Troubleshooting with human-readable automated reasoning

Alva L. Couch, Tufts University, couch@cs.tufts.edu

Mark Burgess, Oslo University College and CFengine AS, mark@cfengine.com

Formal logic?

How many of you have studied logic?

… because I am going to do something very "illogical".

"Logic is a bouquet of pretty flowers, that smell bad." ☺

- Leonard Nimoy, as Spock

What is this talk about?

- How **to troubleshoot systems based upon their architecture,**
- based upon a **naïve logic of causal relationships** between architectural entities,
- that is **optimized for readability by sysadmins,** understandability, and efficient computation.
- that describes which relationships might be present as a first-order approximation, like a "**bloom filter for logic**"

Architecture and troubleshooting

- Architecture defines **connections between entities.**
- Troubleshooting requires **understanding those connections.**
- We provide a way to:
  - recall connections relevant to a problem
  - make and explain new connections

  via a strange kind of logic.

Entities and relationships

- Entity: something one manages, e.g.,
  - Hosts
  - Services
  - Classes of hosts or services
- Relationship: some constraint between entities
  - Causal: *determines*, *influences*
  - Dependence: *provides*, *requires*
  - Intent: *promises*, *uses*
  - Class: *is an instance of*, *is a subclass of*
  - Structural: *is a part of*, *is a component of*

Architectural facts

host01	***provides***	**file service**
subject	*verb phrase*	object
entity	*relationship*	entity

- Notation

```
host01|provides|file service
```

Inference rules

Make new connections between entities.

Change the **level of abstraction** of a fact.

Three ways to infer relationships

Implications: raise the level of abstraction

Inverses: allow a fact to be “reversed”

Connections: document indirect relationships

Implication

If host01 *provides* file service,
then host01 *influences* file service.

provides : a concrete relationship
influences: an abstract relationship

motive: reason abstractly, report concretely.

Notation:
```
provides->influences
```

Inverses

host01 *provides* file service

whenever

file service *is provided by* host01

This is just a matter of notation.

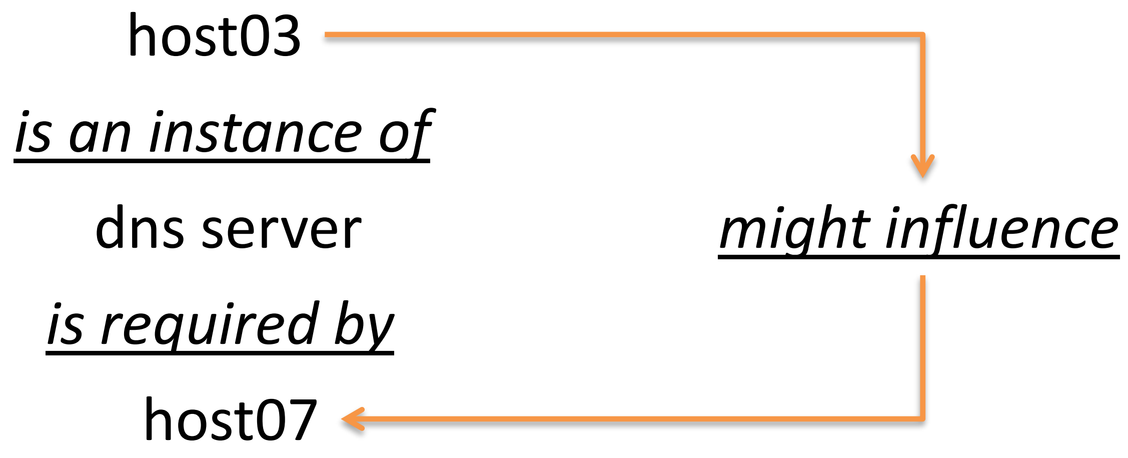
It makes other rules easier to write down.

Notation:

```
provides<>is provided by
```

Connections

If host03 *is an instance of* dns server,
and a dns server *is required by* host07,
then host03 *might influence* host07.



Notation:

```
is an instance of^is required by^might influence
```

Why this is strange

- Most attempts at computer logic attempt to **translate English into logic** and then reason from that.
- This method **translates architectural information to simple English** and then reasons from that, **without translating the English into logic!**
- Main advantage is **incredible speed!**

Exterior semantics

- Usually, one defines the meaning of English phrases **in a dictionary**.
- In our system, one defines relationship meanings via their **interaction with other relationships**.

