Change *during* runtime?

- Critical systems require “continuous availability”
- Power grid, financial systems, ...
- Increasingly important in everyday systems
State of the Practice

- redundant and fault-tolerant hardware
- “hot pluggable” drives and memory
- binary code patching
- programming language facilities for dynamic loading, linking, and patching of code
- architectural styles and patterns that support adaptation
Making Comparisons

- Each approach has its place
- No one approach encompasses the others
- Clear benefits to enacting change at multiple levels of abstraction
- Use a framework for comparing and combining approaches

BASE Framework

With respect to a model of a system

**Behavior:** what aspects of behavior can be changed? How represented? Deployed?

**Asynchrony:** is execution suspended during adaptation? Partially?

**State:** how is the system’s state changed? Described? Types & instances changed?

**Execution context:** What changes are made to the VM executing the model? What about the stack? Registers?
Architectural Styles and Adaptation

- A simple message: if you want or need adaptable applications you can either:
  - Make no constraints on developers
    - ... and then work like crazy to try to obtain adaptation
  - Or, constrain development to make adaptation easier and predictable

- This should not be news: the message is *styles*

Two Definitions

A software system’s *architecture* is the set of *principal design decisions* made about the system.

An *architectural style* is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.
Pipe and Filter

- Independent filters (programs)
- Pipes route data streams between filters
- Filters may not maintain state between processing of chunks of data
- But no intrinsic support for runtime change
Dynamic P&F: Weaves

- Explicit buffering of messages in pipes
- Flow control mechanisms to regulate filter actions
- Jointly enables dynamic rewiring

Tool support for rewiring

Event Notification/C2

- Inter-component communication via explicit events
- Explicit connectors route events
- Components do not share state or VM

Evolution support through ADE tools
Dynamism in the WWW

- Clients come and go
- Servers come and go
- URLs come and go
- Datatypes (MIME types) come and go (albeit slowly)

And no one is in charge!

REST: REpresentational State Transfer

1. Information is a resource, named by a URL (locally determined)
2. Representation of a resource is a set of bytes, plus meta-data describing them (meta-data standard allows new types)
3. All interactions are context-free
4. Only a few operations available, and act in accordance with the other REST principles (barrier to adding new components low)
5. Idempotent operations and representation meta-data support caching (supporting dynamic optimization)
6. Presence of intermediaries is promoted
Peer-to-Peer Architectures

- Uncertainty intrinsic to applications
  - For example: Skype and BitTorrent

- Typical characteristics:
  - Replicated system state
  - Fault-focused protocols and design
  - Discovery

Service-Oriented Architectures

- Distributed computations that attempt to decouple elements
  - For example:
    - decouple RPC-binding by encoding function name and parameters in XML
    - “enterprise software buses” (event-notification)
Computational REST

- URLs name interpreters (virtual machine + URL-specific binding environment)
- Peers exchange closures and continuations for execution
- Content is a side-effect of computation (the value of a computation)

<table>
<thead>
<tr>
<th>Arch Style</th>
<th>Update Behavior</th>
<th>Asynchrony of change</th>
<th>Update State</th>
<th>Update Exec context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pub-Sub</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2/Event-based</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>REST</td>
<td>✓</td>
<td>✓</td>
<td>Data-State externalized</td>
<td>✓</td>
</tr>
<tr>
<td>CREST</td>
<td>✓</td>
<td>✓</td>
<td>All computation state externalized</td>
<td>✓</td>
</tr>
<tr>
<td>P2P</td>
<td>Peers replaced wholesale</td>
<td>✓</td>
<td>State replicated across peers</td>
<td>Peers replaced wholesale</td>
</tr>
</tbody>
</table>
Leverage Points

- Determine the elements subject to change and make them “first class”
- Make interaction controllable
- Provide for management of state
- ✅ Delay bindings
- ✅ Use explicit events/messages

Questions?