University of Edinburgh
Division of Informatics

LCFG For Mac OS X

4th Year Project Report
Computer Science

Angus W Hardie

May 28, 2003

Abstract: LCFG is a large scale wide area configuration system. This project investigates the feasibility of implementing this system on the Mac OS X platform. The project involved the development of a working prototype of an LCFG client for Mac OS X. The results are discussed.

Acknowledgements

- Paul Anderson
- Carwyn Edwards

Lots of of other People who helped in various ways

Document Information

This document was made entirely with a Macintosh and Latex.

The latest version of this document should be available at

- http://homepages.inf.ed.ac.uk/s9812803/project/
- http://www.malcolmhardie.com/angus/project/

Contents

1 Introduction and Synopsis **1**
- 1.1 Introduction . . . 1
- 1.2 Motivation . . . 1
- 1.3 Mac OS X - Discussion and Description . . . 2
- 1.4 LCFG - Description and Discussion . . . 2
  - 1.4.1 How does LCFG work? . . . 3
- 1.5 Overview of Configuration Tools . . . 4
  - 1.5.1 Imaging or Scripting? . . . 4
  - 1.5.2 Declarative or Procedural? . . . 5
  - 1.5.3 Comparison . . . 5
- 1.6 Similar Systems and Existing Solutions . . . 5
- 1.7 Apple Mac OS X Server . . . 5
- 1.8 Rsync/RsyncX . . . 6
- 1.9 Radmind . . . 6
- 1.10 MacAdministrator . . . 7
- 1.11 Summary of Results . . . 7

2 Proposed solution **9**
- 2.1 Objectives . . . 9
- 2.2 Early Research and Prototypes . . . 9
- 2.3 Architectural Overview . . . 10
  - 2.3.1 Selection of Scope . . . 10
  - 2.3.2 Reuse . . . 11
- 2.4 Integration Overview . . . 11
- 2.5 Design Decisions . . . 12
  - 2.5.1 Separating Tools and Components . . . 12
  - 2.5.2 'The Macintosh Way' . . . 12
  - 2.5.3 Hardware & Software . . . 13
  - 2.5.4 Cohesiveness . . . 13
  - 2.5.5 Usability & Integration . . . 14
  - 2.5.6 Implementation . . . 14
- 2.6 Summary of Design and Plan . . . 14

3 Installation **17**
- 3.1 Installation . . . 17
  - 3.1.1 An Installation Package . . . 17
  - 3.1.2 Apple Software Restore (ASR) . . . 17
  - 3.1.3 A Bootable CD-ROM . . . 17
  - 3.1.4 Network Booting . . . 18

3.1.5 Installation Conclusion 18

4 Framework **19**
4.1 Porting From Linux 19
4.1.1 Limitations 19
4.1.2 Interpreted vs Compiled 19
4.1.3 File Locations - A Macintosh Like Directory Structure . . 20
4.1.4 Features Not Included 20
4.1.5 Component Writing 21
4.1.6 Perl Database Access 21
4.1.7 Standardisation 22
4.1.8 Minor Changes 22
4.1.9 Conclusion 22
4.2 The Deferred Task Manager 22
4.2.1 Problem 22
4.2.2 Solutions 22
4.2.3 Implementation 23

5 Components **25**
5.1 Components 25
5.1.1 Final Component Selection 25
5.2 Design of Key Components 26
5.3 Networking Component 26
5.3.1 The System Configuration Framework 26
5.3.2 Implemention 29
5.3.3 lcfgnetworktool 30
5.4 Machine Component 33
5.5 Display Component 33
5.5.1 Overview 33
5.5.2 lcfgdisplaytool 34
5.6 Netinfo Component 35
5.6.1 Design 36
5.6.2 Implemention 37
5.7 Preferences Component 38
5.8 Package Management Components 39
5.8.1 A Plug-In Approach 40
5.8.2 PKG - The Default Choice 40
5.8.3 APP - The Application Bundle 41
5.8.4 The lcfgbundletool 41
5.8.5 Extending the Profile. 42

6 Future Extensions **43**
6.1 Future Directions 43
6.1.1 Automated Profile Discovery with Rendezvous 43
6.1.2 Classic Environment Configuration 44

7 Conclusion **45**
7.1 Evaluation 45
7.2 Improvements 47
7.3 Level of Functionality Achieved 48
7.4 Review 49

A Source code for plist example **51**

B Glossary of Terms **53**

Bibliography **55**

1. Introduction and Synopsis

1.1 Introduction

The configuration of computers has become extremely important. With the increasing complexity and sophistication comes increasingly complicated configuration. A modern computer may have thousands of different configuration parameters. While the home user may never need to change these from the default and is also able to make the small number of changes that are required quite easily, in a large scale environment the time requirement could be immense. [4]

To reduce the burden on technical staff various tools have been developed to automatically configure networked computers.

LCFG (Local ConFiGuration) was developed at Edinburgh University to perfom this function. LCFG software is currently deployed across the Division of Informatics managing a large network of Linux machines.

This project had the aim of investigating the feasibility of extending LCFG support to the MacOS X platform.

1.2 Motivation

Configuring networks of computers is something that is becoming increasingly important with increasing connectivity and the development of grid computing. With new Apple hardware available in a rack mountable form and in a special cluster node version [25] there is increasing need for a scalable configuration system for Mac OS X. Some groups have been using Apple hardware for this purpose since before the release of Mac OS X [24] and others have been encouraged to switch. Groups such as the University of Colorado at Bolder are deploying large networks of XServe servers. [25] [1] LCFG could be used very effectively in this type of situation and in the absence of any similar technology being available for the Macintosh platform the need is clear.

[1] Although currently some of these servers are currently running Linux it is possible that they may decide to switch to Mac OS X in the future.

1.3 Mac OS X - Discussion and Description

Mac OS X is a descendent of the NextStep operating system developed by the Next corporation. It was based on the Mach kernel developed at Carnegie Mellon University.[33] Apple Computer acquired the Next corporation in December 1996 and NextStep became the basis for what was later to be called the Mac OS X operating system. Mac OS X retained the object orientated frameworks developed for NextStep under the name of *cocoa*. [13] Additions included a compatibility library called the *classic environment* that offered virtual machine support for earlier Macintosh software and a framework that allowed older software to be easily ported to the new operating system with minor changes called *carbon*. The core operating system has been released as open source[2] under the name *darwin* and is tracks the FreeBSD operating system quite closely.

1.4 LCFG - Description and Discussion

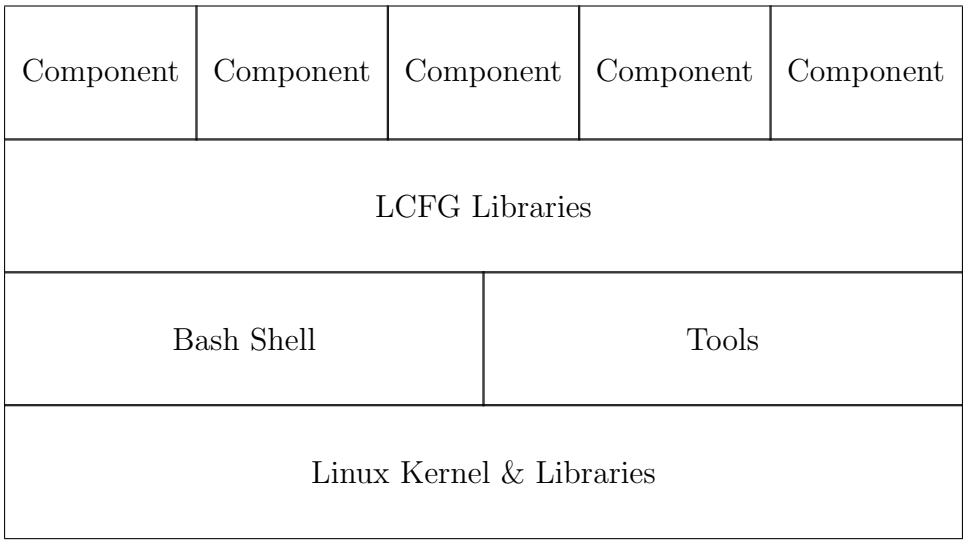

Figure 1.1: The overall architecture of the Linux LCFG client

LCFG is a component based declarative configuration framework[4]. Each machine has a profile that represents it.[2] The profile is created by combining various source files representing individual or shared aspects of the configuration into a single file. [1] This is then served to the client over the network. On the client this configuration is distributed to a number of components, each component being

[2]The upper layers of the system including the graphical interface and libraries are not open source.

responsible for configuring some aspect of the machine. Libraries provide common functionality between components such as reading and writing information from the profile or logging information.