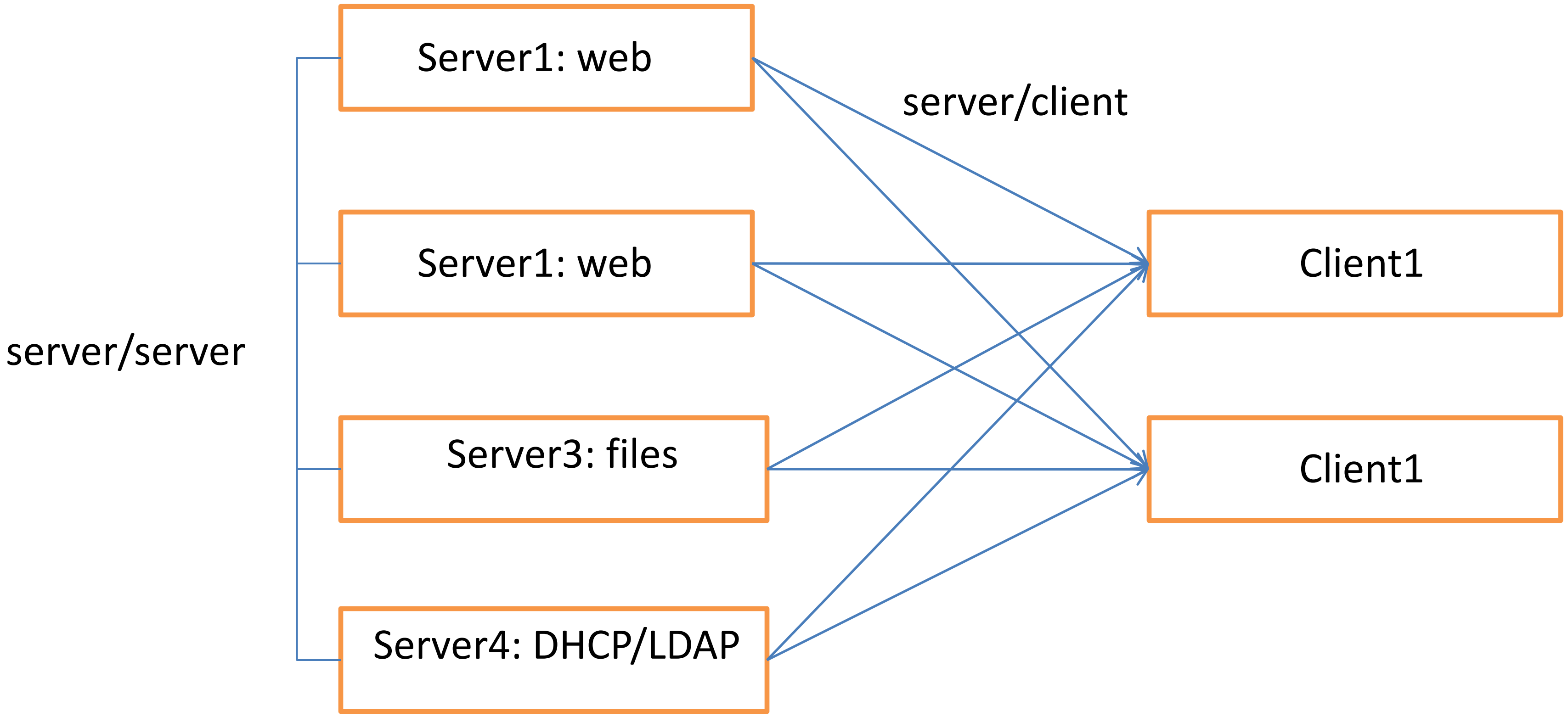
What does “influences” mean?

determines->influences
determines^determines^determines
determines^influences^influences
influences^determines^influences
influences^influences^influences
determines^has part^determines
determines^is a part of^influences
is an instance of^determines^determines
has instance^determines^influences
provides^is required by^might influence

Two claims of this paper

- Claim 1: this logic is easy to describe and compute.
- Claim 2: the results of inference are human-readable.

Demonstration:
A really simple architecture



A naïve architectural description

```
file server|provides|user file service
file server|provides|web file service
file server|requires|dns

web server|provides|web service
web server|requires|web file service
web server|requires|dns

network server|provides|dns
network server|provides|dhcp

workstation|requires|dns
workstation|requires|dhcp
workstation|requires|user file service
workstation|requires|web service
```

```
# assign roles to machines
server1|is a|web server
server2|is a|web server
server3|is a|file server
server4|is a|network server
client1|is a|workstation
client2|is a|workstation
```

What can cause problems with client1?

