Seeking Closure in an Open World: a Behavioral Agent Approach to Configuration Management

Alva Couch, John Hart, Elizabeth G. Idhaw, Dominic Kallas
{couch,hart,greenlee,dkallas}@cs.tufts.edu
Goals

• Long range goal is portable validation: validate a configuration once, works the same everywhere(!).

• Short-range goals include developing:
  – an algebraic model of configuration management
  – Relationships between that model and established mathematical knowledge
  – examples of next-generation components and interfaces
Pressures

- So many parameters
- So little time
- Unclear semantics
- Latent effects
- … a sea of minutiae
Closures and Conduits

• A **closure** is a “domain of semantic predictability” where parameter bindings make sense. “What you ask for is what you get.”

• A **conduit** is an approved mechanism for communication between closures

• **Contract**: if you use only the conduit, and all will work as documented

• Can **close the box** and stop remembering the minutiae that make the closure work

• Closest thing we have to a **unit of modularity**
Closure is not new

- Network appliances
- Highly reliable subsystems (e.g., DHCP and DNS)
- Switch and grid fabrics
- Anything that always does exactly what you say.
Creating a Closure

requirements
  Design
  processes
    Implement
    parameters
      Set values
      configurations
        Select subsets
        policies
          closure

Extract

behavior
  Codify
  tests
    Measure effects
    states
      Coherence

Software Engineering process

Closure Engineering process
Kinds of Configuration Parameters

• **Behavioral (exterior)**: determine what user sees

• **Incidental (interior)**: no effect on user perception
  – Dependent: determined by choices for behavioral parameters
  – Environmental: determined by operating environment
  – Arbitrary: value doesn’t affect behavior
Example: web server

• Exterior (behavioral) parameters
  – What content is served?
  – Response time/robustness/reliability
  – Bindings to other services (e.g., databases)

• Interior (incidental) parameters
  – Where to locate software (environmental)
  – Where content is stored (depends upon response time, robustness, etc)
  – Protection model for files (depends on content)

• Apache httpd.conf: about 80% interior
SA and SQA

• System administration is the **opposite** of software quality assurance
• In SQA, we want to **locate** problems in software
• In system administration, we want to **avoid** problems
• Primary technique: **limit achievable configuration states**; validate all possible states
Minimizing Achievable State

- Always use unvarying order for configuration operations
- Generate whole configuration from same declaration every time
- Always copy a validated state
- Always use same values for arbitrary parameters
- Enforce invariant structure for configuration files
Constraints and Expense

• Interior (incidental) parameters are under-constrained
  → incidental heterogeneity
  → difficulty learning or troubleshooting
  → maintenance expense!

• By contrast, exterior parameters are strongly constrained
  → enforced homogeneity
  → shorter learning curve
  → cheaper process maturity!
(Intelligent?) Agents

• Our approach: interpose an agent between system administrator and system

• Input to agent: exterior parameters

• Output from agent: settings for all parameters, including incidental ones

• Minimal intelligence: maps from desired exterior behavior to incidental configuration
Cost and Value

- Value of agents: site consistency and homogeneity improve **portability of validation**
- Cost of agents: must represent enough exterior data to completely determine incidental data
  - Must define service constraints
  - Must supply all content **through** the agent
- Result: agent-controlled web servers **require** content staging!
Theory and Practice

• Theory: how do closures combine?
  – Formal definitions
  – Preliminary results

• Practice: what building blocks does one need to create a closure?
  – Incremental changes to configuration files
  – Service provision architecture
Theory: Preliminary Results

• Can easily construct compositions of closures that are not closures.
• Key component in maintaining closure during composition is awareness of parameter overlap between closures
Theory: Some Subtleties

• Closure A dominates closure B if for every reasonable configuration of A there is a matching and consistent configuration of B.
• Dominance isn’t transitive: If A dominates B and B dominates C, then A need not dominate C.
• Even if dominance is transitive in a set of closures, this does not assure global consistency.
• Problem: lack of parameter knowledge.
Foolproof Composition

Dominance hierarchy

Parameter hierarchy

A → B means “A controls B”

Containment represents parameter structure”
Practice: Preliminary Prototypes

- Build closures based upon transactional file control, not stream editing
- Build coherent service architecture by interacting with file closures
Incremental File Editing

/etc/services
↓ parse ← XML structural declaration
services.xml
↓ change ← Editing commands
New services.xml
↓ render ← XSLT format
New /etc/services
Declaring File Structure (once)

```xml
<xmft:file path="/etc/services">
  <xmft:repeat sorted-by="port" keys="service:port+prot" name="lines">
    <xmft:line>
      <xmft:var type="string" desc="service name" name="service"/>
      <xmft:whitespace/>
      <xmft:var type="integer" desc="ip port number" name="port"/>
      <xmft:text>/</xmft:text>
      <xmft:choice type="protocol name" name="prot">
        <xmft:option><xmft:text>tcp</xmft:text></xmft:option>
        <xmft:option><xmft:text>udp</xmft:text></xmft:option>
      </xmft:choice>
      <xmft:repeat>
        <xmft:whitespace/>
        <xmft:var type="string" desc="protocol alias" name="alias">
        </xmft:var>
      </xmft:repeat>
    </xmft:line>
  </xmft:repeat>
</xmft:file>
```
Preliminary Editing Operations

• insert what (service='tftp',
  port='6900', proto='udp')
• delete where (service='tftp'
  and proto='udp')
• update where (service='tftp')
  what (port='8800')
20-20 Hindsight: Ideal Editing

assert service=tftp port=6900
proto=udp
retract service=tftp
Service Synthesis: FTP

configuration
group service

file

package

process

services

inetd

/etc/services

/inetd.conf
Conclusions

• Our lives as system administrators are full of interdependent minutiae
• Behavioral thinking can determine which are important and induce a modularity of effect
• Agents can manage modules and shield us from dealing with non-behavioral parameters
• Result is increased consistency, lower bug exposure, and lower administrative cost.
Lessons Learned

• We seek the rosetta stone that will link system administration to the rest of computer science and engineering, as well as mathematical knowledge
• Subtleties of our goals and practices cause surprising and subtle results
• Cannot simply apply known theorems; must repeat their proofs and see if they still work!
Current Status

• Software still prototype
• New theory:
  – Can split validation into two phases:
    1. Avoid effects of latent variables
    2. Validate outcome
  – Avoidance of latent problems is **statically verifiable** in configuration scripts
Acknowledgements

• Lssconf working group
• Configuration Management and Infrastructure workshops and BOFs
• CFengine (happy 10\textsuperscript{th} birthday!)
• Network Appliance Corp
Contact

• Email: couch@cs.tufts.edu
• Speaker table: moved to 5:30 pm session (due to unavoidable conflicts)