Configuration Management with Subversion, YAML and Perl Template Toolkit

Ray Miller
University of Oxford

SANE 2006
May 15th–19th, 2006

Abstract

In this paper we discuss the methods and tools used by the Systems Development and Support Team at Oxford University Computing Services to manage the installation and configuration of more than 60 Debian GNU/Linux servers, ensuring that these systems are in a consistent and reproducible state. We also give a brief overview of some of the existing software for configuration management and discuss the rationale and evolution of the system currently in use at Oxford. This consists of three tiers, treating generation, distribution, and installation of configuration files as independent processes.

Our toolkit is built from familiar free software components: Template Toolkit for configuration file generation; Subversion for revision control; rsync for file distribution; Perl for scripting; YAML for data serialisation. We introduce each of these technologies and describe how they fit together to provide a modular and flexible system for managing configuration files.

Introduction

The key to successful systems administration is *reproducibility.* If we can reproduce the current state of a system, then we can restore service in the event of a disaster and can rapidly roll out additional server capacity to meet growth in demand for services.

Very often, server installation and configuration has to be performed by a systems administrator working under extreme pressure—especially in a disaster recovery situation. Doing this accurately and quickly implies advance planning and a degree of automation.

Even in the normal course of events, systems administrators will make mistakes, and from time to time configuration changes have to be reverted.

Traditionally, Unix systems administrators have relied on system backups to recover old configuration files or to restore service in the event of a hardware failure. Back in the days when an institution might have one or two large multi-user computer systems, this strategy served us well. Now that we have 100-node compute clusters, classrooms full of workstations, and servers providing a diverse range of services, we need a more scalable and flexible solution.

A number of tools for automating system installation are available. These range from disk imaging solutions (e.g. SystemImager[1]) that can be used to clone a golden image of almost any operating system, to more flexible but distribution-specific tools (e.g. Jumpstart[2] for Solaris, Kickstart[3] for RedHat, FAI[4] for Debian).

Automated system installation solves only part of the problem. In the course of its lifetime, a server will undergo numerous configuration changes, software and operating system patches, and security updates. How do we manage this process?

Background and History

Here at the University of Oxford, the Systems Development and Support Team is responsible for the development, deployment, and ongoing maintenance of more than 60 servers. These include two 8-node clusters (one providing an IMAP mail store, the other a web/IMAP gateway) and a 4-node LDAP cluster, where the machines in each cluster are configured almost identically. The remaining systems are of a diverse nature with only the base configuration in common.

Apart from a handful of OpenBSD[5] firewalls, all of our systems are running the Debian GNU/Linux[6] operating system.

These systems are managed by a team of six, whose time is split between systems administration and development work. Although there is a degree of specialisation within the team, management and configuration of our production servers is a shared responsibility. Change control is vital in this environment: we need to know what changed, when it changed, who it was changed by, and why the change was made. There are also times when we need to revert a system's configuration to a previous state.

In 2000, we started to store system configuration files in a Perforce[7] repository. Once the files were under version control, we had to tackle the problem of installing them on client systems and keeping client systems synchronised with the version control server.

We imposed an additional requirement that a change in the Perforce repository should not be enough on its own to reconfigure dozens of client systems. This was achieved by introducing an additional step of pushing changes from the Perforce repository to an intermediate `rsync`[8] server, from which client systems pull their configuration data.

Another benefit of this design is that dependencies on the client systems are kept to a minimum (there is no need to install the Perforce client software across all systems). Furthermore, the clients will generally have a clean copy of their configuration data on local hard disk, enabling reconfiguration when they do not have network connectivity—or, by specifying a different repository path, for configuration against a repository stored on removable media.

On the client systems themselves, we run a simple Perl script that walks over the local copy of the configuration data and makes any necessary changes in the filesystem, performing pre-install and post-install operations as required (more on this later).