It should be noted that the version of LCFG that is being discussed is the improved Next Generation (NG) version [5] rather than the original version. The main differences are that the NG version uses XML and HTTP transport while the original version used NIS as the transport method and stored the information in database files. [2]

Fig 1.1 shows the overall structure of an LCFG client as it runs on the Linux platform.

1.4.1 How does LCFG work?

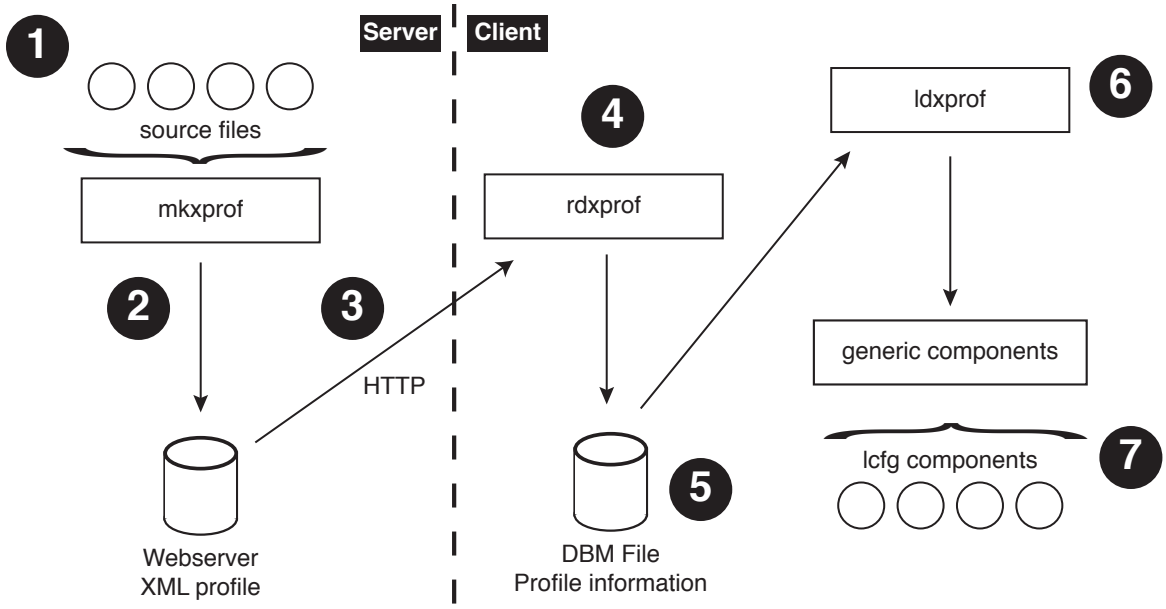

Figure 1.2: Flow of data within the LCFG system. The numbers correspond to sections in the main text. This image is based on Figure 1 from [2]

.

1. A number of source files are created. There is one primary source file for each physical machine. This machine source file can then include other more general source files for site or location specific information.

2. Then the profile is compiled into XML typically using the *mkxprof* tool. This tool handles dependencies and ordering and also includes a list of packages from a separate source that are required for the machine.

3. The profile is then copied to a web server. This allows them to be accessed by the client machines.
4. On the client machine a tool called *rdxprof* downloads the XML profile from the server.
5. The *rdxprof* tool then parses it into a local database which currently uses the DB format.
6. The profile can be loaded or accessed via the *ldxprof* tool.
7. Components can then access this information and configure the machine appropriately. To factor out common code there are currently two generic components (for Perl and Bash scripting). These generic components simplify the task of writing components by providing commonly used functions such as profile reading (via the *ldxprof* tool or directly from the database) and method dispatching (via the *lcfg object manager (om)* tool or directly).

1.5 Overview of Configuration Tools

There are a number of approaches to configuration tools for groups of computers. One way of categorising these systems is to divide them into two groups: cloning systems and scripted systems.[4]

1.5.1 Imaging or Scripting?

- *Cloning and Imaging Systems*

  With an imaging system a server hosts a complete representation of how the client machine should behave and be arranged. Regular comparisons are conducted between the client and the server and if there are differences on the client machine then it is restored to match the server image.

  Cloning systems are usually defined as imaging an entire disk at once.

  Examples include Radmind, Apple Net Install and Norton Ghost.

- *Scripting Tools*

  A scripted tool has a representation of the correct configuration of the machine. The tool then attempts to make the machine match this representation by altering settings and adding or removing packages. LCFG is a scripted tool.

1.5.2 Declarative or Procedural?

A declarative system has an explicit, declarative description of the required configuration. A procedural system simply specifies the steps necessary to configure the system. LCFG is a declarative system.[4]

1.5.3 Comparison

Imaging tools are inherently less complex than scripted tools. To make an image all that is needed is a machine that is operating in the desired way. The imaging tool then copies an image of this system to each client machine. This approach has advantages where the configuration is very stable but it can be resource intensive to deploy and changes are difficult to implement. Also in some cases the disk may not be the definitive source of information for configuration and altering a file in the hope of changing settings may fail unexpectedly, this is often the case with Mac OS X so some type of scripted tool must often be run in addition to the imaging tool.

To configure a scripted system it is necessary to declare all of the parameters of the required configuration and then make the target machine configure itself according to those parameters by interacting with the operating system and contents of the disk.

A scripted approach allows greater flexibility, less storage overhead and faster reconfiguration. But it is more difficult to set-up and the configurable parameters are limited to the subset that that are supported by the configuration system.

1.6 Similar Systems and Existing Solutions

There are several solutions to configuring networks of Macintosh computers, however most use a static image approach to configuration. It does not appear that any scripted and declarative configuration system like LCFG is available for Mac OS X, the closest is perhaps MacAdministrator.

1.7 Apple Mac OS X Server

Network install is provided as part of OS X Server [19]. It offers a simple image based software installation approach. The administrator defines an image on the server which is then replicated to machines across the network automatically.

There is a limit of 25 images per server which suggests that it is not a system for mass customisation of client machines [16]. Instead it is much more focused for providing an identical cloned image to a medium number of identical machines. The overhead of distributing a large image file make the system resource intensive. Another system that is supported by Mac OS X server is *Open Directory* [23]. This offers a directory based system for centrally managed configuration, although it has a focus of providing network centric configuration and services rather than configuring individual machines. It is not really in the same solution space.

1.8 Rsync/RsyncX

Some sites [8] have deployed Rsync, a disk cloning tool, as an approach to configuration management. This approach typically involves configuring a disk image and then replicating this to each machine in turn. Rsync provides for subsets of files and system configuration is handled by writing configuration files to the disk. RsyncX provides additional Macintosh specific features and includes a graphical user interface. Although this solution offers better performance than simple image cloning it still has limitations as a configuration tool. Altering system configuration files while the system is operating may lead to an unpredictable state. It is possible that RsyncX could be deployed in parallel to LCFG to handle file distribution in place of the package component. It is also possible that an LCFG component could be written or ported to manage RsynX.

1.9 Radmind

Radmind [29, 40] appears to be a popular Mac OS X imaging tool. Developed at the University of Michigan for general Unix configuration it has been further developed by the use of a graphical user interface that provides administrators with a complete and easy to use solution.

Radmind is a client-server imaging tool that updates files from a central location if they are found to be different. However there is no single source image, instead multiple image sets are combined to form the download for a particular machine. These image sets can be overlaid on each other to provide different configurations for different machines.[12]

Despite this sophistication, Radmind is still an imaging tool and has all of the advantages and disadvantages of similar tools.

