IT’S ELEMENTARY, DEAR WATSON: APPLYING LOGIC PROGRAMMING TO CONVERGENT SYSTEM MANAGEMENT TASKS

Dr. Alva L. Couch and Michael Gilfix
Electrical Engineering and Computer Science
Tufts University, Medford, MA
Email: couch@eecs.tufts.edu
Web: http://www.eecs.tufts.edu/~couch/prolog
THE SYSTEM MANAGEMENT
“LANDSCAPE”

“rules”  
RDist  CFEngine
Slink/Distr

“scripts”

“configuration”

“behavior”

“where no one has gone before!”

“where no one needs to go!”

PIKT

SWatch
“DISCLAIMER”

- We love CFEngine and PIKT.
  - Both do what they claim.
  - Both are optimal in execution time.
- But extending either one is difficult.
  - Is execution speed all-important?
  - Or is our time more valuable?
ELEMENTARY!

- CFEngine is “almost” Prolog (see paper).
- PIKT scripts “must do” what Prolog does.
- So try using a subset of Prolog.
- Gains:
  - Prolog programs “look more like policy”.
  - Loops and configuration changes are implicit.
- Losses:
  - Can’t explicitly control actions.
  - Lower runtime efficiency.
OUR EXPERIMENTAL PROTOTYPE

- SWI-Prolog on Solaris 7.
- Used builtins for manipulating files.
- Wrote extensions for manipulating system configuration, processes, etc.
- Wrote CFEngine-like configuration primitives (except for file distribution).
- Wrote PIKT-like output parsing primitives.
- Used these to pre-parse common command output formats (du, ps, quota, who, etc)
A USEFUL SUBSET OF PROLOG

- We don’t need the whole language.
- We only have one real goal: system health.
- Every script we write will be to assure health.

health: - /* same goal every time */
    passwd(Login, _, _, _, _, Home, _),
    du(Home, Usage),
    Usage>20000,
    email(Login, 'you are a pig!',
          'oink!').
?- health, fail. /* check ALL cases */
**Implicit Actions**

- In a normal scripting language, this “program” would look like:

```plaintext
for each Login, Home pair,
    Usage = du(Home)
    if (Usage>20000)
        email(Login,'you are a pig!','oink!')
```

- In Prolog, the underlined text was *implicit* and *inferred from context.*
IMPLICIT QUERIES AND TESTS

- `passwd(Login,_,_,_,_,Home,_)` sets Login and Home to *all valid pairs* in turn.

- `passwd('couch',_,_,_,_,
  '/home/couch',_)` is a *conditional test* that’s true if the home directory of couch is `/home/couch`!
**IMPLICIT EFFECTS**

- `owner('/etc/motd',Owner,Group)` reads Owner and Group
- `owner('/etc/motd',0,10)` chown's `/etc/motd`

**IMPLICIT CONVERGENCE**

- `copy('/Master/etc/motd','/etc/motd')` copies *only if necessary.*
- `link('/Master/etc/motd','/etc/motd')` makes a link *only if necessary.*
**CONTROVERSIAL CLAIMS**

- Prolog can emulate the function of CFEngine through appropriate extensions.
- Prolog supports script genericity as in PIKT.
- See paper for details.

**WHERE NO ONE HAS GONE BEFORE:**

- Service-level declarations.
- Declarative rules for dynamic policies.
SERVICE-LEVEL DECLARATIONs

- Define generic high-level services, e.g., ftp.
  
  `health:- service(ftp), os(Os),
   config_path('inetd.conf',Os,Path),
   config_path('ftpd',Os,Ftpd),
   file_base_name(Ftpd,FBase),
   appendIfNoSuchLine(Path,
       [ftp,stream,tcp,nowait,root, Ftpd,Fbase]).`

- Describe systems requiring service:
  
  `service(ftp):- hostname('fred').
  service(ftp):- not os('solaris').`
DECLARING DYNAMIC POLICY

- The lesson of COBOL:
  - subtle: requires a Dilbert
  - “natural language” policy
  - process
  - “computer language”
  - easy: any PHM can handle it.

- “The best we can do” is to code policies so that once they’re coded and working, their meanings are obvious (to PHM’s)!
POLICY TO PROCESS

- Policy:
  kill all Internet Explorer runaways that remain after the user has logged out.

- Prolog: “messy, but effective”

  \[
  \text{ps_f(User, Pid, _, _, _, _, _, _, Cmd),}
  \text{match(Cmd,'iexplorer'),}
  \text{not who(User, _, _, _),}
  \text{kill(Pid, 9).}
  \]
**PERFORMANCE**

- If written well, Prolog executes as quickly as comparable Perl code.
- If written poorly, it can take thousands of times longer due to superfluous implicit loops.
- Example: the sequence
  
  ```prolog
  passwd(_,_,_,_,_,_,Home,_),
  passwd(Login,_,_,_,_,_,Home,_,)
  ```

  can take thousands of times longer than the second goal alone.
CONCLUSIONS

- Prolog is a “glue language” that allows coding of rules for both configuration management and behavior modification.
- But efficient coding is a *subtle art*.
- Prolog is an *assembly language* in which future configuration tools can be written.
THE FUTURE

- Preprocessor converts policy rules into Prolog.
- Syntax closer to usual scripting languages.
- Type checking and superfluous loop elimination.
- Example of proposed declarative syntax:
  ```prolog
  pid(Pid), Name=pidName(Pid),
  match(Name, 'iexplorer'),
  User=pidUser(Pid),
  not userTty(User),
  kill(Pid, 9).
  ```