Large Scale Linux Configuration with LCFG

Paul Anderson
Division of Informatics, University of Edinburgh
paul@dcs.ed.ac.uk

Alastair Scobie
Division of Informatics, University of Edinburgh
ajs@dcs.ed.ac.uk

This paper describes the automatic installation and configuration system currently being used to manage several hundred Linux machines in the Division of Informatics at Edinburgh University. This is a development of the LCFG system which has been used successfully for several years under Solaris. The introduction provides some background on the general problem of large-scale configuration, together with a short comparison of typical solutions, and a brief description of the original LCFG system.

The specific changes required to support Linux are then discussed; in particular, the issues of installation bootstrapping, and the *updaterpms* program. This automatically synchronises client software packages with a specification in the central database. We describe how the system is used in practice, and how it enables us to automatically maintain large numbers of machines with very diverse and evolving configurations.

Some future plans are then discussed, including a major reworking of the LCFG implementation, LDAP integration, and our intention to make the technology more widely available.

1 Background

For many years, the Unix community has recognised the inadequacy of vendor-supplied configuration tools for managing large networks of disparate machines. A wide range of solutions have been proposed and developed by systems administrators, frequently just for their own use. These range from simple cloning mechanisms to highly flexible systems (a survey of some previous techniques is available in [1]).

The LCFG framework [3] was developed by The Department of Computer Science at Edinburgh University to handle their own network of several hundred (mostly Solaris) Unix machines. This system was designed to satisfy several fundamental properties that we could not find in any existing implementation, and has been very successful. Over the last three years, the site has migrated rapidly towards Linux, and the LCFG framework has been ported and extended to support this with good results. The Department has recently merged with several others to form the Division of Informatics and work is underway to extend the use of LCFG to the larger domain. As part of this work, we are taking the opportunity to improve the design and implementation in some areas.

2 System Configuration

When a standard release of a large software product, such as an operating system, is distributed to many users, it invariably needs *configuring* to tailor it to the requirements of each individual installation. Even within a single site, there can be a huge difference between configurations of different machines; the following are some examples of the parameters which may vary:

- Hardware configuration and drivers.
- Network configuration and servers.
- Installed software.
- Network services provided.
- Access control.

2.1 Manual Configuration

Obviously the amount of variety between machines depends on the type of installation; an academic Computer Science environment tends to have a larger variety of software and configurations than a site concerned primarily with a single application. Our current LCFG system supports over 2000 parameters, about 25% of which routinely vary between different systems. For individual machines, or a small site, these parameters would traditionally be configured manually, and most distributions include graphical tools to make this process more straightforward (for example [9, 17]). However, large installations require more sophisticated techniques:

2.2 Automatic Configuration

Most obviously, the effort required to configure several hundred machines manually from a graphical interface is not normally acceptable. However, most sites will also want to have a reasonable confidence in the correctness of their configurations; misconfigured systems represent serious security problems, as well as leading to unpredictable failures. Manually configured systems, with no explicit representation of the configuration, are notoriously difficult to guarantee correct. Early attempts to overcome these problems were often based on a *cloning* procedure where a single machine is configured by hand and the resulting disk image is copied directly onto a set of other machines. This is usually followed by execution of some scripts to apply any machine-specific differences. This process is useful for large numbers of very similar machines which do not change regularly, such as those in a student laboratory. It is also widely used in Windows environments.

In many installations, such as our own, both the variety of different configurations, and the rate at which they change, makes cloning impractical. We support a range of machines from file servers, to student laboratory clients, to researcher's laptops, and the hardware and software requirements are all very different. New machines arrive continuously, old machines are reallocated, and systems are rebuilt after hardware failures or OS upgrades; all of these imply a reconfiguration, and we estimate that, on average, about 10% of our machines are completely reconfigured each week. Small configuration changes also occur very frequently in a complex environment; for example, changing a server or gateway can imply configuration changes for many other hosts. Software updates also occur at an average rate of several tens of packages per day.