1.10 MacAdministrator

Hi-Resolution systems' MacAdministrator is a system that has been widely deployed to manage networks of computers running Mac OS 9.x. [39]. The company has recently announced the development of a Mac OS X version. It appears that it does support basic procedural functionality with some declarative functionality; it supports a number of features that are specifically aimed at academic lab situations. However there are a number of problems. It is a port of an existing Mac OS 9.x application, this suggests that it may not fully support the Unix like features of Mac OS X. (A common problem with ported applications). If it is similar to the Mac OS 9.x version[3] it may also change whole aspects of the user experience which may lead to disorientation of the user. In addition the system is not extensible except by Applescript. While this technology has a number of benefits, it is not widely known outside of the Apple community and support is limited. MacAdministrator has a particular market niche with school and university laboratories running mainly Mac OS 9.x and moving towards OS X. If existing networks have been deployed using it, it may well be the best choice, however in the absence of a deployed network it may be better to consider other options.

1.11 Summary of Results

The LCFG client framework was sucessfully ported to the Mac OS X platform and was tested with Mac OS X 10.2.5 and 10.2.6. The server software was also ported to Mac OS X to allow for testing. It is believed that this software would also work on Mac OS X server (version 10.2+) although this has not been tested.

A number of components were developed including components to install packages, control the display, configure networking access, configure users and groups, control netinfo and edit application preferences. All of these components were original to this project and, although some similarities exist to equivalent components on other platforms (Linux), no code was shared between the equivalent components. This was mainly due to the widely differing approach required to configuring system services on the Macintosh.

A number of command line tools written in C++ were developed to handle interaction with the operating system. These tools may be of use separately from the LCFG/Mac OS X project.

[3]It was possible to inspect the Mac OS 9.x version, but it was not possible to inspect the OS X version due to the commercial nature of the application and the fact that it was still under test during the research phase of the project

The system has been used to configure two Macintosh computers running Mac OS X and is in current use.

2. Proposed solution

2.1 Objectives

The first thing with the project was to develop a set of objectives. These describe the outline of the project and help to determine relative priority of tasks within the project.

- *Research the feasibility of porting LCFG client technology to the Macintosh platform and investigate how this might be done*
- *Create a framework that can configure a Macintosh client computer and supports extension via components; use LCFG client technology if the first objective is feasible*
- *Develop a number of working components that allow basic configuration of the most important settings for a client Macintosh computer and represent examples for future expansion*
- *Integrate this system with both LCFG server technology and the Mac OS X operating system software to the best extent possible*
- *Document the system carefully to allow future expansion and further development of a Macintosh client as an ongoing effort*

2.2 Early Research and Prototypes

The first objective was to investigate the feasibility of porting the LCFG framework to the Macintosh OS X platform. To investigate this a series of small prototypes were developed to test various aspects of the final system. These prototypes were developed further and formed part of the final project code base.

It became clear early on that the LCFG framework was inherently portable to the Mac OS X platform. Both Perl [38] and Bash [26] scripting languages were supported and the library support seemed very good. Early tests of this proved successful and the framework was working at a basic level fairly early on in the project, although some bugs remained in less frequently used sections of the code until late in the development process.

Porting the components proved to be much more difficult. In most cases the concept of static files being read from disk to provide configuration information did not apply to core Mac OS X system services. Most of these relied on XML

preferences or on opaque data files with unspecified formats. This presented more of a challenge. As will be seen in later sections this lead to a significant change in approach. Instead of parsing the profile database and producing a static file, the database is parsed and command line tools are called that interact with system libraries to effect changes in the system services. There are some existing LCFG components that arrange things in this way too but it does not seem to be the preferred approach.

Package management proved a particular problem. [26, 6] The Mac OS X package format has unfortunate limitations, in particular uninstallation is unsupported and there are odd problems with symbolic links. Competing formats suffered fewer problems but had little or no popular support. One early thought was not to manage packages at all, however this was soon proved to be wrong. An alternative approach was to consider RPM format packages, however this was of little use in deploying Apple software, all of which is distributed in PKG format. This is further discussed in the section on components.

2.3 Architectural Overview

2.3.1 Selection of Scope

Given the broad scope that the project allowed it was necessary to identify areas to concentrate on. Developing a complete implementation of LCFG for Mac OS X was quickly deemed to be impossible. However it became clear that by implementing a number of basic components, a reasonable start could be made towards a more complete system later.

The main focus for the project is the development of an LCFG client. This client can download profiles from a server running Linux and then, by executing components, configure the machine, its applications and preferences.

Comparing this project against the discussion of "The Configuration Task" in [4] (pages 11-12) this project covers the following sections;

- Software Installation - This is supported to some extent as discussed in the "Installation" section
- Initial Configuration - This is supported by LCFG and the package installation approach
- Configuration Change - The main focus of the project
- Feedback and fault recovery - Feedback is supported under LCFG and fault recovery can be performed using the facilities for configuration change

One section in particular is already handled by the operating system without any requirement for intervention;

- Pre-install tasks - While the Open Firmware boot system is accessible and can be configured this is usually not done as a pre-install task. Much of the functionality that a IBM-PC compatable Linux machine uses the BIOS for is handled by the operating system itself or is automatically discovered at boot time. The settings could be configured using a suitable component after the system has been booted with the help of various Apple supplied command tools such as *nvram*.

  The only parameter that is likely to be used is the firmware password setting which would normally be set after installation is complete. Apple has spent substantial effort making the Open Firmware system invisible to users and has almost entirely succeeded.

2.3.2 Reuse

While this system is considered something of a prototype it is hoped that it may be converted into a more sophisticated, deployable system without significant extra work on the core framework. To do this it is important to ensure that the code written is reliable and well constructed. This will allow future versions to reuse part or all of this project as well as ensuring that this project is stable and reliable. It is intended that the system should follow the Dice guidelines [3] to ensure efficient operation in conjunction with existing LCFG components and systems.

2.4 Integration Overview

Fig 2.1 shows the overall architecture of the Mac OS X LCFG client. It is quite similar to fig 1.1 but it differs from that (Linux) version in that some of the lower levels have been changed. Note how there is a tools box in the second layer down. This represents tools that have been built specifically to effect changes for the top level components. These tools are an integral part of the project and will be described in detail with the associated components later in this report. This diagram also shows how the (future) extension to support the classic environment might work.

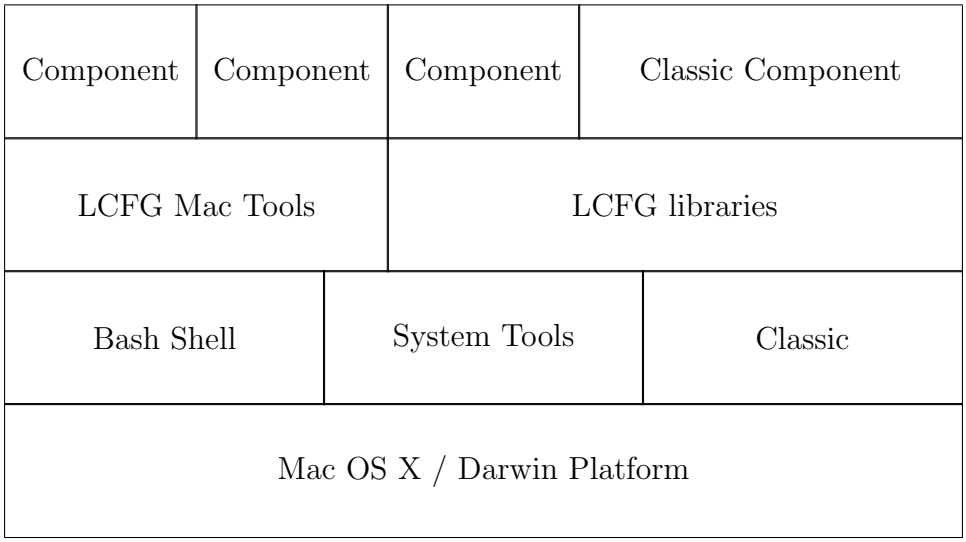

Figure 2.1: The overall architecture of the Mac OS X LCFG client

2.5 Design Decisions

2.5.1 Separating Tools and Components

During the development of the LCFG components a number of high level design decisions were made. One of the most important ones was to separate the functionality for configuring the system from the component itself. This meant in many cases the production of a command line tool (mostly in C++) to perform the configuration as well as production of an LCFG component (mostly in Perl).

The main reason for doing this was so that the effort invested in developing the tools could be used without the requirement for LCFG. This allows smaller installations to use the tools equally well. In some cases there are few tools available that can perform the actions that these tools can. This may be because there is little demand for them but alternatively it could be because the demand has not been identified. If the latter is true then it might be very useful to release these tools independently from the LCFG/Mac OS X release. This approach should not unduly complicate the project.

