

# ***Specification Languages for Fabric Configuration***

Paul Anderson  
<[dcspaul@inf.ed.ac.uk](mailto:dcspaul@inf.ed.ac.uk)>

University of Edinburgh

 School of  
**informatics**



# Overview

- What is fabric configuration?
  - Why is it important for the Grid?
- The current state of fabric configuration
- What we would like to be able to do
- Some specific issues looking for solutions ..
  - Generating configurations from policies and high-level specifications
  - Composing aspects
  - Distributed evaluation
  - Provenance and authorisation
  - Usability
  - Security

# ***What is “fabric configuration” ?***

- Starting with:
  - Several hundred new PCs with empty disks
  - A Repository of all the necessary software packages
  - A specification of the 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 when the required service specification changes
- Reconfigure the machines to keep providing the specified service when the fabric changes
  - Hardware fails, or is replaced with different hardware

# ***Grid fabrics***

- Fabric configuration is not the same as the configuration of *Grid* services (applications).
  - There are however some similarities
- All *Grid* applications assume the existence of correctly configured "fabrics"
  - Quality of service depends on the fabric quality
- Misconfiguration is a major source of errors
  - What happens if one node in 500 has the wrong version of a maths library installed?
- *Grid* security depends on fabric security
  - Can we be sure that every node has the latest security patches?
  - Can we prove that security policies are being honoured?

# ***The current state***

- Current configuration tools are extremely primitive
  - Most are completely ad-hoc developments
  - See the reports at [www.gridweaver.org](http://www.gridweaver.org) for more background information
- The better existing tools tend to ...
  - Provide an abstraction of the low-level configuration information
  - Provide a template mechanism to group common features for classes of nodes
  - Provide a mechanism for composing classes
  - Provide a method of deploying the configuration
- This is a bottom up approach and does not usually connect with any specification for the function of the whole fabric

# Configuration files

- Each service (on each node) requires one or more configuration files, all with their own syntax and peculiarities.
  - Traditionally, these would have been edited by hand

```
snmpd : localhost 129.215.216. 129.215.58.
```

```
stunnel : localhost inf.ed.ac.uk
```

```
.dcs.ed R$* < @ [ IPv6 : $+ ] > $* $: $2 $| $1 < @@ [ $(dequote $2 $) ] > $3 mark IPv6 addr
```

```
slapd : R$+ $| $* < @@ $=w > $* $: $2 < @ $j . > $4 self-literal
```

```
Mc R$+ $| $* < @@ [ $+ ] > $* $@ $2 < @ [ IPv6 : $1 ] > $4 canon IP addr
```

```
Modeline "1024x768" 65 1024 1032 1176 1344 768 771 777 806 -hsync -vsync
```

# ***The node “profile”***

- It is possible to represent the entire configuration of single node as a simple data structure which we call a “profile”
- Any node is completely specified by a combination of the profile, and a (common) repository of software packages
- Some example elements might be ...
  - Sntp-relay = mailrelay.ed.ac.uk
  - Packages = emacs-3.7 apache-2.4 foo-45.5 ...
  - Users = joe, jane, @staff
- The profile only needs to contain the information which may vary between machines in the same management domain

# ***LCFG profiles***

- The profile can be conveniently represented as a simple XML document
- LCFG is one tool which can build nodes from the "bare metal" given a profile and access to the repository
- Typical profiles currently have about 5000 elements
- The level of the information in the profile is usually very low
  - ie. close to the contents of the configuration file
  - Think of the profile as the "machine code" for the configuration compiler

# ***Classes and templates***

- Since we do not want to write each profile by hand, most systems provide a "class" mechanism for grouping common parameters
  - eg. "laptop", "web server", "toshiba", "student m/c"
- This involves value (not type) inheritance, and has some similarities with prototype-based programming languages
- Conflicts between values in multiple templates are common
  - Normally resolved by a crude "override"
  - LCFG has "mutation"

## ***LCFG configuration example***

```
#include <lcfg/os/redhat71.h>
```

```
#include <lcfg/opts/redhat71_sp1.h>
```

```
#include <lcfg/hwbase/dell_optiplex_gx240.h>
```

```
#include <inf/sitedefs.h>
```

```
#include <inf/wire_c.h>
```

```
#include <inf/dcspaul.h>
```