Configuration Management Software

Before implementing a custom solution, we considered several configuration management systems available at the time, including:

- Cfengine[9]
- PIKT[10]
- LCFG[11]

Each of these systems falls into one of two categories: *state-based* or *action-based*[1]. In an action-based system, the administrator specifies a number of actions to be carried out (and conditions under which particular actions are performed) to bring a system's configuration into the desired state; in a state-based system, the administrator specifies the final state, and the management software determines what actions are required to bring the system into the desired state.

Both Cfengine and PIKT are action-based systems. They provide a high-level language for specifying the actions, and agents for file distribution and scheduling.

[1]I am grateful to Alastair Scobie for his clear description of this fundamental difference

LCFG, on the other hand, is a state-based system. It provides a language for specifying a system's desired configuration, agents for notifying systems of changes, and component scripts for performing changes on clients. (See [12] for an overview of the LCFG architecture.)

LCFG is currently developed primarily for the RedHat Linux platform. While some parts of the system have been ported to Debian, these changes have not been incorporated and this is not a well supported platform. This makes LCFG a poor choice in our Debian-centric environment.

Although mature and well-documented, we ruled out PIKT at an early stage because of its reliance on a custom scripting language.

Cfengine does not introduce a new scripting language, but a configuration language based around the idea of *classes*, with actions performed based on a system's membership in a particular class. It is part of an ongoing research project, with extensive documentation and academic papers clearly linked from the web site[9]. Cfengine is a mature system, widely deployed and with a large user base.

A problem with action-based systems is that it can be very difficult to determine from the specified actions exactly what the final state of a given system will be. Further, if you unwittingly define contradictory actions, a system might oscillate between two different states.

This, and the requirement to run additional listening daemons on client systems, lead us to look elsewhere for a solution.

Our Solution

The system we developed is a state-based system, with the state expressed in the system configuration files. Our goals were:

- All configuration files should be under version control
- . . . but a change committed to the version control repository should not be enough to trigger updates of client systems
- Where possible, standard tools should be used
- Requirements for software running on client systems should be kept to a minimum (e.g. no additional listening daemons)
- The system should be "client-pull"

As described above, we introduced an intermediate `rsync` server for distributing configuration files. A simple Perl script, `configtool`, is run periodically on each of the client systems to update a local configuration cache and bring the system configuration into line with the version-controlled copy.

Because we enforced a separation between the client systems and the version control system, we have no dependency on the version control software. This enabled us in 2003 to switch seamlessly from Perforce to Subversion[14], an open source version control system designed to replace CVS.

Of course, simply putting configuration files under version control is not enough: we need a higher-level abstraction to manage these files. Often multiple systems will share the same configuration settings, or differ only in minor details. For example, systems on the same local network will share the same DNS resolvers, gateway, and netmask, while each having a unique IP address. Furthermore, these values may appear in multiple places: the address of an outgoing mail server will appear in each system's MTA configuration, but may also be required in a local `iptables` rule set or edge firewall. If the address of the mail server changes, we do not want to manually edit every file in which it occurs—how much better things would be if we could interpolate the value from a common source!

To address this requirement, we introduced a mechanism for generating configuration files from templates, initially using Perl's `Text::Template`[15] but eventually settling on Template Toolkit[16], which we were also using for other development projects. This paper does not go into a lot of detail about Template Toolkit; see [16, 17] for more information.

Configuration parameters are stored in YAML[18, 19] format. YAML is a data serialisation format designed to be human readable (YAML Ain't Markup Language). It can be used to represent data structures from most modern programming languages—including Perl, which is the language we have chosen for implementing our systems administration tools.

For example, the following YAML encodes a hash (associative array) whose values are either scalars or arrays of scalars:

```
---
hostname: server.example.com
system_administrators: [alice, bob, margaret]
ssh_allow_from:
  - alice-pc.example.com
  - bob-pc.example.com
```

A template for the `/etc/hosts.allow` file might include the following:

```
ssh: [% params.ssh_allow_from.join(', ') %]
```

With the above parameters, this would expand to:

```
ssh: alice-pc.example.com, bob-pc.example.com
```

(perhaps Margaret is only allowed to log in on the console).