2.5.2 'The Macintosh Way'

One of the main difficulties for developers creating Macintosh software has always been the views of users. Macintosh users have always had strong views on the applications that they use. In some cases entire applications have failed commercially because users believed that they did not follow the 'Macintosh Way'.

Exactly how this might be defined is sometimes unclear but it appears to be a philosophical point rather than a technical issue. In technical terms the platform has never been greatly different from its competitors. The difference has been in the cohesiveness of the system, the integration of hardware and software, and the common way in which applications work.

2.5.3 Hardware & Software

Hardware integration on the Macintosh Platform is almost total. The Macintosh is a vertically integrated platform with Apple providing both the operating system and the hardware that runs on it. This has a number of benefits such as better control of the hardware (auto-discovery and hardware buttons that call software functions) and fewer hardware-software interface problems. The main disadvantage has always been cost, but this has often been disregarded by long time Macintosh users.

This does give a change in approach compared to the Linux version of LCFG. The Linux version might have hardware settings changed statically in the BIOS while the Macintosh version must reconfigure the hardware settings dynamically or rely on the system default values (which are usually correct in most cases anyway[1]). It also makes some components simpler in concept. If automatic hardware discovery is already supported then it is unnecessary to write software to discover it.

2.5.4 Cohesiveness

The key Macintosh advantage has always been in the way that users can understand it. Unlike other operating systems the Macintosh platform has only one appearance style. There are no competing window managers[2] and no themes[3]. This has a great advantage in consistency and it is easier to transfer understanding between programs.

However it does make life more difficult for the developer. The system and applications are *expected* to look and work in a particular way.[15] Failure to do this can lead to condemnation by reviewers and users.

Fortunately for this project, little of it will be visible to the User. Most aspects will be hidden within the Unix like operating system layer which is below the visible user interface layer. Although some unexpected reconfiguration may occur

[1] There are in general fewer commonly configurable options available
[2] Although X11 with a number of window managers is available
[3] Except as a third party addition

on occasion, most of the time LCFG/Mac OS X will be invisible to the User. Since LCFG is more similar to Unix than Macintosh this is probably a good thing.

2.5.5 Usability & Integration

Macintosh Users have always been very interested in usability. The historical strength and appeal of the Macintosh platform has been in ease of use and the use of system configuration software has sometimes conflicted with this. With Mac OS X there have been some complaints [36] in this area but it is often believed by many that it is easier to use than some other competing operating systems[35]. The system is designed to be easy to understand and easy to operate. Ease of use is of paramount concern when developing Macintosh [15] applications and is perhaps one of the most important non-functional requirements for this project. LCFG/Mac OS X must not degrade the usability of the system and the functionality of LCFG should not affect either performance or interface to the system.

Apple documentation [15] states that applications should:

> Allow the User, not the computer, to initiate and control actions.

Which presents an obvious problem for a tool that must do this. So there must be a balance. The LCFG system should perform its actions but not so that the User notices them. It must work behind the scenes to keep the machine running and allow the User to concentrate on their use of the machine.

2.5.6 Implementation

To achieve this requirement it was necessary to look at how the configuration changes are managed and timed. It is important not to change the system too greatly while the User is logged on. Doing so will give a disconcerting experience to the User and may cause instability of the system, particularly if packages are being replaced or upgraded. So it is useful to wait until the User has logged out before making changes. Please see the The Deferred Task Manager on page 22 for further discussion.

Integration is a definite challenge.

2.6 Summary of Design and Plan

1. Port the LCFG libraries which are written in Perl with some C++

2. Integrate the libraries with the operating system
3. Write components to manage a Macintosh client machine
4. Develop a method of installing the LCFG system to a new client Macintosh
5. Document the system

3. Installation

3.1 Installation

Installation is an area that is very platform specific but it is necessary to consider it as part of a complete configuration solution. To configure the machine the software is required to have been previously installed on that machine. So there must be a way of placing the software on the client machine by the use of an installation procedure.

Various options were considered for installation of the initial system.

3.1.1 An Installation Package

If the machine to be configured already has an installation of Mac OS X on it, then it is possible to install the software from a simple installation package. The LCFG initial settings are configured during the installation and the installer will prompt for a restart upon completion then machine will start LCFG along with other system services.

3.1.2 Apple Software Restore (ASR)

An alternative to using an installation package is a disk image cloning tool called Apple Software Restore. This approach is faster for a specific configuration as it follows the concept of the *gold image* [4] and it also has proven reliability (It is used by Apple for restoration CDs).

3.1.3 A Bootable CD-ROM

The simplest approach to installing a new machine with Mac OS X and LCFG is to produce a bootable CD-ROM containing a modified installation meta-package. [1] The Installer would then install the operating system and subsequently install the LCFG system as before. A piece of software called BootCD [37] is used to create a bootable CD-ROM or the process can be manually performed by creating a disk volume and copying files across.[42]

[1] A meta-package is an installable package that references other packages instead of containing files. It acts as a structure to manage more complicated multi-part installations

This method could also be used with a Firewire or USB external hard disk containing a bootable system image.

3.1.4 Network Booting

Network booting is the most complicated to configure but is the most efficient when in use. To use this approach a modified DHCP server is placed on the same nework subnet as the machine and configured to offer a boot image to connecting clients. The boot image contains a complete operating system in a read-only form and also some writeable space for storing data. The client Machintosh can then be booted by holding down the 'N' key on the keyboard. It will automatically discover the server and start up from the network. The process then continues as for the Bootable CD-ROM. It is also possible to use netbooting to run the machine for general use but this is a different problem.

Network boot is supported directly using under the Mac OS X Server operating system and this provides a user interface and user tools to configure the process.[19] It is also possible to do this using a Linux server. [10, 34]

3.1.5 Installation Conclusion

Netbooting is more efficient than a CD-ROM because it requires less intervention. However a CD-ROM requires less effort initially. Both methods use the same basic image, only the method of delivery is different so the final choice would depend on the circumstances and available equipment. (DHCP server, CD-ROM, Hard Disk, etc). The selection of installation hardware is of little importance but the development of the install image is considered part of the project and was quite simple to produce. A complete copy of Mac OS X was installed onto a prepared partition and it was configured to automatically install a complete Mac OS X installation and LCFG software onto specified disk when it booted. This image can then be deployed to any of the supported technologies without change.

It was decided that it was better to develop an approach with an installation package than to use Apple Software Restore. Apple Software Restore is a well documented technology and creating an image including LCFG would be relatively easy. Using ASR would involve replacing the Apple Installer based system with the ASR based system.

4. Framework

4.1 Porting From Linux

Porting the basic framework from Linux to Mac OS X was relatively easy although a number of areas did present difficulties. In addition some aspects were abandoned entirely due to lack of need.

The framework was mainly constructed in shell scripting tools (Bash) and Perl. Both of these are available on the Mac OS X platform so the main changes that needed to be made were in areas where the implementations differed slightly between platforms.

4.1.1 Limitations

The feature set that has been verified is not the complete feature set. Although the majority of features in the Linux implementation should work no guarantee is offered for features outside the scope of the project.

The server tools are also operational including the CGI functionality but with the client focus of the project these too are not guaranteed. The server has been in regular use during testing and it is believed to work.

4.1.2 Interpreted vs Compiled

The areas that proved most difficult with the framework were those that included compiled code. The LCFG-utils code included software to manage messages and contained some Perl modules and a command line tool called *lcfgmsg*. This proved quite challenging to port because of some major differences in the contents of the standard library on the Mac OS X platform. It was initially thought that the component would have to be re-written. However by changing the code slightly to use a different set of functions, it was possible to work around the library differences. This has the consequence that this version of *lcfgmsg* does not contain all of the functionality of the current Linux version which might cause issues with the porting of some components. However it does match the functionality of older versions and the extra arguments are ignored or mapped to other options so it should not cause any serious problems.

4.1.3 File Locations - A Macintosh Like Directory Structure

One major change that has been made is to move all of the files into a single directory. */Library/LCFG*. This directory contains almost all of the system including components and configuration files. It also includes all temporary files. There are some parts of the system outside of this directory, the directly executable scripts such as rdxprof and the Perl libraries. Also some of the *man* pages are stored outside in order to be in the *MANPATH* environment variable.