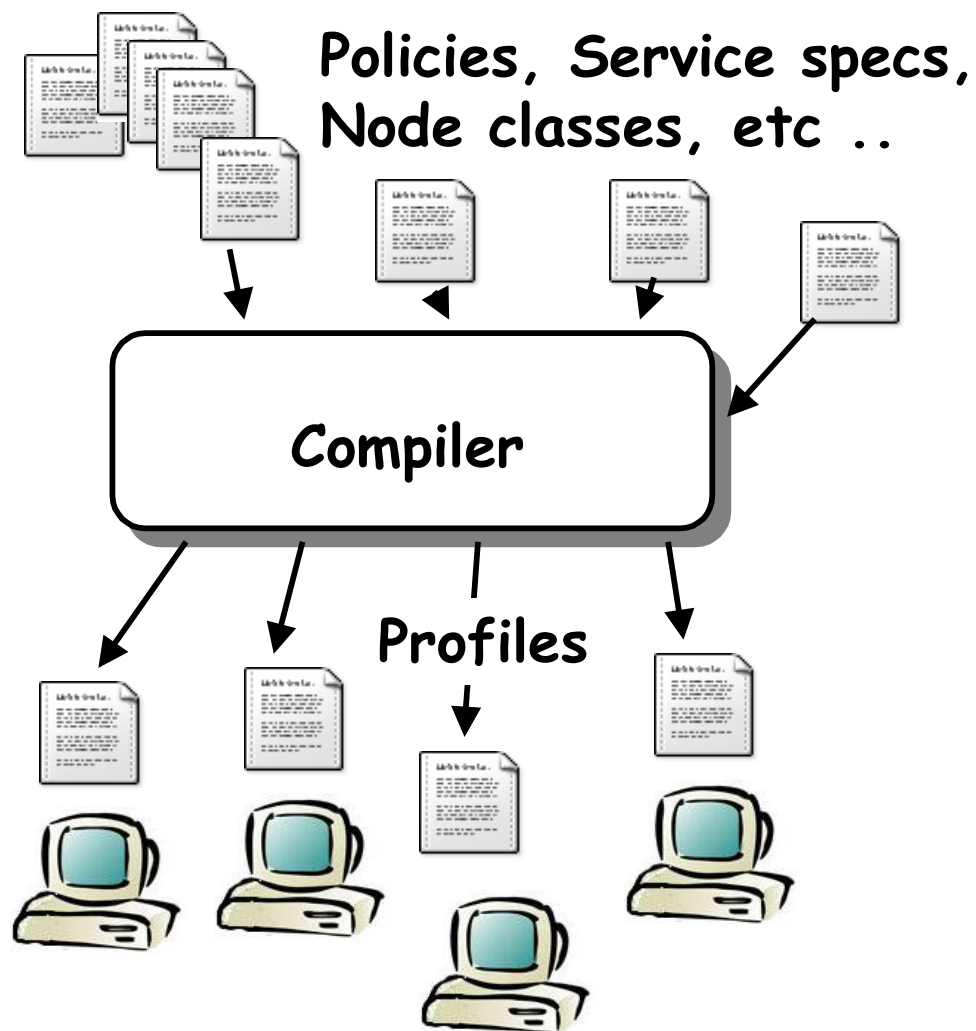
```
dhclient.mac 00:06:5B:BF:87:2E
```

```
!xfree.mice mADD(usbwheel)
```

# ***What do we really want ?***

- When specifying the configuration of a fabric, we want to talk in terms of higher-level entities ...
  - Services involve relationships between nodes
    - A shared filesystem involves a fileserver and a number of clients, configured to mount the specified server
  - Policies specify predicates that must be satisfied globally
    - Students must not be allowed to log in to file servers
    - We must have at least two DHCP servers on each network segment
- Current systems do not provide a connection between these specifications, and the deployed configuration

# Compiling the configuration



- We need a language to describe the high level specifications and policies
- We need a "compiler" to generate machine profiles
- We need an engine to deploy and maintain the configuration on the nodes - but this is not the focus of this talk

# ***Some specific issues***



- Generating configurations from policies and high-level specifications
- Composing aspects
- Distributed evaluation
- Provenance and authorisation
- Usability
- Security

# ***High-level configurations***

- We don't currently have a good practical way of modelling higher-level concepts in the fabric
  - There are many components involved
  - They change rapidly
  - Different sites have different management policies
- How do we generate configurations which satisfy policies ?
  - We must have at least two DHCP servers on each network segment
- Many specifications are more naturally expressed as constraints, than explicit values
  - "I need a DNS server" (but I don't care which)
- We do want a declarative language
- The prototype model feels natural
  - Give me a machine like X, but a bit different ...

# Aspects

- Different people are frequently responsible for different “aspects” of a fabric
  - Web service, Dell hardware, Linux, Solaris, Security, File servers, etc ...
- It is convenient for these people to be able specify the parameters for their aspects without being concerned about other aspects of the configuration
- Often, the values supplied conflict
  - Current resolution mechanisms are very crude
- Conflicts often occur because the aspect author is forced to supply explicit values when they really only require a much looser constraint
  - We are looking at how CLP might be used ...

# Aspect composition

- The language forces explicit values to be specified:
- Aspect A
  - Use server Y
- Aspect B
  - Use server X
- This conflict is irreconcilable without human intervention because we don't know the intention
- The user really only wants to say ...
- Aspect A
  - Use any server on my ethernet segment
- Aspect B
  - Use one of the servers X, Y or Z
- These constraints can be satisfied to
  - Use server Y (assuming Y is on the right segment)

# ***Distributed evaluation***

- The previous example assumes that all the information is available to a central server which can perform a complete compilation
- This is not scalable
- Some parameters change dynamically - for example if a server node fails, the set of available servers will change
- Can we compile the same declarative constraints into a form which can be delegated to execution by P2P protocols?

# ***Provenance & authorisation***

- Especially if the system is making complex decisions based on constraints, it is very important to understand the derivation of the parameters
  - Why is the departmental web server running on my laptop?
  - This is similar to the data provenance issue
- We also want to establish parameter-level authorisation
  - How do we determine the set of people responsible for the value of a particular parameter?
  - This is very hard if the evaluation is distributed, because the configuration of the machine doing the evaluation needs to be considered

# ***Other issues***

## ■ Usability

- Usability appears frequently in the major problems cited by real users
- Real configurations are composed from many different levels and aspects, created by people with very different skill levels

## ■ Security

- Security is very difficult
- Under Unix, too many processes need to run as root

# *Some specific issues*



- Generating configurations from policies and high-level specifications
- Composing aspects
- Distributed evaluation
- Provenance and authorisation
- Usability
- Security

# ***Specification Languages for Fabric Configuration***

Paul Anderson  
<[dcspaul@inf.ed.ac.uk](mailto:dcspaul@inf.ed.ac.uk)>

University of Edinburgh

 School of  
**informatics**