To recap, our configuration management system consists of the following components:

1. A version control repository (Subversion)
2. A file distribution mechanism (rsync)
3. A templating system (Template Toolkit)
4. Data serialisation for template parameters (YAML)
5. The `rb3` script for generating system-specific files from templates and parameters (Perl)
6. The `configtool` script for updating a system's configuration (Perl)

The following diagram shows how it all fits together:

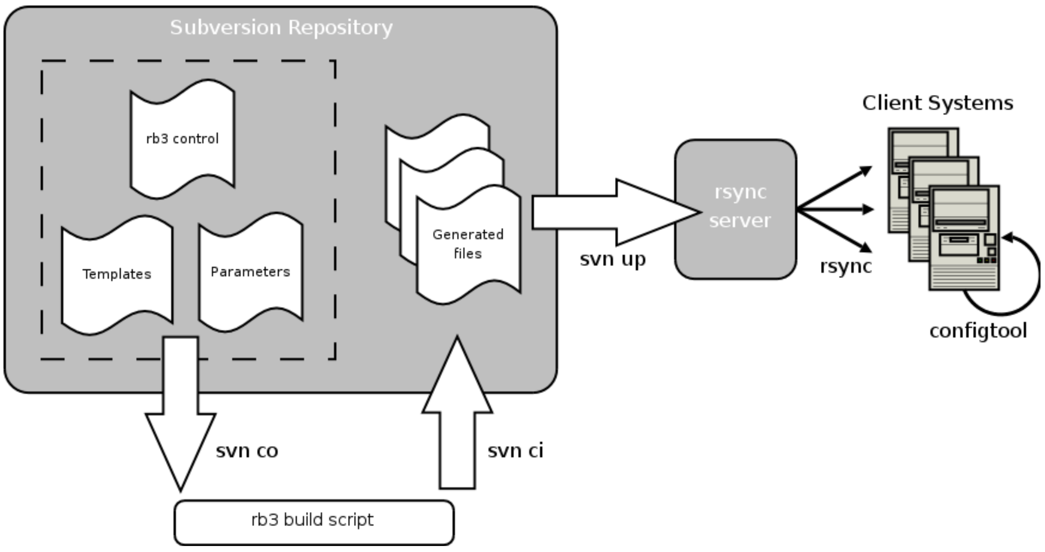

Figure 1: Configuration management architecture

Repository Builder

The YAML and Template Toolkit components are brought together by a *repository builder* script, `rb3`[2], which drives the generation of configuration files from a combination of global and system-specific configuration parameters. This system was originally inspired by Jon Finke's LISA '03 paper *Generating Configuration Files: The Directors Cut*[20], but with the goal of avoiding XML/XSLT and the overhead of a relational database.

Each client system has an `rb3` control file that directs configuration file generation. The syntax of this file is very straightforward (and deliberately terse), and is described more fully below.

Rather than storing configuration parameters in an SQL database, we store data structures directly in YAML files, also kept under version control.

`rb3` is a simple Perl script that parses a control file, builds a data structure of configuration parameters for that system (from the YAML data files), processes templates as directed, and writes output to the system's configuration root. We have added a `shuffle()` method to the Template Toolkit list operations, and a number of utility functions to facilitate filename, IP address, network and netmask manipulation in templates.

Repository Layout

The tool chain does not dictate a particular filesystem layout, but we have found that the following works well for us:

`defaults/` Default `rb3` configurations and parameters

`devel/` System repositories under development

`groups/` Sources, templates, `rb3` configurations and parameters for host groups (classes)

`sources/` Default sources and templates

`systems/` Live system repositories, control files, system-specific sources and parameters

The `systems/` directory contains one directory for each host under `rb3` control, named after the primary hostname. Each host directory has a `root/` subdirectory containing the files to be installed on that system.

[2]`rb3` because this is our third attempt at getting it right.

