Architecture

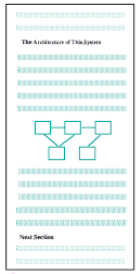
Mary Shaw and David Garlan

Taken from Tutorial on Architectures for Software Systems
http://spoke.compose.cs.cmu.edu/shaweb/p/pubs.htm

Part I: Architectural Design

I. Architectural design: planning system structure
   > What is software architecture?
   > What is its relation to other aspects of design?
II. The variety of software architectures: common styles
III. Deciding which architecture to use
   --- break ---
IV. Concrete examples
V. Dealing with mismatched parts
VI. Topics of possible future interest

Typical Descriptions of Software Architectures



- Descriptions of software systems often include a section on "the architecture of this system"
- Usually informal prose plus box-and-line diagram
- Lots of appeal to intuition
- Little precision, rarely formal

The Role of Software Architecture

User view of problem	User Model
Software view of problem	Requirement
Components & connections	***Architecture***
Algorithms & data strs	**Code**
Data layouts, memory maps	Executable

Architectural Design Task

Different issues for architecture & programs

Architecture	*Programs*
interactions among parts	implementations of parts
structural properties	computational properties
declarative	operational
mostly static	mostly dynamic
system-level performance	algorithmic performance
outside module boundary	inside module boundary
composition of subsystems	copy code or call libraries

Typical Descriptions of Software Architectures

> "Camelot is based on the client-server model and uses remote procedure calls both locally and remotely to provide communication among applications and servers." [Spector 87]

> "We have chosen a distributed, object-oriented approach to managing information." [Linton 87]

> "The easiest way to make the canonical sequential compiler into a concurrent compiler is to pipeline the execution of the compiler phases over a number of processors." [Seshadri 88]

> "The ARC network [follows] the general network architecture specified by the ISO in the Open Systems Interconnection Reference Model." [Paulk 85]

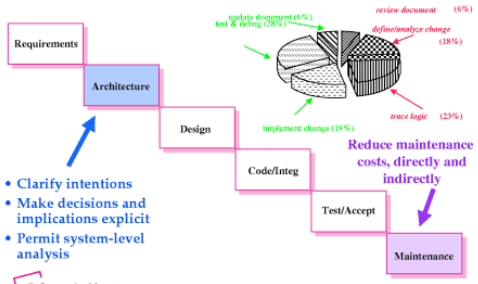
Observations about Designers

- They freely use informal patterns (idioms)
  - > Very informal, imprecise semantics
  - > Diagrams as well as prose, but no uniform rules
  - > Communication takes place anyhow
- Their vocabulary uses system-level abstractions
  - > Overall organization (styles)
  - > Kinds of components and interactions among them
- They compose systems from subsystems
  - > Tend to think about system structure statically
  - > Often select organization by default, not by design

Expected Benefits

- Clarify design intentions
  - > Intended architecture is often lost. It's mostly informal, it's hard to communicate anyhow.
- Provide basis for analysis in design
  - > Engineering design entails performance prediction and design tuning. Routine practice
- Improve maintenance
  - > Over half of maintenance effort goes into figuring out just what's there
- Provide good questions
  - > Even without formal methods, explicit attention to experience yields guidance about sensitive points

Anticipated Benefits



Part II: The Conceptual Vocabulary of Styles

I. Architectural design: planning system structure
II. The variety of software architectures: common styles
  > Components and connectors
  > Style as design rules
  > Lots of examples
III. Deciding which architecture to use
  --- break ---
IV. Concrete examples
V. Dealing with mismatched parts
VI. Topics of possible future interest

Common Architectural Idioms

1. Data flow systems
  Batch sequential — Pipes and filters
  Control loops
2. Call-and-return systems
  Main program & subroutines — Object-oriented systems
3. Independent components
  Communicating processes — Event systems
4. Data-centered systems (repositories)
  Databases — Blackboards
5. Virtual machines
  Interpreters — Rule-based systems
  Hierarchical layers
... and more ...

Group 1: Dataflow Architectures

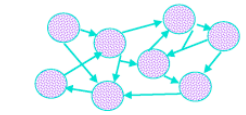
• Batch sequential
  > Historically dominated by database updates
• Pipes and filters
  > Filters connected in a dataflow graph
• Others
  > Pipelines
  > Control loops (Closed-loop control)

Data Flow Systems

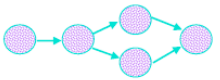
- A data flow system is one in which
  - > availability of data controls computation
  - > the structure of the design is dominated by orderly motion of data from component to component
  - > the pattern of data flow is explicit
- In a pure data flow system, there is no other interaction between processes

