Configuration Tools

Working Together

Paul Anderson
dcspaul@inf.ed.ac.uk

Edmund Smith
esmith4@inf.ed.ac.uk

School of Informatics

Edinburgh University
Introduction

- **Assumptions**
  - Automated configuration is becoming essential
  - Existing tools are inadequate - why? ....

- **As with the early days of hardware** -
  - Existing tools are like early machine code
  - No high-level languages
  - No code sharing or libraries
  - High-level development difficult (impossible?)

- **No progress until we get the equivalent of high-level programming languages and compilers**

- **How do we do this?**
  - Providing common interfaces to low-level tools
  - Providing generic solutions to high-level problems
An overview

- System configuration
  - what is it?
  - configuration tools
- Some “high-level” operations
- One possible interface
System configuration

System Operating according to specification

Configuration

Hardware

Software

Specifications and Policies

Feedback
System configuration

□ Starting with:
   - Several hundred machines (with empty disks)
   - A “repository” of all the necessary software packages
   - A specification of the overall required service

□ Load the software and configure the machines to provide the service
   - This involves many internal services – DNS, LDAP, DHCP, NFS, NIS, SMTP, Web ...

□ Reconfigure the machines whenever any of the inputs change
   - Requirements, hardware, or software
Levels of configuration automation

“Copy this disk image onto these machines”

“Put these files on these machines”

“Put this line in sendmail.cf on this machine”

“Configure machine X as a mail server”

“Configure machine X as a mail server for this cluster”
(and the clients will automatically be configured to match)

“Configure any suitable machine as a mail server for this cluster”
(and the clients will automatically be configured to match)

Configure enough mail servers to guarantee an SMTP response time of X seconds
Configuration tools

- **Low-level concerns dominate**
  - These involve most development & maintenance
  - Designed from the bottom-up
  - Little effort expended on high-level concerns
    For example, relationships between machines

- **Existing tools do not communicate**
  - Different languages, architectures and protocols
  - LCFG, cfengine, bcfg, etc....

- **Tools are usually developed by working syasadmins**
  - The entire problem is too big for this to continue
  - An interface standard allows sharing
  - Specialists can develop the “tricky bits”
    Like the math library for a compiler
An overview

- System configuration
- Some “high-level” operations
  - Classing
  - Aggregation
  - Sequencing & planning
  - Delegation & authentication
- One possible interface
Classing

- Classing is one way of specifying the configuration of individual machines at a higher level -
  - A “web server”, a “laptop”, a “student machine”, a “Debian machine”, a “print server”, a “Dell gx240”

- Many tools provide classing facilities
  - This is an example of an abstract high-level operation

- No existing tools handle this well
  - Classes are really “aspects” with overlapping concerns
  - These often conflict if created by different people

- Constraints of priority mechanisms may help
  - Can we solve these problems in a tool-independent way?
Aggregation

- There is often a relationship between the configuration of one machine (a “server”) and many others (the “clients”)
  - An NFS server wants to export to known clients
  - A DHCP server wants to provide fixed IP addresses

- The configuration for the server should be automatically constructed by “aggregating” the client information
  - We don’t want mismatches
  - We don’t want the firewall hole to remain when the client has been decommissioned

- This is another tool-independent operation
  - A common interface would allow different solutions to be “plugged in”
Sequencing & planning

- In a “declarative” system we would like to specify (only) the desired state
  - The tool computes the actions required to get there
- This may involve several intermediate states
- A fully automatic tool needs to plan the sequence of changes, so that intermediate states are all valid
- For example -
  - Configure the new server
  - Switch over the clients
  - Decommission the old server
- Such automatic planning is tool independent
Delegation & authorisation

- Existing tool tend to have a binary notion of authorisation
  - “root” or user
- This is hard to improve if configuration involves access to the low-level configuration files
- A high-level tool makes it possible to authorise access to meaningful properties
  - A single property may involve wide-ranging changes to (parts of) many config files
  - Eg. configure my machine to look like a student lab machine
- Higher-level specifications allow security implications to be automatically validated more easily
An overview

- System configuration
- Some “high-level” operations
- One possible interface
  - A document-based interface
  - An example
A document model

☐ A common lexicon is not necessary
  – High level tools do not have to understand the meaning of the parameters
  – This would be hard to agree at this stage anyway

☐ We do need to define the data structure
  – Simple key-value pairs (entries)
  – Compound data types (structs)
    Represents lists and records

☐ XML is a convenient format
  – Is is “portable”
  – There is a selection of tools
  – The paper contains some examples

☐ We do not need to define the architecture
  – Centralised or P2P, for example
A (sketch of an) example

- Configure two DHCP servers on every Ethernet segment
  - Providing fixed addresses for all other machines
  - Yes, there are other ways of doing this!
    But this is illustrating a general solution

- Select two candidate machines with a constraint mechanism
  - In practice, we want two active machines!

- Use a classing mechanism to allocate a DHCP server class to the chosen machines

- Use an aggregation mechanism to collate the IP/MAC mappings from each individual machine
Conclusions

- Automating higher-levels of system configuration involves some common generic operations
- These are different from the detailed low-level manipulation
  - And not really handled by existing tools
  - They need specialists to work on them
- Addressing them independently in different tools is not likely to be productive
- What is the simplest interface that we can create to allow us to build generic solutions to these problems?
  - This does not require a common lexicon
  - Or a common architecture
Configuration Tools
Working Together