Layered Style

- ❑ Hierarchical system organization
 - “Multi-level client-server”
 - Each layer exports an “API” to be used by above layers
- ❑ Each layer acts as a
 - Server: service provider to layers “above”
 - Client: service consumer from layers “below”
- ❑ Connectors are protocols of layer interaction
- ❑ Example: operating systems
- ❑ *Virtual machine* style results from fully opaque layers





13

Center For Software Engineering

Layered Style (cont.)

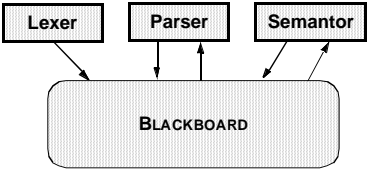
- ❑ Advantages
 - Increasing abstraction levels
 - Evolvability
- ❑ Changes in a layer affect at most the adjacent two layers
 - Reuse
- ❑ Different implementations of layer are allowed as long as interface is preserved
- ❑ Standardized layer interfaces (e.g., X window system protocols)
- ❑ Disadvantages
 - Not universally applicable
 - Performance
- ❑ Layers may have to be skipped
 - Determining the correct abstraction level

14

Center For Software Engineering

Blackboard Style

- ❑ Two kinds of components
 - Central data structure — blackboard
 - Components operating on the blackboard
- ❑ System control is entirely driven by the blackboard state
- ❑ Examples
 - Typically used for AI systems
 - Integrated software environments (e.g., Interlisp)
 - Compiler architecture

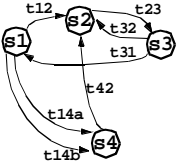


15


Center For Software Engineering

State-Transition Style

- ❑ Components represent (sets of) system states
- ❑ Connectors are (sets of) named state transitions
- ❑ Disadvantage
 - Even trivial systems have enormous state spaces
- ❑ Remedy
 - Abstract away states into coarser-grained components
 - e.g., StateCharts/StateMate




16

 Center For Software Engineering

Client-Server Style

- ❑ Instance of a more general style
 - Distributed systems
- ❑ Components are clients and servers
- ❑ Servers do not know number or identities of clients
- ❑ Clients know server's identity
- ❑ Connectors are RPC-based interaction protocols
- ❑ A number of different flavors of client-server


17

 Center For Software Engineering

Implicit Invocation Style

- ❑ Event announcement instead of method invocation
 - “Listeners” register interest in & associate methods with events
 - System invokes all registered methods implicitly
- ❑ Component interfaces are methods and events
- ❑ Two types of connectors
 - Invocation is either explicit or implicit in response to events
- ❑ Style invariants
 - “Announcers” are unaware of their events' effects, if any
 - No assumption about processing in response to events


18

 Center For Software Engineering

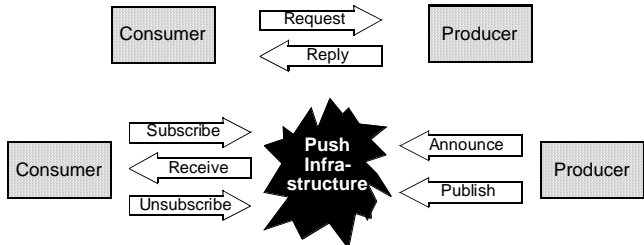
Implicit Invocation (cont.)

- ❑ Advantages
 - Component reuse
 - System evolution
 - Both at system construction-time & run-time
- ❑ Disadvantages
 - Counter-intuitive system structure
 - Components relinquish computation control to the system
 - No knowledge of what component(s) will respond to event
 - No knowledge of order of responses
 - Analysis via pre- & post-conditions is difficult

19

 Center For Software Engineering

Push-Based Style

- ❑ Distinguished from pull-based (e.g., the Web)
 
- ❑ Examples
 - employee information systems
 - maintenance manuals
 - stock ticker

20

Center For Software Engineering

Push-Based Style (cont.)

- ❑ “Components”
 - Producer
 - Receiver
 - Channel
 - Broadcaster
 - Transport system
 - Repeater, cache, proxy
 - Transparent to all other components
- ❑ Asymmetric communication model
 - Producers \neq Receivers
 - Typically fewer producers; but more consumers per producer than event-based style
- ❑ Relatively tight coupling between source and receiver via subscribed channels

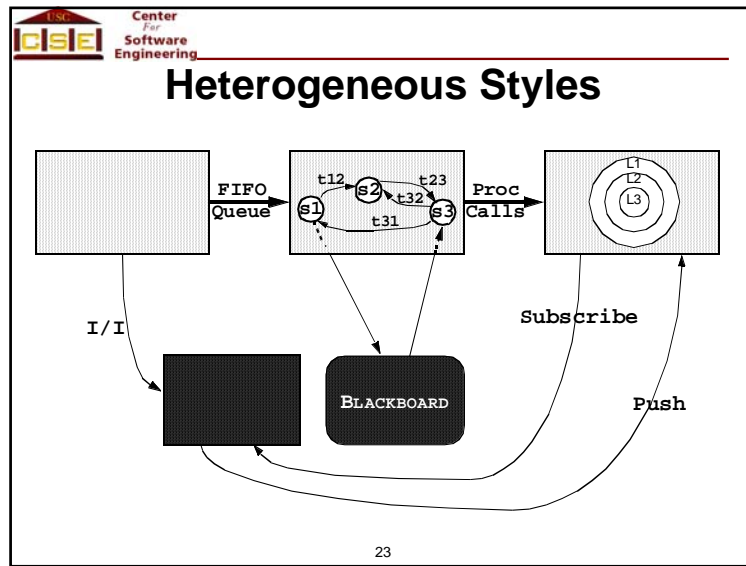
21

Center For Software Engineering

Push-Based Style (cont.)

- ❑ Advantages
 - Communication flexibility
 - Via repeaters, caches, proxies
 - Back-channels ~ peer-to-peer
- ❑ Disadvantages
 - Relatively tight coupling between producer & consumer via subscribed channels
 - Many system don't fit model

22




Center For Software Engineering

Observations

- ❑ Different styles result in
 - Different architectures
 - Architectures with greatly differing properties
- ❑ No style does fully specifies architecture
 - Single style can result in different architectures
 - Considerable room for
 - Individual judgment
 - Variations among architects
 - Different emphases
 - e.g. imposed by customer
- ❑ Style defines domain of discourse
 - About problem (domain)
 - About resulting system
 - Different architectures lead architects to ask different questions

24



Open Issues

- Use of styles is generally ad-hoc
- Difficult to
 - Delimit system aspects that can/should be specified by a style
 - Compare styles based on their properties
 - Relate systems developed in different styles
 - Select appropriate style(s) for given problem
- Unclear how existing styles can be most effectively combined to produce new style
- What is exact relation between domains & styles?

25