Software Architectures

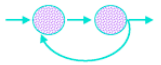
Kinds of Data Flow Systems



In general, data can flow in arbitrary patterns



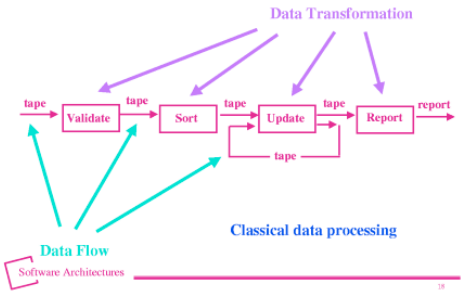
Here we are primarily interested in nearly- linear data flow systems,



or in very simple, highly constrained cyclic structures

Software Architectures

Batch Sequential



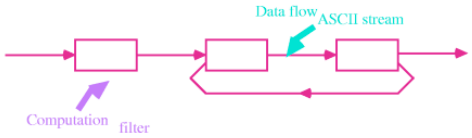
Software Architectures

Batch Sequential: Model

- Processing steps are independent programs
- Each step runs to completion before next step starts
- Data transmitted as a whole between steps
- Typical applications:
  - > classical data processing
  - > program development

Software Architectures

Pipes and Filters



Software Architectures

Pipes and Filters: Model

- Filter
  - > Incrementally transform some amount of the data at inputs to data at outputs
    - » Stream-to-stream transformations
  - > Use little local context in processing stream
  - > Preserve no state between instantiations
- Pipe
  - > Move data from a filter output to a filter input
  - > Pipes form data transmission graphs
- Overall Computation
  - > Run pipes and filters (non-deterministically) until no more computations are possible.

Software Architectures

Batch Sequential vs Pipe & Filter (UNIX

Both

Decompose task into fixed sequence of computations
Interact only through data passed from one to another

Batch Sequential	Pipe/Filter
Coarse-grained, total	Fine-grained, incremental
High latency (real-timeput ok	Processing localized in input
No concurrency	Feedback loops possible
Non-interactive	Often interactive, awkwardly

Software Architectures

Group 2: Call-and-Return Architectures

- Main program and subroutines
  - > Classical functional decomposition
- Object-oriented (abstract data types)
  - > Information hiding, especially hiding of representations
- Layered hierarchies
  - > Decompositions in which each element interacts only with adjacent elements
- Other
  - > Client-server systems
  - > Remote procedure calls

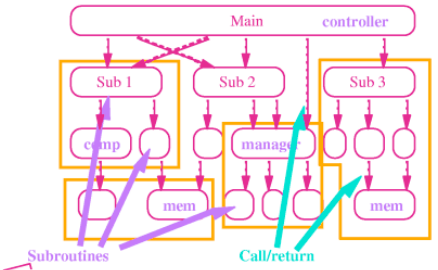
Software Architectures

Control Flow vs Data Flow

- Control Flow
  - > Dominant question is how locus of control moves through the program
  - > Data may accompany the control but is not dominant
  - > Reasoning is about order of computation
- Data Flow
  - > Dominant question is how data moves through a collection of (atomic) computations
  - > As data moves, control is activated
  - > Reasoning is about data availability, transformation, latency

Software Architectures

Main Program/Subroutine Pattern



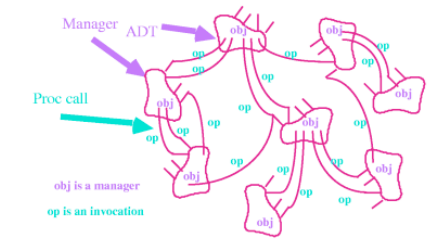
Software Architectures

Main Program and Subroutines

- Hierarchical decomposition:
  - > Based on definition-use relationship
- Single thread of control:
  - > Supported directly by programming languages
- Subsystem structure implicit:
  - > Subroutines typically aggregated into modules
- Hierarchical reasoning:
  - > Correctness of a subroutine depends on the correctness of the subroutines it calls

Software Architectures

Data Abstraction or Object-Oriented



Software Architectures

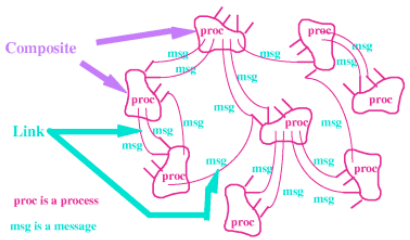
Object Architectures: Model

- Encapsulation:
  - > Restrict access to certain information
- Inheritance:
  - > Share one definition of shared functionality
- Dynamic binding:
  - > Determine actual operation to call at runtime
- Management of many objects:
  - > Provide structure on large set of definitions
- Reuse and maintenance:
  - > Exploit encapsulation and locality

Group 3: Independent Components

- Communicating processes
  - > Passing messages among known participants
- Event systems
  - > Implicit invocation with unknown participants
- Others
  - > Multicast messages with dynamic binding
  - > Interrupt-driven processes

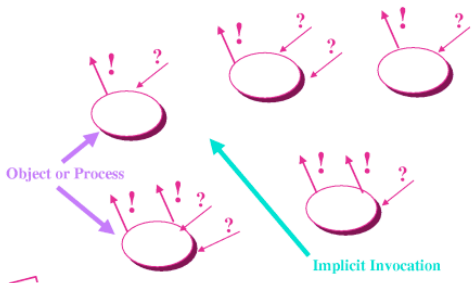
Communicating Processes



Communicating Processes

- Components: independent processes
  - > typically implemented as separate tasks
- Connectors: message passing
  - > point-to-point
  - > asynchronous and synchronous
  - > RPC and other protocols can be layered on top