If a file named `config.rb3` exists in the host directory, then `rb3 --build` will generate files in that system's `root/`. The host directory may also contain a `params.yml` file or other miscellaneous data specific to that host.

It is possible to suppress `rb3` generation of particular files and add files directly to a host's `root/` directory instead. This is particularly useful during development and testing, or when a system diverges significantly from the norm and templating no longer makes sense.

Control File Syntax

The `rb3` control files are processed one line at a time, with the first character representing an action, and subsequent words (separated by whitespace) representing arguments to the action.

`#` indicates a comment; the rest of the line is ignored.

`=` specifies an `rb3` file to be included at this point; the single argument is the path of the included file.

`+` adds a generated file; the first argument is required, and specifies the install path of the generated file. The remaining arguments are optional:

- **owner:group** Specify the owner and group for the installed file (default `root:root`). One of owner or group may be omitted, so `owner:` and `:group` are also valid.
- **mode** Specify the mode of the installed file (default `0444`).
- **path** Specify source from which this file is generated. If the path ends in `.tt`, it will be processed via Template Toolkit, otherwise it is copied verbatim.
- **!params.yml** Specify a parameter override for this file only.

`-` suppress generation of a file; the single argument is the install path to suppress.

`!` indicates that a parameter specification file should be processed at this point; the single argument is the path of the parameter file, which should be in YAML format.

By default, the `rb3` build script looks for included files relative to the current directory, but this can be overridden with the `--basename` command-line option.

An included rb3 file can add or remove install paths, or override the templates used to generate some files. The entire control file and any included control files are parsed before file generation begins, allowing later control statements to override earlier ones (the last value read wins). An included parameter file can specify new parameters or override old ones; again, the last value read wins.

The following simple example demonstrates most of these features:

```
1 # A simple rb3 control file
2 =defaults/config.rb3
3 +/etc/resolv.conf sources/etc/resolv.conf.tt root:root 0444
4 -/etc/hosts
5 !systems/node1.example.com/params.yml
```

1. A comment; the entire line is ignored.
2. The `defaults/config.rb3` control file is processed at this point.
3. The template `sources/etc/resolv.conf.tt` will be processed and installed on this system as `/etc/resolv.conf` with owner `root`, group `root` and mode `0444` (the `root:root` and `0444` arguments could safely be omitted, as these are the defaults).
4. No `/etc/hosts` will be generated for this system (overriding any specification for `/etc/hosts` that might have been loaded from the default control file).
5. System-specific parameters are loaded.

With suitable use of defaults and group (class) configurations, the rb3 control file for our standard servers becomes very simple. For example, to deploy a new server in our webmail cluster, we need only the following two files to prime the configuration generation:

`systems/webmail224.herald.ox.ac.uk/control.rb3`

```
=defaults/config.rb3
=groups/webmail.ox.ac.uk/config.rb3
!systems/webmail224.herald.ox.ac.uk/params.yml
```

`systems/webmail224.herald.ox.ac.uk/params.yml`

```
---
apache2_instances_addrs:
  webmail: 163.1.0.224
network_interface: eth1
```

Configtool

`configtool` is a Perl script that runs on each client system to bring its configuration into line with that in its version-controlled repository. When invoked as `configtool --sync`, the local copy of the system configuration is updated via `rsync`. This skeleton filesystem is stored in `/var/lib/configtool`.

After refreshing the local copy of the configuration data, `configtool` (invoked with no arguments) will walk through the local configuration repository and, for each file in the repository, perform the following algorithm:

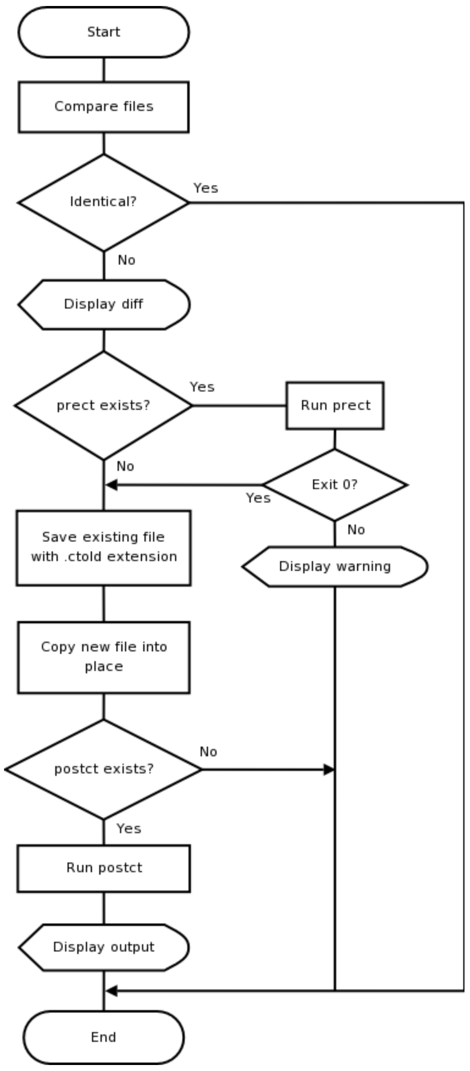

Figure 2: `configtool` processing

When processing a file in the configuration repository, `configtool` also looks for two supplementary files named after the configuration file but with a `.prect` or `.postct` suffix. These files are never installed in the filesystem, but allow us to perform optional processing when a configuration file changes.

If a matching file with suffix `.prect` exists, it is executed before a new configuration file is moved into place. If this script does not exit `0`, a warning is emitted and further processing of that file aborted. This allows us, for example, to check the syntax of a new configuration file before installing it.

If a matching file with suffix `.postct` exists, it is executed after a new configuration file has been moved into place. The exit code of this script has no effect on processing.

For example, the `exim` configuration file `/etc/exim/exim.conf` might have pre-install and post-install files as follows:

`/var/lib/configtool/etc/exim/exim.conf.prect`

```
#!/bin/sh
exim -C $1 -bV
```

(The single argument passed to this script is the path of the new configuration file.) The corresponding post-install script can be used to restart affected services:

`/var/lib/configtool/etc/exim/exim.conf.postct`

```
#!/bin/sh
invoke-rc.d exim reload
```

Each of our client systems is configured with an hourly cron job that runs `configtool --sync && configtool`, automatically picking up any new or modified configuration files and reporting changes back to us. Changes that cannot wait for the cron-initiated update may be made at any time by invoking the same command. Any output from the pre-install and post-install scripts, or from configtool itself, is written to the terminal during interactive operation, or captured by cron and emailed to the administrator.

The regular cron job picks up any anomalies introduced on a system, and discourages systems administrators from making manual configuration changes directly in the filesystem. Of course, there are times when changes *must* be made quickly, so we introduced the `--disable` command line option to temporarily disable `configtool`. The cron job still runs every hour, but now just reports periodically that `configtool` is disabled.

Package Management

The `rb3` and `configtool` scripts address the issues of configuration file management, but we have said nothing so far about package management—what software is installed where?

Almost all of our systems are running Debian GNU/Linux, so we have the APT[21, 22] package management tool at our disposal. We maintain a local APT repository for customised and back-ported packages, and make heavy use of a local Debian mirror.

To facilitate server deployment, we use APT's dependency and conflict functions, creating "meta-packages" that have no content but *depend* on required packages, and *conflict* with those to be deinstalled. For example, the `sysdev-server-base` package depends on `configtool`, `cron`, `dnsutils`, `make`, and a number of other packages we want installed on all of our servers.

Additional packages required for our IMAP mail store servers are specified as dependencies of the `sysdev-server-imap` package (which in turn depends on `sysdev-server-base`). Thus, when installing a new IMAP server, we need simply install the `sysdev-server-imap` package once the base install is complete.