The main reason for this was that the Macintosh concept calls for a wholly encapsulated application if possible, this allows for easy installation and if required removal. It also reduces the likelihood that a future system update will accidentally overwrite an important file. With Mac OS X administrators have slightly less control over the system than they might under Linux. Apple often requires things to be done in a specific way and it occasionally happens that this way may conflict with some other piece of software, such as LCFG.

This was a relatively early decision in the project and in retrospect it may not have been entirely beneficial. Some time was spent reorganising files to fit this new scheme which might have been better spent on other things. It does make the whole thing more like a Macintosh Application but perhaps at the risk of adding confusion to people moving across from Linux. However it is possible to recompile and move the files back to conventional locations.

It is possible that a future version could encapsulate the entire system within an application bundle with a drag and drop install. This would provide a very easy install path. However this proved excessively complicated for the prototype. Companies using this type of install tend to have simple applications rather than complicated utilities that integrate into the system as LCFG/Mac OS X does.

4.1.4 Features Not Included

A number of components and features were of no relevance, being specific to some particular feature of Linux. The kernel component is an example. Other components were replaced entirely because of major differences between Mac OS X and Linux. The networking and display components are the main examples of the latter.

4.1.5 Component Writing

It is believed that components that do not make system specific assumptions should port to the Mac OS X version of LCFG without difficulty. This has not been extensively tested however and results may be unpredictable. It would be recommended to ensure that testing is carried out with any components that are to be ported. An example is the Apache component. While the basic functionality of the component is unchanged the locations of the configuration files are different and to ensure correct operation with the user interface some additional steps must be carried out, this might suggest a modified component or some sort of interface tool to work with the existing component.

Integration with the user interface was a major concern throughout the project. Unlike Linux where it often expected that the user will have to edit configuration files, with the Macintosh Platform this approach is generally discouraged except for very rarely used settings. The Apple developers have achieved a significant amount of separation between the user and the underlying operating system and many users may never need to use the command line tools or the terminal application at all.

However if this separation is broken then the user may become disorientated. So care must be taken that the user interface is updated to match the settings. With the components that were written for this project both the network and display components do achieve this most of the time. As is discussed later there is a slight issue with the *system preferences network pane* but other than that the networking component works with the user interface. The display component is also capable of updating the user interface correctly. The package component has some difficulties. In particular it is unable to update the software update system to indicate that packages have been installed. It is therefore recommended that this should be disabled. This appears to be a problem with the installation system rather than this project although the exact cause is still unknown.

4.1.6 Perl Database Access

There were initially some problems with database handling. In an early revision of the software, under earlier versions of the operating system it was not possible to access databases at all. However this was later traced to an obsolete database package that had been installed as part of the base system installation. The Apple supplied database DB package was an early version due to a difficulty with obtaining a distribution license for the latest version.[32] However there are no licensing difficulties with installing and using the latest version with this

project.[31] Updating the libdb to the latest version [1] solved these difficulties.

4.1.7 Standardisation

One thing that was done early on was to make the decision to support only one database format. This made things somewhat easier and reduced requirements for special casing of optional programs and libraries.

4.1.8 Minor Changes

There were some minor changes and bug fixes that made to various components most of which were Macintosh specific and very minor. There were some changes to the object manager (om) tool to change some of the locations that it uses. There were also some minor changes to fix some bugs that occurred specifically on the Macintosh platform.

4.1.9 Conclusion

Overall the porting of the framework went extremely well. This allowed efforts to focus on the writing of distinct Macintosh components.

4.2 The Deferred Task Manager

4.2.1 Problem

One addition that was made to the framework that was ported was the deferred task manager. Many of the components written for the LCFG/Mac OS X project use API calls directly. This was because in many cases it was not possible to do otherwise. However this does have the disadvantage that the commands or changes are executed immediately. If the change is substantial, this may cause a disruption to the User who may experience significant inconvenience.

4.2.2 Solutions

Various solutions were developed to solve this problem. The main thought behind the solutions was to limit the timeframe in which disruptive changes could occur.

[1] At the time of writing Version 4.1.25 from http://sleepycat.com/products/data.shtml

This would prevent applications being upgraded while people were using them or the screen size being switched in the middle of editing a document.

The first solution considered would be make all changes at system startup. This would prevent changes from happening when the user is logged in, however it has the disadvantage that if the machine is not rebooted then no changes will ever be made.

A better solution was to defer disruptive changes until the user logs out. Most users tend to log out at intervals so this offers a fairly good chance that the changes can be applied.

4.2.3 Implementation

To implement this feature it became clear that more infrastructure was required. To have each component that needed to be able to defer tasks independently re-implement this facility would be inefficient. The proposed design was a central list of tasks that would be automatically executed when the use logged out. The task required would be added to the list via a special tool and the system would call a different tool when the user logged out which would cause the tasks to execute sequentially.

The lcfg-kdm component uses a similar system of deferring actions until the user has logged out although that is less complicated and relies on the component being called. This approach is automatic and does not need the component to be running for the action to occur.

The structure of the task record is shown in Fig. 4.1

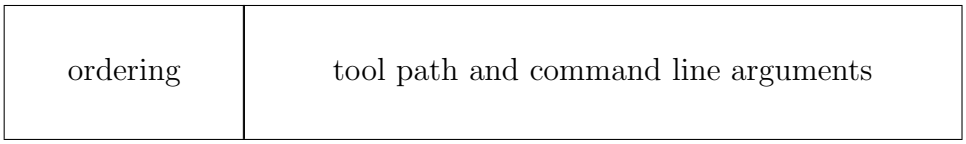

Figure 4.1: The structure of a task record

The ordering value represents the order in which the tasks should be executed. The range is between 0 and 998. It is recommended that each component should have a unique number allocated to it, but this is not essential. Numerically lower tasks are executed first, while numerically higher tasks are executed later. Two tasks with the same ordering value may be executed in the order in which they were added. In addition there is a special value of **999** [2] which is used for restart

[2]Using a special number is potentially questionable and perhaps should be changed in a future revision.

or shutdown requests and as such must always occur last. Only one task with ordering 999 is permitted and subsequent additions with ordering 999 will replace earlier ones.

It is assumed that there will be fewer than 999 requests for deferred tasks however there is no reason that this number should not be higher if needed.

Internally the values are stored using Perl in a **DB** file as an associative array. However it should always be accessed by the *lcfgdeferaction* tool. This allows encapsulation of the information and permits the replacement of the database with an XML or flat file in the future.

A second tool called *lcfglogout* is called by the system when the logout occurs. This is discussed in the Apple technical documentation [21] and requires a change to the */etc/ttys* file. This tool calls *lcfgdeferaction* with special arguments to cause it to execute any tasks in its deferred tasks database and then delete them from the database. Failure to delete the tasks will cause the actions to be repeated on the next logout because the database is persistent. This is desirable because if a task fails on one restart it may work the next time.

Once the script is run the machine then automatically continues with the logout procedure unless a restart has been requested in which case it will automatically restart. After the script has been executed the operating system will resume control and continue with the logout process unless a restart has been requested in which case it will automatically restart. None of these activities are visible to the user, although the logout process may be slightly extended by the time taken to execute the scripts. The other noticeable limitation of this system is that it is not currently possible to delete individual entries. The only option is to delete all entries in the task database. In a non-interactive environment it should not be necessary to delete existing entries.

5. Components

5.1 Components

LCFG is a component based system and the selection of components became very important to the project. Since there was not sufficient time to develop components to cover every aspect of the machine a priority was assigned to each aspect and components were written as required.

The main objectives for the components are task orientated and ranked in order of importance. Satisfing this list of objectives should allow a network of Macintoshes running Mac OS X and LCFG/Mac OS X to be configured to a resonable extent:

- Users and groups can be configured
- Applications and packages can be installed
- The active services can be configured
- Displays can be configured
- Networks can be configured
- Application preferences can be configured

5.1.1 Final Component Selection

The following components were selected and are described in further detail in subsequent chapters.

Network Component

This component allows for complete configuration of the networking settings of the machine. This was extremely important because without networking the system can't function at all. Configuring the network is also required as part of the initial installation

Display Component

This component allows for configuration of the monitor.

Package Management Component

This component allows for installation and possible removal of packages.