2.3 Supporting Diversity & Change

Automatically supporting such diversity and rate of change requires two main features from a configuration system: Firstly, there must be some representation of the configuration information which is stored independently of the host systems. This may be simple copies of machine configuration files stored on a central server, or it may be a more complex "database". Secondly, the system must be capable of tracking changes to the configuration and applying them to individual machines as necessary, rather than requiring an explicit reconfiguration operation. *cfengine* [7, 6] is a popular system which addresses this problem.

Both the format and the structure of the configuration information are very important. An obvious solution is to store configuration files in the same format as they appear on the target host, but this coarse grain approach means that the configuration system is not aware of the relationships between data in the various files; for example, the "owner" of a machine may have special access rights which involves the username appearing in several different configuration files, and we would like to be able to change this at a single point. Storing the configuration specifications in a more abstract format allows the configuration to be examined and manipulated in a more meaningful way. It also provides a degree of platform independence, analogous to using a high-level programming language which can be transformed into code for any specific platform. The configuration system may use this information to generate traditional configuration files or the services on the host machine may be modified to read this configuration format directly.

An object-oriented structure is usually the most convenient way of organising the configuration information. Hosts usually fall into various different categories (laptop, web server, student desktop, etc.) and it is natural to specify new machines by using inheritance to describe just the difference (if any) between the new machine and some existing category. Many systems adopt this approach with varying degrees of sophistication (for example [15, 14, 16]). Once the information is available in this high-level form, there is considerable potential for analysing and generating the specification automatically. This opens up the possibility of treating the configuration of a whole site as a complete entity; for example, we should be able to prevent a gateway being removed while there are still clients which depend on it.

The process of actually changing a machine configuration to match a particular specification is not normally straightforward. Ideally, we would like this to take place automatically as soon as the specification is changed. Sometimes this is possible; for example changing the an

entry in a TCP wrapper. However, changing the disk partitioning is probably not desirable, or even possible, while a machine is in use. In practice, changes take place at different times, as appropriate; sometimes immediately, sometimes from a nightly cron job and sometimes at reboot. Laptops are an interesting case because they can normally only be reconfigured while they are connected to the network, and this might not happen very often at boot time, or at the time when a nightly cron job would normally run. We allow laptops to be reconfigured on demand by the user so that updates take place at a convenient time.

3 LCFG

LCFG was designed to handle automated installation and configuration in a very diverse and evolving environment. Abstract configuration parameters are stored in a central repository where they are organised in files based on machine categories. A simple inclusion mechanism provides inheritance, as well as a form of modified inheritance which we call *mutation*. The centralised configuration repository and the abstract representation of configuration parameters are key features of LCFG.

A collection of scripts on the host machine read these configuration parameters and either generate traditional configuration files, or directly manipulate various services.

3.1 Configuration Parameters

The configuration parameters are stored in the form of key-value pairs, inspired by X resources, and similar to the parameters used by COAS. For example:

```
mambo.dns.servers localhost
```

The key specifies the hostname, the *subsystem* and the parameter. The configuration files are passed through the C preprocessor, supporting simple inheritance by file inclusion:

```
#include <standard_laptop.h>
auth.owner paul
....
```

The machine-specific file need only list those resources which are different from a standard laptop (the first component of the resource keys is generated automatically from the name of the file). Notice that the included *class* file relates to a high-level concept (standard laptop machine) which may contain resource specifications for many different low-level subsystems. The class files may of course be nested.

Together with the use of preprocessor variables, this simple mechanism provides a powerful way of presenting complex host configurations in a clear way, with very little specialised software. However, it is not sufficiently fine-grained to process information inside the resource keys. This causes difficulties in some cases; for example, a standard configuration might specify that the CD-ROM should have its ownership changed to match that of the user at the console:

```
auth.consolepermclass_cdrom
  /dev/cdrom
```

If we have a second SCSI CD on our machine then we might want to specify:

```
auth.consolepermclass_cdrom
  /dev/cdrom /dev/scd0
```

Specifying this directly for a particular host is not good, because it overrides the specification for the standard machine, so that changes to the standard specification (such as changing the location of the default CD-ROM) would not be reflected on this particular host.