We have encountered a number of problems with this approach. Although we can specify a rich set of dependencies and conflicts, we cannot easily control the order in which APT proceeds to install/remove packages. There is also no link between these meta-packages and the repository builder system, which can lead to duplication of information and reduces the potential for further automating server installation.

Another specific problem we have run into is with file ownership. Some files have to be installed with a particular user/group, but the required user/group might not exist at the time `configtool` runs. This might happen when the user/group is created by a package installation script and the package has not been installed yet. But if we were to install the package before running `configtool`, a network service might be started with the wrong configuration.

Future Development

In its original instantiation, `configtool` had no support for *removing* configuration files from a system. This deficiency was addressed recently in a new version with support for deleting files, directories, or symbolic links specified in a control file `/etc/configtool.delete`. The contents of this file can be generated on a per-system basis by the repository builder tool chain.

We are considering adding functionality to `configtool` to allow for more granular control over package installation and removal than is provided by our current meta-packages, and also to integrate management of local system users and groups to work around some of the package management issues mentioned above.

Another enhancement under consideration is to add an extra step in processing the `rb3` control files, to allow these files to be templated. This would replace the existing include mechanism (lines in the control file beginning with '`=`') with the more flexible `INCLUDE`, `PROCESS` and `INSERT` methods from Template Toolkit, and would allow for parameter substitution in the control file itself.

Getting the Software

The `configtool` and `rb3` scripts are released under the GNU General Public License[23], version 2, and are available for download from the author's web site[24].

Conclusion

Using only familiar tools and with relatively little custom code, we have deployed a modular system that solves part of the problem of configuration management. This enables us to keep all of our system configurations under version control, and to manage updates on client systems. This system has served us well for several years, but as the number and diversity of systems we manage grows its deficiencies are becoming more apparent.

It is often difficult to decide whether to invest in writing software, or in learning how to use someone else's (that might not do quite what you need). But a lot of research has been done in the field of configuration management, and this brings definite benefits: we can achieve much more by building on existing work. Mark Burgess, in his SANE 2000 paper[25], writes:

> In spite of USENIX/SAGE efforts, system administration is not yet a continuing research dialog, but more a cycle of reinvention of one-off solutions. If one is to avoid such reinvention and advance the state of the field, more introspection and criticism of the technologies at hand is needed.

This is good advice and, indeed, a number of new configuration management systems have come to our attention since we first surveyed this area. We plan to do a fuller investigation of currently available technology before investing substantial effort in further enhancements to our custom system. I conclude by drawing your attention to some of these systems.

The puppet[26] configuration management system looks particularly interesting, and is starting to attract some attention[27, 28]. There are a lot of good ideas to be drawn from the Arusha project[29], whose development to date has been slow but is certainly worth watching. Bcfg2[30, 31, 32] offers a mature system already in use at a number of large sites, as does Quattor[33], originally developed for the European Data Grid. Of course, we should not forget the two systems mentioned earlier in this paper—LCFG and Cfengine, which are perhaps the best-known applications in this arena.

Both puppet and Cfengine offer interesting possibilities for replacing the `configtool` functionality, and could work in tandem with our existing repository builder scripts. Tobias Oetiker has taken this approach with his Template Tree II system[34, 35], which uses Cfengine as the transport mechanism.

We will certainly be taking a closer look at all of these systems, with a view to integrating one or more into our environment and contributing enhancements back to the community.

References

[1] SystemImager home page, `<http://www.systemimager.org/>`.

[2] Amorin, K. *Solaris Jumpstart Automated Installation*, `<http://www.amorin.org/professional/jumpstart.php>`.

[3] *What are Kickstart Installations?*, Red Hat Enterprise Linux 4: System Administration Guide, `<http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/sysadmin-guide/ch-kickstart2.html>`.

[4] FAI (Fully Automated Install) for Debian home page, `<http://www.informatik.uni-koeln.de/fai/>`.

[5] OpenBSD Project home page, `<http://www.openbsd.org/>`.

[6] Debian Project home page, `<http://www.debian.org/>`.