NetInfo Component

This component allows for configuration of the NetInfo database.

Preferences Component

This component allows for configuration of applications using XML or the Apple *defaults*[1] system. The application to configure does not have to be identified specifically except in the profile.

5.2 Design of Key Components

5.3 Networking Component

One of the most obvious components that was selected was the network component. Networking is fundamental to modern computing and it is required for full operation of LCFG.

5.3.1 The System Configuration Framework

Networking on the Mac OS X platform is quite interesting from a development point of view. The networking system has a concept of a location or set of preferences. Each set represents a mutually exclusive possible networking configuration. The user interface terminology for a set is a location, although both are used interchangeably.

Each set contains a number of service and the services exist in parallel with each other. Although they can be individually enabled and disabled, it is usual for all of the services within a set to be operating at one time but there is an ordering of priority for use. For instance, an ethernet service might be used in preference to a modem service if both were available. An example of a set is an ethernet

[1]The defaults system provides a common preferences system. This is described further in the preferences component information

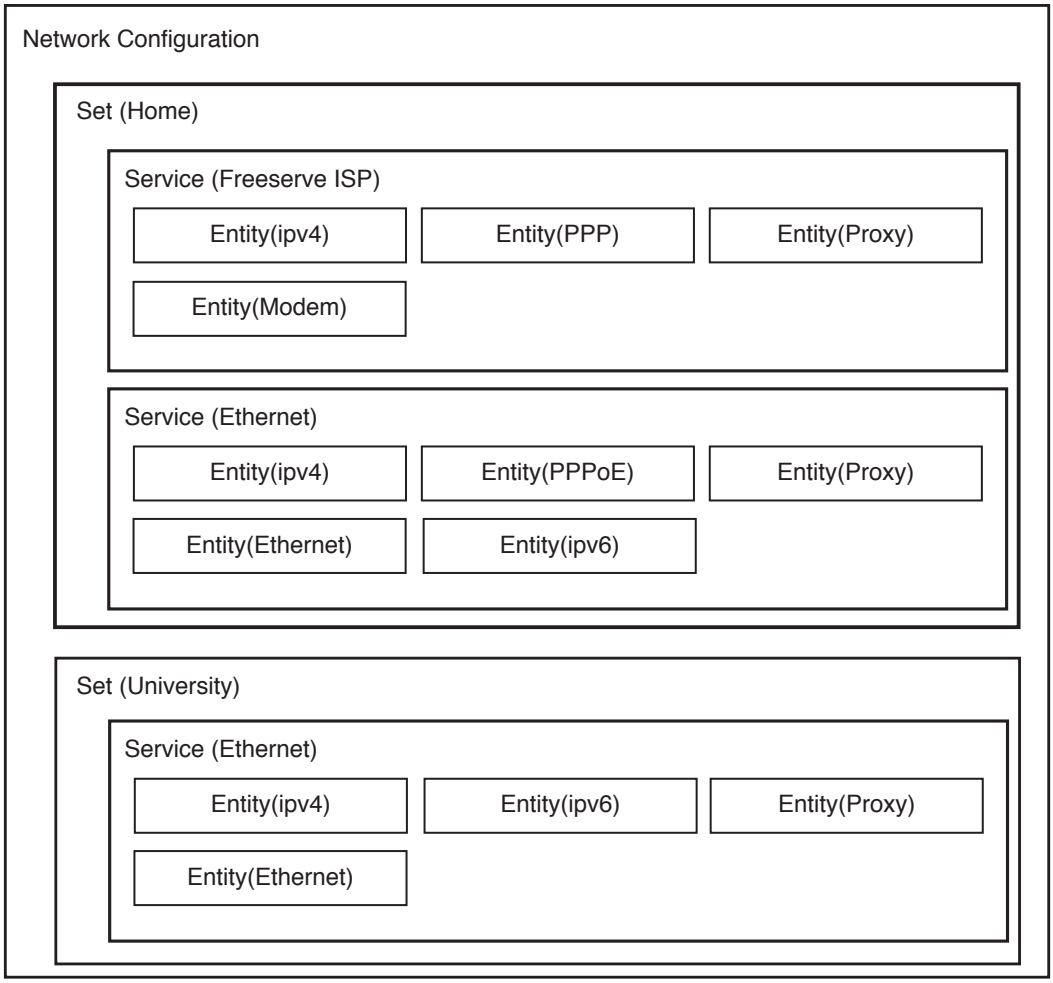

Figure 5.1: Diagram of the SystemConfiguration framework structure with support for multiple sets

interface configuration. It contains entities that then contain information such as the IP address of the interface, whether the address is obtained by DHCP or manually, the DNS servers that should be used to resolve domain names, settings for AppleTalk networking, and a facility to store settings for proxy servers.

Fig. 5.1 shows how an example of two distinct locations (sets) that contain different services that in turn contain entities representing aspects of the service.

Other examples include modem configurations and Airport (802.11) network configuration.

A point that is important to note is that for a specific type of data (e.g. IPv4 configuration) the entity will always be of the same format.

A service for a modem might have a modem configuration entity, a PPP entity, an IPv4 entity and a proxy entity. While a service representing an ethernet interface might have an identical IPv4 entity, an identical proxy entity but a different entity representing an ethernet adapter and an AppleTalk entity.

It is technically possible although not currently implemented by the user interface to connect together the same entity to more than one service, the current disk structure being more like a normalised database than a flat file. This breaks the analogy of the service containing entities.

This information is maintained centrally by the System Configuration Framework. Currently the actual data is then written to disk as an XML file.

This system has some similarities to the *divine* [41] system that is used under some Linux variants. Both support multiple sets of connections and can switch between them, however *divine* is more sophisticated in that it can automatically detect which set should be used whereas the system configuration framework must be told via the location menu. Fig 5.2 shows this menu.

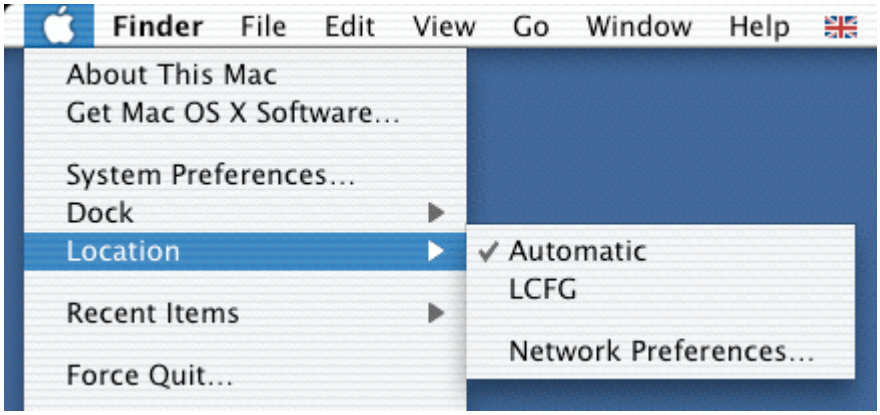

Figure 5.2: The Location Menu

5.3.2 Implemention

My first attempt at editing networking preferences used Perl to access the XML file directly. However this had a number of defects. The most significant one was that it was very disruptive to the system when the settings were changed. Because the system daemons assumed that they had complete control over the file, there were substantial issues with locking and with atomicity.

The locking problem occurred because there was no way to lock the file while writing to it. At any time the system might decide to change the settings or the user might access the control panel in the user interface and the file on the disk would be changed.

However the state would not be consistent. I found that if changes were made then the system configuration daemon might be maintaining one or more other copies in memory or attached to parts of the user interface. Then when these copies were written to disk the changes that had been made were lost Only by stopping the daemons completely could this problem be prevented and this had detrimental consequences for various aspects of the system, some of which assumed that the daemons would be available.

Fig 5.3 shows how different agents can access the store in parallel and shows how the disk based file store is not the primary source of information.

A new solution was needed. The next approach that was tried was to write the files only at startup and before the system configuration daemons had been started. From a technical point of view it was not particularly challenging because the startup script that the system configuration daemon used could easily be altered to make changes before starting the daemon. However this too had undesirable problems. The most significant was that, despite the system supporting dynamic changes to the configuration, this approach only supported changes at a restart.