At the same time as porting LCFG to Linux, we added the facility to specify a regular expression for transforming any inherited resource value. This allows us to easily append or prepend items to a standard value:

```
auth.consolepermclass_cdrom
  !/(.*)/\$1 /dev/scd0
```

We call this process *mutation*. Although it is very powerful, overuse can lead to configurations which are very hard to understand, and we usually restrict its use to a few well-defined macros:

```
auth.consolepermclass_cdrom
  ANDALSO(/dev/scd0)
```

The files containing the configuration parameters for the entire installation are maintained on a central server under RCS control. When changes are made, the parameters are preprocessed into a single table which is distributed to the clients as an NIS map. This provides a replicated database which is easily accessible to all machines and simple to implement, but it was only originally intended as a temporary solution and it has a number of problems. These problems and a possible replacement technology are discussed later (section 6). Since this configuration information must be available at boot time, laptops are all configured as NIS slaves which update their maps on demand.

3.2 Subsystem Scripts

Each *subsystem* on a host has a controlling script which is very similar to the startup scripts used by the System V init mechanism. These scripts accept a number of *methods* such as `start` and `stop` which are invoked at appropriate times. Each script reads configuration parameters from the repository and configures the appropriate subsystem. This may involve translating the configuration parameters into a traditional configuration file, or controlling a service directly; for example, starting some daemon with command-line parameters derived from the configuration resources.

Traditionally, these scripts have been simple shell scripts although, under Linux, Perl is available at install time and many scripts now include some Perl code as well. A set of default routines are available which can be included into the script to load resources and perform other utility functions, such as retrieving default values for missing resources. Most scripts are quite short and it is easy to write a configuration script for a new package or service. Initially, new scripts are normally written just to support the subset of configuration parameters which are expected to vary in our installation, and this is easily expanded later as the need arises. The independence, and ease with which these scripts can be created has been a major reason for the success of the system; they are usually created by the person responsible for the corresponding service, and many people have contributed.

The example in section 8 shows a section of code from a script which starts the Samba server for VMware. Notice that most of the numerous Samba parameters are hardwired into a template configuration file, but those which we expect to vary between machines are set from the LCFG resources by a simple substitution of variables in the template. Some of these are also generated from a "higher-level" LCFG parameter which specifies whether or not we want the server to be visible on the external network. Notice also, that VMware itself is started using the standard init script.

3.3 Script Execution

As mentioned earlier, it is not always obvious when a particular service should be reconfigured. Should a daemon be stopped and restarted if necessary to force a reconfiguration? Or should we restart it during the night? Or should we wait until the next reboot? In practice, we never force a reconfiguration immediately whenever a specification changes (although this would not be hard to do). Those services which can be reconfigured while a machine is running, are normally reconfigured nightly by a cron job. Other services are reconfigured at boot time. It is possible for configuration changes to schedule a night-time reboot if it is essential that they are implemented as soon as possible.

A simple client-server application (`om`/`omd`) allows scripts on any remote machine (or group of machines) to be executed manually.

3.3.1 The boot Script

A subsystem script called `boot` is invoked both from the system init scripts, and from a regular `cron` job. This inspects LCFG resources to determine which other subsystems should be run at each stage. The `start` and `stop` methods for each specified script are called by `boot` at changes in the runlevel as specified by the appropriate resource. The `run` method is called at specified intervals, using `cron`. The resources can therefore determine which subsystems are reconfigured, and when.

3.3.2 The update Script

The `update` script controls the updating of software packages. This is described in more detail below and completely replaces the `updatelf` script previously used under Solaris to update software using `lfu` [2]. `update` is also capable of running at install time when it uses LCFG resources to configure those aspects of the machine which are too difficult to change while the system is running; for example, the primary network address and the disk partitioning. The installation process is described in more detail below (section 4.1).

