**Software Architectural Design: Introduction**

What is Architecture?

Current Practice in Software Architecture

A Model of Software Architecture

Why Software Architecture?

---

**What is Architecture?**

*the underlying structure of things -
  buildings, communication networks, spacecraft, computer, software, nature*

- Civil engineering
  - Customer engineer gets customer requirements
    - **functional units:**
      - 3 bedrooms,
      - 2+1/2 bathrooms
      - 1 living & 1 dining rooms
      - 2-car garage
      - kitchen
      - backyard
    - **other considerations:**
      - cost
      - esthetics
      - workmanship
      - neighborhood
      - maintainability
      - economics

- Architect starts thinking about architectural styles
  - **architectural styles:**
    - Victorian, Duplex, Condominium, Townhouse, Cathedral, Pyramidal, ...
  - floor plans & elevations for functional units
  - **other considerations:**
    - immense amount of details not present about various detailed design considerations such as electrical wiring, plumbing, heating, etc.
What is Architecture?

the underlying structure of things -
buildings, communication networks, spacecraft, computer, software, nature

Civil engineering

① Designers/Contractors think about detailed design considerations

- electrical wiring, plumbing, heating, air-conditioning, carpeting, etc.

② Sub-contractors/Construction Workers:

- electricians, plumbers, furnace installers, carpenters, locksmith, brick layers, bathtub technicians, etc.

Reading Assignment: Chapter 1

Current Practice in Software Architecture

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

client-server model -> what? -> clients (applications) & server(s) as components
RPCs both locally and remotely -> what -> communication/interaction mechanism

But,

Why client-server model?
- distributed data?  distributed processing?  cooperative processing?

What’s a client like?
- a terminal emulator?  + a domain-specific application?

What’s a server like?
- a file server?  a db server?  a transaction server?  a CGI server?
- a groupware?

Why RPCs?
- why not sockets?  why not MOMs?  why not events?

What’s communicated?

Any constraint?
- like passive Web browser?  like client-centric Java?
- like server-centric CGI?  like CORBA?  like OLE2/IDCOM?
- uni-/bi-directional communication?  multi-paradigm?  multi-platform?

Good software developers have often adopted one or several architectural patterns but informally and often only implicitly.
Current Practice in Software Architecture

Abstraction layering and system decomposition provide the appearance of system uniformity to clients, yet allow Helix [distributed file system] to accommodate a diversity of autonomous devices. The architecture encourages a client-server model for the structuring of applications.

We have chosen a distributed, object-oriented approach to managing information. The architecture encourages a client-server model for the structuring of applications.


What's a client like?

What's a server like?

What's the communication mechanism?

Any constraint?

Good software developers have often adopted one or several architectural patterns but informally and often only implicitly.

Observations

Software architectures are indeed used, very often but without even knowing it. It's natural to use software architectures!

Carries some, and more often than not a lot of, information. Care must be exercised!

No explicit description of the structure. No clear basis for communication or reasoning!

Good software developers have often adopted one or several architectural patterns but informally and often only implicitly.
A Model of Software Architecture

Software architecture:

- **elements (components/parts):**
  - from which systems are built
  - e.g., process, data, object, agent

- **interactions (connections/ connectors/glues/relationships):**
  - between the elements
  - e.g., PCs, RPCs, MOMs, events

- **patterns:**
  - describe layout of elements and interactions, guiding their composition
  - e.g., # of elements, # of connectors, order, topology, directionality

- **constraints:**
  - on the patterns (i.e., on components, connectors, layout)
  - e.g., temporal, cardinality, concurrency, (a)synchronous, etc.

- **styles:**
  - abstraction of architectural components from various specific architectures.
  - (Sometimes interchangeably used with patterns)
  - e.g., Unix OS, OSI protocol layer, Onion ring IS structure -> layering