AppleScript, a system level scripting tool, was briefly considered. Applescript was often used for configuring networking under Mac OS 9.x and earlier systems. However under Mac OS X it does not appear to be possible to use Applescript for this purpose without writing a library for the functionality in C or C++.

This approach was briefly considered, however it was concluded that the addition of Applescript to the component was unnecessary if such a library could be called from the Perl directly. The use of Applescript only increased the separation between the component and the configuration system.

Finally the correct approach was determined. I came across a utility called *ncutil* that promised to solve all of these problems. It had been lost and appeared unavailable. However a reference was found to a piece of sample code produced by

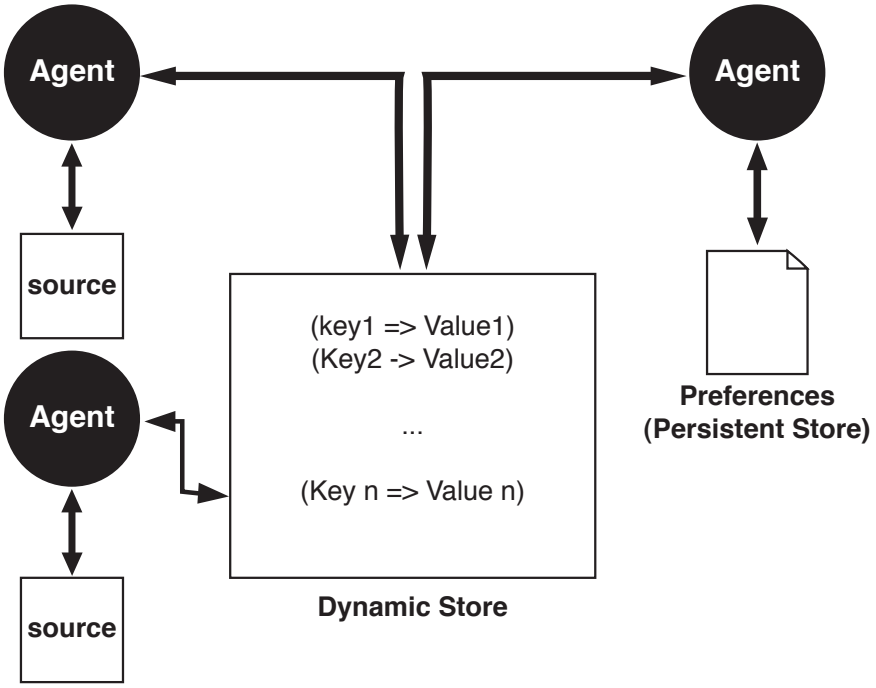

Figure 5.3: SystemConfiguration Framework outline showing interactions

Apple that allowed basic manipulation of the Networking system: The MoreSCF sample [22]. This framework, although not offering a direct solution, offered a number of useful functions that could be combined into a system to access the configuration daemon via its programmatic interface. This also offered the possibility of being able to change the settings without restarting.

5.3.3 lcfgnetworktool

There are a number of networking settings that may need to be configured Fig. 5.4 shows a screen shot of the Operating System's Network System Preferences Panel containing some of these settings. However there is no way to control this operating system panel to act from the command line or from a script. So a significant amount of time was spent developing a command line tool called *lcfgnetworktool* in object orientated C++ with some Core Foundation and Carbon aspects. The tool allows for almost complete configuration of all networking settings by accessing the system configuration framework directly. It works in a way that is very similar to the operating system's own networking panel.

It is possible that with DHCP many of these settings are unnecessary. However the amount of time taken to add all rather than a subset was insignificant.

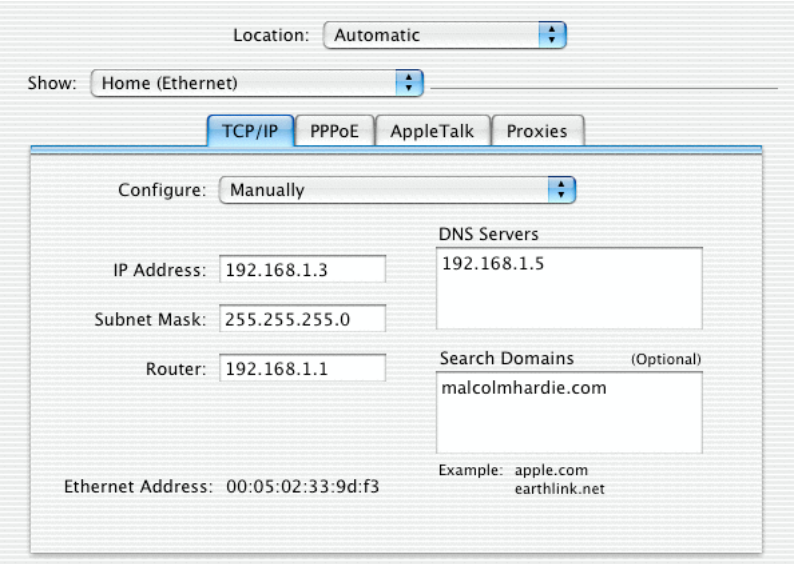

Figure 5.4: The Network System Preferences Panel with the ethernet Tab Visible

Currently the only settings that cannot be configured in this way are those relating to modems and AppleTalk. I made the assumption that these settings were unlikely to be required in a large-scale network deployment as both are now usually considered legacy technologies. PPPoE support was also grouped with modems and excluded although since a framework has now been developed for the other settings it should be feasible to extend this to cover these additional connection methods. The proxy, IP address and DNS setting support would support configuration of the relevant aspects of these additional technologies. The entities used are identical regardless of the service type, it is the selection of entities attached to each service that is important. The format of the entities remains constant regardless of service type.

It is possible that there may be interest in the tool separately for configuring the network interface from the command line. Currently it does not appear that a useful and simple command line tool to configure the networking preferences is available. Had such a tool (e.g. ncutil) been available it might have been preferable to use it instead to reduce the requirement for new code and testing.

More common approaches seem to involve editing the file directly or using AppleScript, which has assorted disadvantages as previously discussed.

More work would probably need to be done to improve the interface to the LCFG network tool before releasing it independently. In particular the development of a graphical user interface perhaps in cocoa is a possibility. Currently the tool does not have a graphical user interface, all of the functionality included can be accessed by using the network control panel.

It might be possible to repackage the code into an Applescript library for users of Applescript to configure networking as discussed earlier. It might also be possible to use lcfgnetworktool and some of the *divine* code to produce a version of divine that worked on Mac OS X platforms. Currently, set switching must be done manually.

Such extensions are, however, outside the scope of this project.

To integrate this tool with LCFG it was then necessary to write a component to obtain values from the profile and to pass these values to the tool.

A complete dictionary of options has been made available to profile authors to allow for configuration using any supported method.

The changes are updated to the system preferences as soon as the component is called and occur immediately. The user interface updates in real time and open preferences windows are advised as to the new state. However it should be noted that the network preferences control panel pane does not reflect the current state and must be reloaded before changes can be seen. This is a limitation of that program's user interface and does not reflect the underlying state of the system. If for some reason it might be desirable to ensure that this doesn't happen a script could be written to close the System Preferences application prior to changing the network configuration.

In addition the whole process is transparent to the user. All applications should normally adjust to the new settings automatically and the

Most internet applications for instance will update immediately and without loss of service. This includes changing proxy information and even the IP address or DNS servers.

Applications that follow the correct guidelines will immediately change to use the new settings without seriously affecting the user. Web browsers that support the system proxy settings will also update these automatically and immediately although not all browsers support this facility yet. Typically the only visible alteration will be to the operating system location sub-menu in the Apple menu that will change to indicate the name of the newly selected location (set) and this will only occur if there is more than one set defined. In tests all supported configuration parameters could be changed without loss of service, although some inconsistencies could occur if a change occurred while a connection was open. After the change it was usually necessary to reconnect. However this behaviour is expected and does not impact the user greatly due to the infrequent requirement to change IP addresses.

5.4 Machine Component

The machine component is designed to configure assorted aspects of the machine that do not fall into any other component, are necessary for the functioning of the machine and do not warrant the creation of a dedicated component of their own.

Initially the idea was to have a single component that configured the network settings and arranged the machine specific preferences. However the networking component proved to be quite independent of the machine settings and so it was separated, or rather, never joined into the machine component.

The machine component is mainly an LCFG component although there are some other small bits of code that have been produced to make minor alterations to the system.