4 Linux-Specific Issues

A number of small improvements were made to various aspects of LCFG when porting to Linux, and two areas were changed significantly:

- The installation bootstrapping process has been changed to accommodate the differences in hardware, and to make use of the new updating mechanism:
- Updating of software uses a completely different technique to takes advantage of the Redhat Package Manager (RPM) [4] and the wide availability of packages in this format.

4.1 Installation Bootstrapping

Providing the ability to install new machines with the absolute minimum of manual intervention is very impor-

tant. This allows failed machines to be replaced, and new machines to be installed, quickly and correctly, by unskilled staff [1].

Installing an operating system on to bare hardware requires some sort of bootstrap process. Typically:

- A minimal version of the operating system or other install program is loaded from the network, or removable media.
- Once booted, this program partitions the system disk and installs a copy of the operating system.
- There will usually be some additional software installation and configuration, the first time that the machine reboots from the newly installed system.

The first operation tends to be operating system-dependent and the original LCFG made use of Solaris Jumpstart [15, 14] to boot an initial image from the network. The current Linux port requires a boot floppy (or CD) for the same purpose, since not all hardware supports network booting (Kickstart [12] is not used).

Desktop machines may not have CD drives, and they use BOOTP/DHCP to mount the root filesystem from NFS. Laptops cannot use NFS so easily in this way because of the need for PCMCIA drivers, and they are installed using a bootable CD which contains an equivalent image. Once the system is booted, access to the network is necessary to retrieve the configuration parameters from the NIS and the RPMs for building the system disk.

When the minimal system has booted, an `update` script runs automatically to partition the system disk according to the LCFG resources and load the software. In the original Solaris implementation, a small hand-crafted image was first copied directly from the network onto the system disk, but this is difficult to maintain. Under Linux, the `update` script builds the root filesystem completely from a set of RPMs. Although it is aware of the context, this uses exactly the same process which is used nightly to update the software on a running machine (see 4.2.1), ensuring a consistent interpretation of the LCFG resources at install and update time.

When the software has been installed, the system reboots from the new image. The LCFG subsystem scripts start normally and perform the remaining configuration.

Once the install operation has been started, it runs completely unattended, allowing whole laboratories of machines to be installed easily by one person. However, the server load imposed by large numbers of clients performing simultaneous installations can be a performance problem. A "helper" CD is sometimes used to provide a local copy of many of the packages. If the CD becomes out of date, and newer versions of some packages are available on the network, the newer versions will automatically be installed instead. It would be interesting to look at ways of improving simultaneous large-scale installations, perhaps by using multicast to distribute the RPMs.

4.2 Software Updating

The original version of LCFG used a program called `lfu` [2] to update software on the local disks. At that time, clients tended to have smaller disks and mount much of their software from the network, so `lfu` was mainly concerned with synchronising a comparatively small number of replicated servers, and the performance would be poor for a large number of clients. There was also no obvious candidate for a package format which was widely supported by the packages that we wanted to install; `lfu` provides a crude mechanism for matching files to their packages, based on file ownership, but this was not always adequate. More importantly, it was also difficult to keep track of the link between binaries and their corresponding sources.

In theory, `lfu` could have been used under Linux, but the Redhat distribution is based on the RPM package management software which provides a much better mechanism for managing software packages. Many of the packages that we require are also distributed in this format.

Although there are now numerous programs available for updating and distributing RPMS (for example [13, 11, 5, 10]), few of these tools were available in 1997. We also wanted to interface with the LCFG and provide automatic installation, upgrade and deletion of RPMs. The Linux version of LCFG therefore uses a locally-developed program called `updaterpms`. This is currently written in C and makes use of `rpmlib` [2] (rewriting this in Python is a possibility now that there an `rpmlib` interface available).