Event Systems



Software Architectures

Event Systems: Model

- Components: objects or processes
  - > Interface defines a set of incoming procedure calls
  - > Interface also defines a set of outgoing events
- Connections: event-procedure bindings
  - > procedures are registered with events
  - > components communicate by announcing events at "appropriate" times
  - > when an event is announced the associated procedures are (implicitly) invoked
  - > order of invocation is non-deterministic

Software Architectures

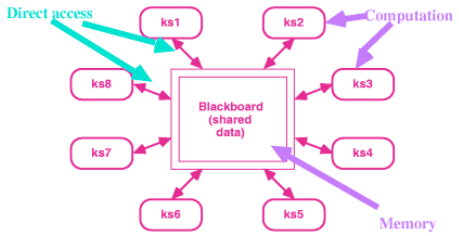
Group 5: Data-Centered Systems (Repositories)

- Transactional databases
  - > Large central data store
  - > Operation order determined by input stream
- Blackboards
  - > Central shared representation tuned to application
  - > "Knowledge sources" execute opportunistically
- Others
  - > CASE tools
  - > Integrated design systems

(mz: were referred to as group 4 earlier)

Software Architectures

Repository (Blackboard)



Software Architectures

The Blackboard Model

- Knowledge Sources
  - > World and domain knowledge partitioned into separate, independent computations
  - > Respond to changes in blackboard
- Blackboard Data Structure
  - > Entire state of problem solution
  - > Hierarchical, nonhomogeneous
  - > Only means by which knowledge sources interact to yield solution
- Control
  - > In model, knowledge sources self-activating

Group 4: Virtual machines

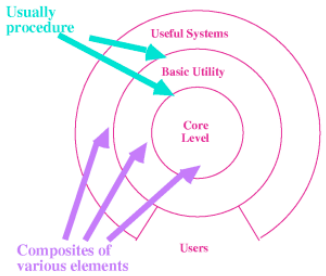
- Interpreters
  - > Create virtual machine when the one you want isn't there
- Rule-based systems
  - > Specific kind of interpreter
- Other
  - > Syntactic "shells"
  - > Command language processors

(mz: were referred to as group 4 earlier)

Interpreter: Model

- Execution engine simulated in software
- Data:
  - > representation of program being interpreted
  - > data (program state) of program being interpreted
  - > internal state of interpreter
- Control resides in "execution cycle" of interpreter
  - > but simulated control flow in interpreted program resides in internal interpreter state
- Syntax-driven design

Layered Pattern



Layered Pattern: Model

- Each layer provides certain facilities
  - > hides part of lower layer
  - > provides well-defined interfaces
- Serves various functions
  - > kernels: provide core capability, often as set of procedures
  - > shells, virtual machines: support for portability
- Various scoping regimes
  - > Opaque versus translucent layers

Comparison of System Patterns

System Model	Components	Connections	Control Struct
Pipeline			
stream -> stream	filters (local processing)	data flow ASCII streams	data flow
Data abstraction (object-oriented)			
localized state maint	servers (ADTs, objs)	procedure call	decentralized, single thread
Events			
implicit invocation	independent components	blind announce	loose coupling
Interpreter			
virtual machine	state mach, two memories	fetch, store	input-driven
Repository			
central database	1 memory N processes	direct access or proc call	internal or external

Common Architectural Idioms

1. Data flow systems
   - Batch sequential
   - Pipes and filters
   - Control loops
2. Call-and-return systems
   - Main program & subroutines
   - Object-oriented systems
3. Independent components
   - Communicating processes
   - Event systems
4. Data-centered systems (repositories)
   - Databases
   - Blackboards
5. Virtual machines
   - Interpreters
   - Rule-based systems
   - Hierarchical layers

... and more ...

Common Architectural Idioms

1. Data flow systems
   - Batch sequential
   - Pipes and filters
   - Control loops
2. Call-and-return systems
   - Main program & subroutines
   - Object-oriented systems
3. Independent components
   - Communicating processes
   - Event systems
4. Data-centered systems (repositories)
   - Databases
   - Blackboards
5. Virtual machines
   - Interpreters
   - Rule-based systems
   - Hierarchical layers

... and more ...

Software Architectures
43

The Punch Line

- Interactions matter -- how a component must interact, not just what it computes
- Packaging matters -- compatibility among components, not just functionality
- Distinctions matter -- there are different types of components, interactions, styles
- Decisions matter -- different kinds of problems require different solutions

Provide a solid basis in models, notations, and tools to support developers' intuitions

Part III: Design Decisions and Processes

I. Architectural design: planning system structure
II. The variety of software architectures: common styles
III. Deciding which architecture to use
  > Problem frames and solution frames
  > Design spaces for considering tradeoffs
  > Rules of thumb for selecting architectures
  > Example
  > Processes for architectural design and review

--- break ---

IV. Concrete examples
V. Dealing with mismatched parts
VI. Topics of possible future interest