[7] Perforce home page, `<http://www.perforce.com/>`.

[8] rsync home page, `<http://www.samba.org/rsync/>`.

[9] Cfengine home page, `<http://www.cfengine.org/>`.

[10] PIKT home page, `<http://pikt.org/>`.

[11] LCFG home page, `<http://www.lcfg.org/>`.

[12] Anderson, P & Scobie, A. *LCFG: The Next Generation*, UKUUG Winter Conference 2002, `<http://www.lcfg.org/doc/ukuug2002.pdf>`.

[13] Burgess, M. *A Tiny Overview of Cfengine: Convergent Maintenance Agent*, Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems, MARS/ICINCO 2005, `<http://www.iu.hio.no/~mark/papers/tiny_intro.pdf>`.

[14] Subversion home page, `<http://subversion.tigris.org/>`.

[15] `Text::Template` home page, `<http://perl.plover.com/Template/>`.

[16] Template Toolkit home page, `<http://www.template-toolkit.org/>`.

[17] Wardley, A. Chamberlain, D. & Cross, D. *Perl Template Tookit*, First Edition, December 2003, ISBN: 0-596-00476-1.

[18] YAML home page, `<http://www.yaml.org/>`.

[19] YAML 1.0 Specification, `<http://www.yaml.org/spec/>`.

[20] Finke, J. *Generating Configuration Files: The Directors Cut*, pp. 195–204 of the Proceedings of LISA '03: Seventeenth Large Installation Systems Administration Conference (San Diego, CA: USENIX Association, October 2003), `<http://www.usenix.org/events/lisa03/tech/finke.html>`.

[21] Debian APT, `<http://packages.debian.org/stable/base/apt>`.

[22] APT how-to, `<http://www.debian.org/doc/manuals/apt-howto/>`.

[23] GNU General Public License, `<http://www.gnu.org/copyleft/gpl.html>`.

[24] `configtool` and `rb3` downloads, `<http://users.ox.ac.uk/~raym/software/configuration-management/>`.

[25] Burgess, M. *Evaluating cfengine's immunity model of site maintenance*, Proceedings of the 2nd SANE system administration conference (USENIX/NLUUG) 2000, `<http://www.iu.hio.no/~mark/papers/sane2000.pdf>`.

[26] Puppet home page, `<http://reductivelabs.com/projects/puppet>`.

[27] Kanies, L. *Next-Generation Configuration Management*, pp. 19–25, USENIX ;login: February 2006, vol 31, number 1, `<http://www.usenix.org/publications/login/2006-02/pdfs/kanies.pdf>`.

[28] Lutter, D. *System recipes and configuration management*, <http://people.redhat.com/dlutter/puppet-app.html>.

[29] Arusha Project home page, <http://ark.sourceforge.net/>.

[30] Bcfg2 home page, <http://trac.mcs.anl.gov/projects/bcfg2/>.

[31] Desai, N et al. *A Case Study in Configuration Management Tool Deployment*, pp. 39–46 of the Proceedings of LISA '05: Nineteenth Large Installation Systems Administration Conference (San Diego, CA: USENIX Association, December 2005), <http://www.usenix.org/events/lisa05/tech/desai.html>.

[32] Desai, N. et al. *System Management Methodologies with Bcfg2*, pp. 11–18, USENIX ;login: February 2006, vol 31, number 1, <http://www.usenix.org/publications/login/2006-02/pdfs/desai.pdf>.

[33] Quattor home page, <http://quattor.web.cern.ch/quattor/>.

[34] TemplateTree II home page, <http://isg.ee.ethz.ch/tools/tetre2/>.

[35] Oetiker, T. *TemplateTree II: The Post-Installation Setup Tool*, pp. 179–185 of the Proceedings of LISA '01: Fifteenth Large Installation Systems Administration Conference (San Diego, CA: USENIX Association, December 2001), <http://www.usenix.org/publications/library/proceedings/lisa2001/tech/oetiker.html>.