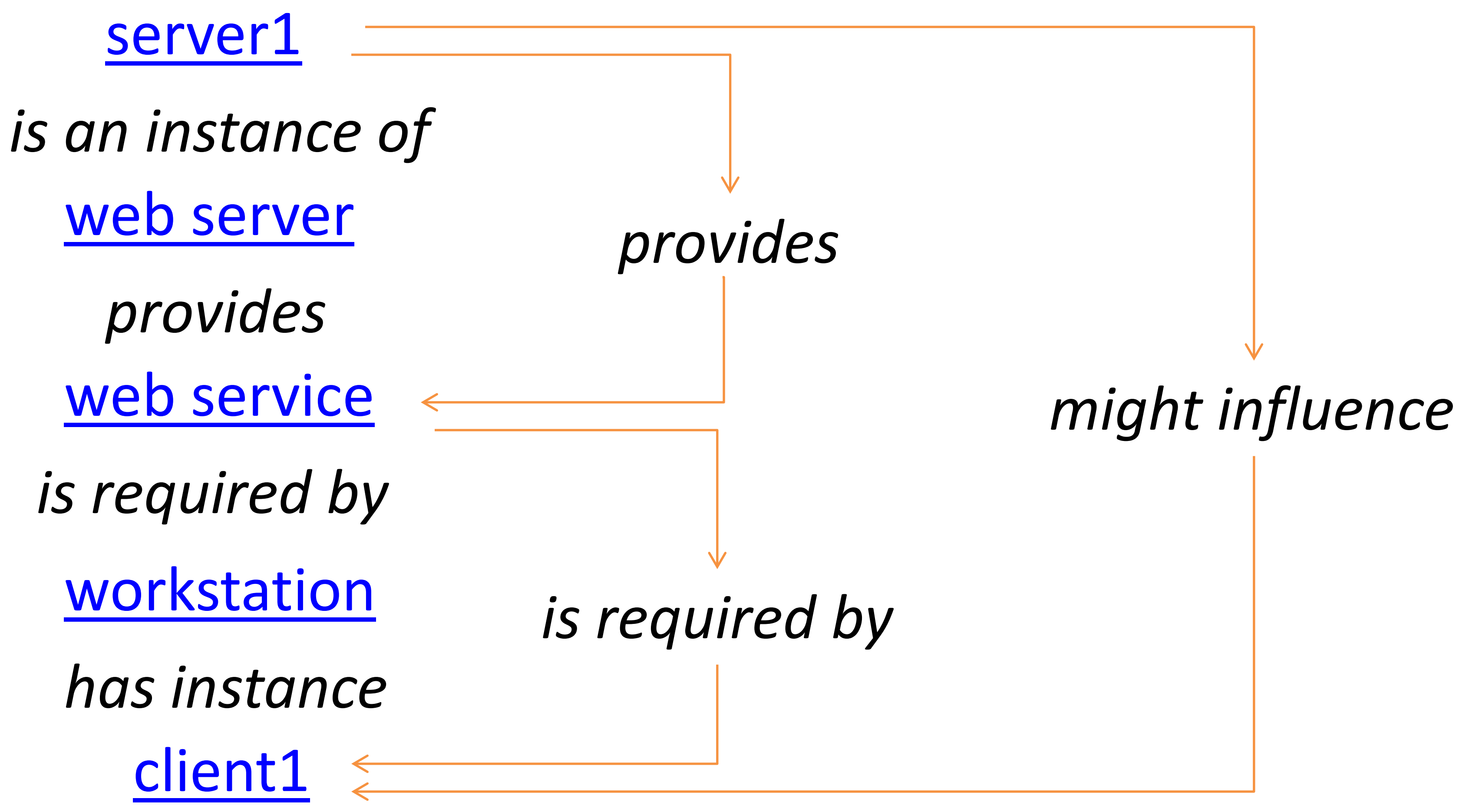
Architectural facts:

```
client1|requires|dhcp
client1|requires|dns
client1|requires|user file service
client1|requires|web service
```

Inferred facts:

```
server1|might influence|client1
server2|might influence|client1
server3|might influence|client1
server4|might influence|client1
```

server1 might influence client1



We don’t need the details

server1

is an instance of

web server

provides

web service

is required by

workstation

has instance

client1

- We can omit the logic.
- The flow speaks for itself.
- By sticking to simple inference, we can understand it without explanation.

A simple prototype

- A Perl CGI script
- All calculations online from text declarations.

Configuring the prototype

- Describe architecture
- Reuse rules.

Using the prototype

- Choose a trouble-spot; connections are listed.
- Click on a connection to explain it.

Critique

+: uses simple sentences

-: doesn’t handle complex sentences

+: very fast

-: doesn’t support complex logic

+: very quick answer

-: relatively naïve answer, the “shortest explanation”

But

a naïve answer is better than no answer at all!

Lessons learned

- Causal connections are much more useful than unrestricted connections.
- Readable logic is much more useful than highly accurate (and expensive) logic.
- A weak logic can be a useful tool in troubleshooting.

Future work

- Field testing.
- Coding in Map/Reduce for at-scale calculations.
- Using regular logic to verify discovered relationships.
- Coupling with other information sources.
- Apply this to other domains, e.g., documentation.
- Build this algorithm into Cfengine Constellation.

Please

- Play with the prototype:

http://www.cs.tufts.edu/~couch/topics

- Let us know
  - how it works for you
  - how it could be improved
  - what it should really do

Alva L. Couch, couch@cs.tufts.edu

Mark Burgess, mark@cfengine.com