4.2.1 updaterpms

`Updaterpms` compares the RPMs installed on a machine with a specification provided by the LCFG and installs, upgrades, or deletes RPMs as necessary. Using the current NIS implementation of the LCFG resource map, it would be unwieldy to hold the full list of RPMs in the

[1] This does not include the restoration of any user data from backup

[2] The latest version of `updaterpms` needs a modified version of `rpmlib` to support per-package options

map itself, so the lists are held in files which are referenced by LCFG resources. This is adequate since we tend to install most available software on most machines, but it is not ideal, and this may change as we move to a different technology for map distribution.

The package specification files are preprocessed with the C preprocessor to provide some degree of structure similar to the LCFG files themselves. Individual machines can therefore include a standard software specification and override individual packages. The RPM specifications may contain wildcards to refer to the latest version (or release); for example, the standard installation might include a specific version:

```
toshutils-1-1.34
```

And a particular machine might override that to carry the latest available version:

```
#include <standard>
+toshutils-1-*
```

The '+' symbol indicates that the new specification overrides any preceeding one thus inhibiting the error message that would normally be generated by the duplicate package specifications.

The ability to import software which has been prepackaged in RPM format saves a considerable amount of work. However the packaging is not always well implemented; post-install scripts, for example, are often poorly designed, perhaps attempting to add users to a password file, or to demand user interaction, neither of which are appropriate for an automated install on a networked system. Occasionally, dependency information is also incorrect. Several standard RPM options such as `--noscripts` or `--force` can be specified on a per-package basis to help with these problems.

Some other options are also available, for example:

- Ignore the RPM; do not attempt to delete or update it (`:i`). This is useful if an RPM has been installed manually (perhaps for testing) and should not be deleted by the automatic update.
- Schedule a reboot if this RPM is changed(`:r`). This is useful for important updates such as new kernel, which demand a reboot.

4.2.2 rpmsubmit

`updaterpms` obtains the the copies of the RPMs for installation from an NFS-mounted repository. This repository is replicated on several servers by an LCFG subsystem script which synchronises the copies before the nightly update.

A program called `rpmsubmit` allows authorised users to submit new RPMs into the master repository. This is capable of insisting on valid PGP signatures, and ensuring that corresponding source RPMs are available for all submitted RPMs. Currently, `rpmsubmit` uses NFS to copy the files into the repository, but this is not ideal because signature verification occurs on the submitting client; we would like to rewrite this an a client-server application.

5 LCFG in Action

This section shows how LCFG is used in practice during a number of common tasks; most normal client installations and system rebuilds are performed by junior technical staff:

5.1 Installing New Machines

A new host is installed by first creating an LCFG file specifying the configuration. Frequently, this contains only included classes and, if the machine is intended to be identical to an existing one, then the LCFG file for the existing machine can simply be copied. This example shows a typical configuration for a student laboratory machine:

```
#include <linuxdef.h>
#include <linux_rh62.h>
#include <linux_wire_at1.h>
#include <linux_cs1.h>
#include <dell_optiplex_g1.h>
```

The included classes define the machine as a Linux system, running Redhat 6.2, using servers on the Ethernet segment AT1, configured as a standard first year Computer Science undergraduate machine, and based on Dell G1 hardware.

At present, the Ethernet and IP addresses are entered separately into the NIS and DNS tables, however, there is no reason why these values could not be generated from the LCFG configuration file. The machine is then booted from an install floppy [3], and after a single warning prompt, it performs a full unattended install.

[3] Portables normally require a CD rather than a floppy because of the extra drivers required for the PCMCIA

5.2 Customising Machines

Individual machines can be customised simply by overriding default resources in the LCFG file. For example, to increase the logging level for VMware:

```
vmware.loglevel 3
```

Servers typically have more machine-specific resources than normal clients, but these are rarely more than about one page; groups of resources which perform a related function are usually collected together into a separate class file.

5.3 Rebuilding Machines

If a machine requires a rebuild after an OS corruption or hardware failure, it can be completely rebuilt simply be booting off the installation floppy. This is all that is required, even if hardware has been replaced, and even if the machine has been customised to a non-standard configuration.

