Modeling Next Generation Configuration Management Tools

Mark Burgess (Oslo University College)
Alva Couch (Tufts University)

Aspects and Closures and Promises (Oh My!)

- Theories of configuration management employ three distinct terminologies:
  - Aspects (Anderson)
  - Closures (Couch)
  - Promises (Burgess)
- How are these terms different or similar?
- We seek the “Rosetta Stone” that relates the three theories.

The Rosetta Stone

- The three theories concentrate on different parts of the problem.
- **Aspects** model **dependencies**
- **Closures** model **behaviors**
- **Promises** model **interactions**
- Comprehensive aspects+behavioral closure = closures
- Closures+promises = distributed closures
- Any tool must incorporate some form of each kind of model (consciously, or not!)

Why should we care?

- *"Cost:"* what we pay for the process.
- *"Quality of service:"* how quickly one can react to changing needs.
- Myth: the *tools and technologies* we use determine the cost and quality of configuration management.
- Reality: cost and quality are more related to *how we conceptualize and define the configuration management problem*.
- It's not what we **use**, but rather how we **think**.

Example: Cfengine

- Cfengine supports a particular way of thinking about configuration management.
  - Decentralized
  - Incremental
  - Partial
  - Convergent

Example: Puppet

- By contrast, Puppet supports a different way of thinking:
  - Centralized
  - Comprehensive
  - Replacing
  - Overriding

Which tool should I choose?

- So, which of the plethora of configuration management tools is most appropriate to *my* site or problem?
- *Wrong question!*
- Better question: Which way of *thinking* best supports what I need to do?
- Then (and only then): what tools support that kind of thinking?

Our contribution

- Better understanding of
  - Complexity of configuration management.
  - How various conceptualizations of the problem relate to one another.
  - The common ground there is between conceptualizations.
  - How future tools can share data and cooperate with one another.
  - How we can combine strategies toward a better and less costly process.

How hard is configuration management?

- How hard can it be to tell everyone exactly what to do? Seems easy enough…
- But there are many risk factors:
  - Interdependencies and interactions between subsystems.
  - Some are known, some are unknown!

Modeling interactions

- [Sun 2005]: *complexity arises from interactions between subsystems.*
- An **aspect** [Anderson 2005] is a set of configuration parameters whose values are interdependent and constrained.
- Example: all of the locations in which the hostname of the machine appears in /etc form ***one local aspect***.
- Example: it makes no sense to create a web server without an advertised address. So its address in its configuration and in DNS comprise ***one distributed aspect.***

A complex aspect

- For a webserver to work,
  - The document root has to exist
  - The content has to be located there.
  - The protections have to allow the web server access.
  - The configuration of the web server has to permit access.
  - Etc
- These choices must be *coordinated*.

Everyday aspects

- The average system administrator copes with aspects on a daily basis.
- Consider the following common story:
  - You configure a system properly.
  - It works.
  - You add a package.
  - Something breaks.
- Somehow, *some aspect was violated* by the package installation.

Properties of aspects

- An aspect is a pair <P,C> where
  - P is a set of parameters.
  - C is a set of constraints.
- A single parameter is an aspect.
- A union of aspects is an aspect.
- A configuration is an aspect.

Why aspects are important

- A *tool-independent* way of describing interaction and complexity.
- Allow **approximating the difficulty** of a specific configuration management task.
- Allow intelligent tool choices based upon task complexity.

Closures

- Aspects describe constraints operating within a configuration.
- ***Closure***: a deterministic map between ***configuration*** and ***behavior***.
- If we have identified all aspects, then that map is well-defined. We say the union of all aspects is **closed**.
- If some aspects remain unknown, the map might not be well-defined. We would then say that the union of all aspects is **open.**

Some examples

- One creates a web-service closure [Schwartzberg 2004] by identifying and controlling all aspects that determine web service behavior.
- One creates an IP address closure [Wu 2006] by identifying and controlling all aspects that determine IP address assignment behavior.

Discovering closures

- The theory of aspects shows that closures are not **created**, but instead **discovered**.
- If we identify and manage all pertinent aspects, and map out behaviors, we’re done; behavior is deterministic!
- Every configuration management tool tries to do this.

How do closures communicate?

- To make larger closures from smaller ones, smaller closures must communicate with one another.
- Question: how is this accomplished?
- Answer: through **promises**.

Promise

- A unit of communication between two autonomous systems.
- Describes intent of sender to receiver.
- A basic part of any kind of service discovery

Promises glue closures together

- Very often, closures must coordinate distributed aspects.
  - Must map clients to servers.
  - Must distribute resources to clients.
  - Often, this is done via request/response.
- A *promise* is an offer, rather than a *request*. It says “certain requests will be granted by the sender”.

Practical promises

- Many might consider promises a purely theoretical and abstract idea.
- In fact, they’re present in every distributed system.
- We can think of a fileserver’s execution of an NFS daemon as a “promise to provide service”.
- We can think of an NFS client mount request as a “promise to use service”.

Promises and exceptions

- One reason for promises: avoid dealing with exceptions.
- In a request/response environment, must always cope with requests that cannot be satisfied.
- A promise does not explicitly require a response.
- The response may come asynchronously, or not at all.

Example of promises in action: service binding

- Multiple servers, one client.
- Servers promise service to client.
- Client promises to use service from one server.
- This establishes a binding.
- No central coordination necessary.

What does it all mean?

- Current tools manage aspects.
- Tools are for the most part unaware of behavior.
- Mapping behaviors is a really hard problem.
- Closures provide a tangible way to break that hard problem up into simpler ones.
- Promises provide the glue that allows closures to efficiently communicate.

The point

- Aspects define constraints.
- Closures define predictability.
- Promises define intent.
- This allows automatic verification of configuration information!

CM-TNG

- Current tools do nothing more than assert what they believe to be appropriate aspects.
- The next frontier: automatic validation and verification.
- Mechanism: closures and promises.

Verification

- Before binding a client to a server, check that the server is functioning via a promise.
- The server checks itself through a closure.
- The local aspect is not set to a value until this remote check is made.
- No more broken service links!

“Present Work”

- “The other half” of this work:
- Burgess and Couch, “Autonomic Computing Approximated by Fixed-Point Promises,” Proc. MACE 2006.
- Purport: conceptualize the notion of management entirely in terms of a set of convergent operators acting in a distributed network.
- A promise is a form of “operator.”

Conclusions (for developers)

- The combination of the theories is greater than the sum of the parts.
- Aspects help one to **discover** closures.
- Closures and promises allow one to **manage verified aspects**.
- This is the first step toward configuration tools that are aware of and manage **behavior** rather than configuration.

Conclusions (for users)

- Aspects provide a methodology by which one can evaluate tools.
- A tool either “manages an aspect” or it does not.
- Some tools are “closer to managing closures” than others.
- Aspects and closures provide a way of comparing tool capabilities.
- Promises provide a way of describing and comparing distributed management tools.

Thanks!

- Mark Burgess (Mark.Burgess@iu.hio.no)
- Alva Couch (couch@cs.tufts.edu)