- **rationale:**
  - describe why the particular architecture is chosen

---

Example: Sequential Compiler

<table>
<thead>
<tr>
<th>elements</th>
<th>interactions</th>
<th>patterns:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexer</td>
<td>source code (characters)</td>
<td>connector (stream of data)</td>
</tr>
<tr>
<td></td>
<td>a+ x * (1-1) +7</td>
<td>process</td>
</tr>
<tr>
<td>Parser</td>
<td>tokens (name table)</td>
<td>(stream of data)</td>
</tr>
<tr>
<td></td>
<td>a plus x mult iParen 1 minus 1 rParen plus 7</td>
<td>process</td>
</tr>
<tr>
<td>Semantic Analyzer</td>
<td>phrases (name table &amp; abstract syntax tree)</td>
<td>process</td>
</tr>
<tr>
<td></td>
<td>a plus [x mult [1 minus 1]] plus 7</td>
<td></td>
</tr>
<tr>
<td>Optimizer</td>
<td>correlated phrases</td>
<td>process</td>
</tr>
<tr>
<td></td>
<td>(name table &amp; abstract syntax graph)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a plus [x mult [1 minus 1]] plus 7</td>
<td></td>
</tr>
<tr>
<td>Coder</td>
<td>(annotated) correlated phrases</td>
<td>process</td>
</tr>
<tr>
<td></td>
<td>(name table &amp; annotated abstract syntax graph)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a plus 7</td>
<td>process</td>
</tr>
<tr>
<td></td>
<td>load a; load 7; add</td>
<td></td>
</tr>
</tbody>
</table>
A Model of Software Architecture

Example: Sequential Compiler

<table>
<thead>
<tr>
<th>elements</th>
<th>interactions</th>
<th>patterns:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexer</td>
<td>source code (characters)</td>
<td>connector (stream of data)</td>
</tr>
<tr>
<td></td>
<td>a+ x (rParen 1 minus 1 rParen plus 7</td>
<td>(stream of data)</td>
</tr>
<tr>
<td>Parser</td>
<td>tokens (name table)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a plus [x mult (1 minus 1)] plus 7</td>
<td></td>
</tr>
<tr>
<td>Semantic Analyzer</td>
<td>correlated phrases (name table &amp; abstract syntax graph)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a plus [x mult (1 minus 1)] plus 7</td>
<td></td>
</tr>
<tr>
<td>Optimizer</td>
<td>(annotated) correlated phrases (name table &amp; annotated abstract syntax graph)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a plus 7</td>
<td></td>
</tr>
<tr>
<td>Coder</td>
<td>load a; load 7; add</td>
<td></td>
</tr>
</tbody>
</table>

(style): pipe&filter
- each element does a local transformation to the input and produces output
- each glue serves as a conduit for the data stream, transmitting outputs of one process to inputs of another

(constraint): processes do not share state with other processes
- processes do not know the identity of their upstream and downstream processes
- Independent processes (other than stream availability)

(rationale): simplicity, process independence

Lawrence Chung

A Model of Software Architecture

Points to ponder about:

- What are disadvantages (& other advantages) of this architecture?
  - Time, Space, Reusability, Adaptability, etc.
- What alternative architectures are possible?
  - Lexer + Parser
    - 2 Semantic Analyzers (forward reference)
    - Shared data + sequential
    - No Optimizer
    - Concurrent compiler (semantic analyzer || optimizer || coder)
- What are some other instances of this style?
  - Unix command processing: e.g., ls|sort|pr|lpr

Lawrence Chung
Common Architectural Styles

- **Dataflow systems** [topic 5: Data Flow]
  - Batch sequential
  - Pipe & Filter

- **Call-and-return systems**
  - Main program & subroutine [topic 4: Modular Decomposition Issues]
  - OO systems [topic 3: ADT]
  - Hierarchical layers [topic 5 & 6 & 10 - Data Flow & Repositories & Middleware]

- **Independent components**
  - Communicating processes [topic 11?: Processes]
  - Event systems [topic 4 & 7 - Modular Decomposition Issues & Events]

- **Virtual machines**
  - Interpreters
  - Rule-based systems

- **Data-centered systems** [topic 6: Repositories]
  - Databases
  - Hypertext systems
  - Blackboards

- **Process-control paradigms** [topic 8: Repositories]

---

Why Software Architecture?

- Abstract solution to conquer complexity
- functionality and performance (Non-functional requirements)
- divide and conquer

- A shared, semantically-rich vocabulary between SEers.
  - E.g., instanceOf (X, pipe & filter)
  - X is primarily for stream transformation
  - functional behavior of X can be derived compositionally from
  - the behaviors of the constituent filters
  - issues of system latency and throughput can be addressed
  - in relatively straightforward ways

- supports reuse
- facilitates (integration) testing
- parallel development
- system evolvability

... and many other conceptual reasons