5.4 Changing Server Configurations

If the class hierarchy is well constructed, it is straightforward to change all dependent client configurations automatically at the same time as a server configuration is changed. Because the configuration of the whole site is held in a single place, we can also identify possible problems in advance.

For example, at our site, the default DNS servers are set by a class file which depends on the Ethernet segment. If we want to remove a DNS server from a segment, we can simply remove it from the class file and replace it with another machine. This will be detected by all clients next time they reconfigure.

It is also very simple to inspect the DNS servers being used by all the clients. This allows us to check that our old DNS server is no longer in use, before physically removing it. One advantage of the crude NIS implementation of resource maps, is that people can simply type:

```
ypcat -k lcfg |grep dns.servers
```

5.5 Changing Security Policies

Many security-related parameters can be set by LCFG resources. Setting these in the appropriate class file allows the security configuration of whole groups of hosts to be manipulated. For example, we could control the ability to access all first year undergraduate machines from a remote `ssh` by setting the following in the appropriate class file:

```
inet.allow_sshd  ALL : rfc931
```

We are aware that this depends heavily on the security of the LCFG system itself which is currently not as strong as we would like. This is one area being addressed in the current re-implementation. Similar issues involving automatic configuration of security parameters are discussed in [8].

5.6 Upgrading Software

New software packages can be added simply by installing the RPMS into the central repository and adding them to the appropriate configuration file. They will then be installed onto all the corresponding machines overnight. Upgrading a package usually involves no more than copying the new version of the RPM into the repository (assuming the specification contains a wildcard).

To upgrade the operating system, a new set of base RPMs and an install floppy are needed. Once these have been prepared, hosts can be updated simply be changing the LCFG file to refer to the new class file and rebuilding the system by booting off the install floppy. The host will then rebuild with the new OS, but retaining any customised configuration previously in use. Changes in the operating system itself may require changes to some of the resources, however, the abstract nature of the resource means that such changes can often be avoided by changing the way in which the subsystem script interprets the resource.

5.7 Adding a New Subsystem

The modular nature of the LCFG scripts means that it is very easy to add a script to control a new subsystem, and this will often start with a copy of an existing script. The script should use the provided routines to load the required resources into shell variables. It might then start a daemon with this configuration, or it might simply generate a configuration file and allow something else to start any daemon (See example 8). The script itself would then be loaded onto all the necessary machines by including its RPM in the appropriate configurations, and it would be immediately available for use. Adding the subsystem to the boot resources would cause it to start automatically on the corresponding machines:

`boot.services ANDALSO(`*myservice*`)`
myservice.key value
myservice.key value

6 Future Plans

The basic concepts behind the LCFG system have proven very sound and the system has been extremely successful. However, a number of aspects of the implementation were not intended for wider large scale use. In particular:

- We would like a finer grained access control on the resource "database", so that we could delegate management more easily.
- We would like a more secure and efficient technology than NIS for distributing resource maps.
- We intend to rewrite the framework for constructing subsystem scripts using a more object-oriented approach, and a different language.

It seems likely that we will use LDAP as the configuration resource repository, and a Perl framework for the subsystem scripts.

We have also learned a good deal about the way in which sysadmins want to specify configurations and classes, and we intend to implement a custom language for describing machine configurations. The design of this language is still under discussion, but we would like to provide:

- Multiple inheritance and mutation.
- Some form of typing to allowing better validation for resource values.
- The ability to specify components (such as a disk configuration) which could be used by several machines (or several times by the same machine).

The current system is also tied closely to other local procedures, making it difficult to export, and we would like to address this, so that we can export it as open source (see below).

7 Availability

The actual amount of code in the LCFG system is comparatively small and we believe that the concepts are more significant than the implementation. However, we do intend to make the software available, and we would like to see it adopted more widely. This includes a number of components:

- The code which takes the LCFG files and transforms them into a single resource map.
- The common subroutines used by the subsystem scripts.
- The subsystem scripts themselves.
- The install subsystem.
- `updaterpms`.

As discussed above, the first four of these are currently being redesigned and we intend to release the new versions as soon as they are available. The current version of `updaterpms` is available via `http://www.dcs.ed.ac.uk/~ajs/`.

References

[1] Paul Anderson. System configuration and installation (SANS97).
`http://www.dcs.ed.ac.uk/home/-paul/publications/Config.pdf`.

[2] Paul Anderson. Managing program binaries in a heterogeneous unix network. In *Proceedings of the 5th Large Installations Systems Administration (LISA) Conference*, pages 1–9, Berkeley, CA, 1991. Usenix.
`http://www.dcs.ed.ac.uk/home/-paul/publications/LISA5_Paper.pdf`.

[3] Paul Anderson. Towards a high-level machine configuration system. In *Proceedings of the 8th Large Installations Systems Administration (LISA) Conference*, pages 19–26, Berkeley, CA, 1994. Usenix.
`http://www.dcs.ed.ac.uk/home/-paul/publications/LISA8_Paper.pdf`.

[4] Edward C Bailey. *Maximum RPM*. Redhat Software Inc.
`http://www.rpmdp.org/rpmbook/`.

[5] Kirk Bauer. autoRPM.
`http://www.kaybee.org/~kirk/html/-linux.html`.

[6] Mark Burgess. cfengine.
`http://www.iu.hioslo.no/cfengine/`.

[7] Mark Burgess. Computer immunology. In *Proceedings of the 12th Large Installations Systems Administration (LISA) Conference*, page 283, Berkeley, CA, 1998. Usenix.

[8] Mark Burgess. Managing network security with cfengine. *Login;*, pages 26–28, August 1999.

[9] Caldera. COAS. http://linux.davecentral.com/-3724_sysutiladmin.html.

[10] Yellog Dog. YUP. http://devel.yellowdoglinux.com/-rp_yup.shtml.

[11] Ken Estes. rpmsync. http://www.moongroup.com/RPM/00-05/msg00092.html.

[12] Martin Hamilton. RedHat Linux Kickstart HOWTO. http://metalab.unc.edu/pub/-Linux/docs/HOWTO/other-formats/-html_single/KickStart-HOWTO.html.

[13] Dirk Lutzebaeck. freshrpms. http://rpmfind.net/linux/RPM/-contrib/noarch/noarch/freshrpms-0.7.3-1.noarch.html.

[14] Scott McDermott. Introduction to Solaris Jumpstart. http://www.octaldream.com/scottm/-talks/jsintro/jsintro.htm.

[15] Sun Microsystems. Automatic installation. In *Solaris 2.3 system configuration and installation guide*. 1993.

[16] Nils Philippsen. RACE. http://www-stud.fht-esslingen.de/-race/.

[17] Solucorp. Linuxconf. http://www.solucorp.qc.ca/-linuxconf/.

8 Example: The Start Method of a VMware Script

```
# Fetch resource values
LoadResources subnet workgroup smblog external \
              printer encrypt loglevel external

# Allow external Samba connections?
if [ "$external" == "yes" ]; then
  nosa=";"
  hostname=‘hostname‘
  subnet=‘grep $hostname /etc/hosts |awk -F. ’{print $1"."$2"."$3}’‘
  xinterface="$subnet.0/255.255.255.0"
else
  nosa=""
  xinterface=""
fi

# Create configuration
sed <$smb_conf.tmpl >$smb_conf \
          -e "s@%SUBNET%@$subnet@g" \
          -e "s@%WORKGROUP%@$workgroup@g" \
          -e "s@%SMBLOG%@$smblog@g" \
          -e "s@%PRINTER%@$printer@g" \
          -e "s@%ENCRYPT%@$encrypt@g" \
          -e "s@%LOGLEVEL%@$loglevel@g" \
          -e "s@%NOSA%@$nosa@g" \
          -e "s@%XINTERFACE%@ $xinterface@g"

# Start VMware
/etc/rc.d/init.d/vmware start >>$logfile
```