The main thing that the machine component does is to alter the /etc/hostconfig file. This file contains a list of services that should be enabled and disabled. The SystemStarter subsystem then uses this list on startup to decide which services should be started. Additional items can be added to the list to support further user defined services. One such additional service is the LCFG startup service which handles processing at boot time.

The format of the file proved very simple and Perl was a logical choice.

5.5 Display Component

5.5.1 Overview

The display component was originally planned to be part of a larger machine component. The current machine component configures only the service list, but it was originally planned as a larger unit that controlled several different aspects of the non-application system software preferences.

It was concluded however that a number of smaller more focused components was a preferable option rather than a single large but complicated component. Smaller components are easier to understand and easier to develop. There is also an advantage in that if a machine or a site is found not to require some particular component then it can be removed completely. This would not be possible with a single larger component.

The display component is designed to control the monitor or display attached to the computer. It currently allows for the configuration of a few basic settings,

however further improvements in this area would not be difficult.

The display component works by calling a command line tool called *lcfgdisplaytool* and specifying values for the required display settings which it obtains via the LCFG libraries.

5.5.2 lcfgdisplaytool

The *lcfgdisplaytool* is a command line tool that was written in C++ and uses the Carbon development libraries. It calls functions in the the CGDirectDisplay API [14] which is a component of the operating system. [20]. This together with the *Display Manager* allows for dynamic configuration of attached displays. It allows for multiple displays, for displays that mirror, and displays that are logically joined to form a single large visible area. [2] Configuring the display manager causes the changes to be reflected immediately. This would present an undesirable experience for the User if it happened during a work session. To prevent this the component offers a setting that determines whether the changes should be made immediately or deferred until a later time. Should the deferred option be chosen then the settings are saved in an action queue and then performed after the user has logged out. The option was retained to support immediate changes because the deferred action is provided by building on the immediate action.

The original plan for configuring the display assumed that changes could be made to the display preferences file[3] and that these changes would alter the display configuration. However it was found that this did not have the desired effect. Making changes to this file caused the machine to start up with the new settings but as soon as the user logged in the previous settings would be restored and the changes would be lost. It became clear that altering the files would not be the correct approach.

It was eventually realised that only by altering the in-memory settings and telling the machine to save those settings could any permanent change be made at all. This has the disadvantage of those settings being changed immediately for which the deferred option was required.

After some experimentation it was discovered that the correct approach was to use the *The CGDirectDisplay API*. This is the same approach that the operating system uses to change the display settings. The desired settings are passed to a function that determines which of the available display modes best matches the request. This setting is then selected and enabled until the application terminates.

[2] Currently the display component does not support some of these additional features however there is no reason that the component could not be extended from its current basic feature set should such functionality be required.

[3] This file is stored typically in */Library/Preferences/*.

To make the changes permanent a second function is called that saves the settings. This approach is believed to be the same as that used by the operating system user interface to configure the display settings.

5.6 Netinfo Component

Most Unix distributions store preferences as text files in the */etc/* directory of the file system. This has many advantages; text files are easy to read and easy to parse. However there are some disadvantages as well. Distributing settings across machines requires a system such as LCFG (or rsync for less complicated scenarios). The text files can have any arbitrary structure and the format has no commonality between applications. (Although many applications do follow standard conventions which improves the situation greatly)

For NextStep the Next developers created a hierarchical tree structure for storing system preferences. It stored these settings in a central database. The database could be either a local database for each machine or a single database which could be replicated across a network. This allowed a number of interesting improvements to be made and allowed for users to be able to log in at any machine or for settings to be changed remotely.

On Mac OS X the Netinfo database is also used to restrict user access to the machine. It supports access control to a number of features including applications and system preferences. It is the main security directory for this type of user limitation.

Systems such as LDAP follow a similar approach to storing information although most Unix distributions do not use LDAP to store information to the same extent as NextStep did.

With the move from NextStep to OS X, Apple retained the functionality of Netinfo although its future does not look that secure. Newer technologies such as LDAP and more standard approaches such as the */etc/* directory have reduced its usefulness. Also the move to XML based preferences has meant that the Netinfo is now being used less than it was with NextStep.

However despite this move towards more industry standard technologies it is still the underlying configuration system for Mac OS X machines. [18]. Every Mac OS X machine has a Netinfo database that typically stores a number of key settings related to the machine. Often the administrator has the option to either use Netinfo or the */etc/* directory configuration system for a specific set of peferences. However the order in which these are searched is configured through Netinfo so it is difficult to ignore it within the context of configuring a machine.

5.6.1 Design

Combining Netinfo and LCFG appears initially to be an illogical thing to do. LCFG appears to replace all of the functionality of Netinfo with the possible exception of user profile information. However as more research was done a better understanding of the situation developed. [18]

Each Mac OS X machine runs its own local Netinfo domain. A machine can optionally be a member of a larger domain. Information is accessed using a look-up program that runs on the local machine which automatically rises up the hierarchy until the correct information is found.

From a configuration viewpoint the Netinfo system provides a number of useful capabilities. It can contain any number of different preferences using a tree like structure. Programs and operating system utilities can load and store data into the tree. In addition there are compatibility libraries that allow posix utilities that expect to find information in the */etc/* directory to access data from the Netinfo directory system without code changes. This functionality has allowed a number of preferences to be consolidated into a single Netinfo component.

The key question that remains therefore is how to define the balance between Netinfo and LCFG. Should intermediate servers be used which could be referenced in LCFG profiles or is it better to load the information directly into the local domain of each machine?

As the project has progressed it became clear that Netinfo is relatively less efficient than might be desired. Apple documentation [18] recommends that a network with more than 100 computers must have multiple Netinfo servers for performance reasons. This suggests that overall performance in large networks may be unsatisfactory. There is also a suggestion in later Apple documentation [23] that the company intends to move away from the Netinfo system in the future towards LDAP and Open Directory[4].

An early thought was to develop an LDAP component and server as part of the project. However this was both relatively time consuming and completely supported by existing technologies. So instead the focus will be on situations where an LDAP or Open Directory server is not available. In many actual deployments it would be recommended to store user information in a central LDAP server and this is already supported by Mac OS X.

One approach that was considered was to require that each network using LCFG/Mac OS X to run a machine with a Netinfo server. This could then replicate user, profile and general Netinfo data across the network to other machines.

[4] An Apple supported, LDAP based, directory system

An alternative, although less secure, route might be to provide user information using the Netinfo component and store the information attached to the LCFG profile. This has issues of security and may be a security risk on a non-secure network. It would not be recommended for a large scale deployment. However it is entirely feasible with the equipment available and it allows for a system to be deployed without special efforts being made to support the Macintosh platform.

One improvement to security might involve a secured HTTP connection to download the profile which would allow the storage of such data within the LCFG profile without security issues. Alternatively the Netinfo data could be encrypted using a public-private key approach.

5.6.2 Implemention

The final approach used to providing LCFG and Netinfo integration was relatively simple in concept. The LCFG XML profile contains a reference to a block of XML that represents the keys and values inside a Netinfo database. This XML can be either in a separate file (useful for working with Linux lcfg servers) or can be contained inside the profile directly (more organised). Each path to a key within the XML represents the same path to the same key in the Netinfo database. There is a one-to-one mapping between the two for the keys specified in the profile. However some keys may exist in the Netinfo database without being in the profile.

The component was developed in Perl and uses the command line tools to configure the Netinfo database. The component uses XPath for the XML parsing because it seemed to be a good approach for a path orientated data retrieval task, possibly more so in this case than the alternatives of XML-DOM or token based approaches. (Although such approaches are used elsewhere within the project for other tasks.)

The component has some configuration options within the main body of the profile. In particular there is a setting to define which keys should be read from the profile. Early versions of the LCFG component for Netinfo did not support NFS configuration within the Netinfo database because such configuration required settings as shown in fig 5.5. The main problem is that the name of the dictionary is the name of the mount. This name is required to reference the information contained but the colon is a path character for the command line tools and as such cannot be used within a Netinfo path. A later solution was to use a valid but incorrect name for the dictionary during the initial naming for NFS mounts and then change it to the correct setting as a final action. There is still some concern about this method and it should perhaps be separated into a independent NFS component. (Possibly the LCFG-NFS component could be used with

minor changes)

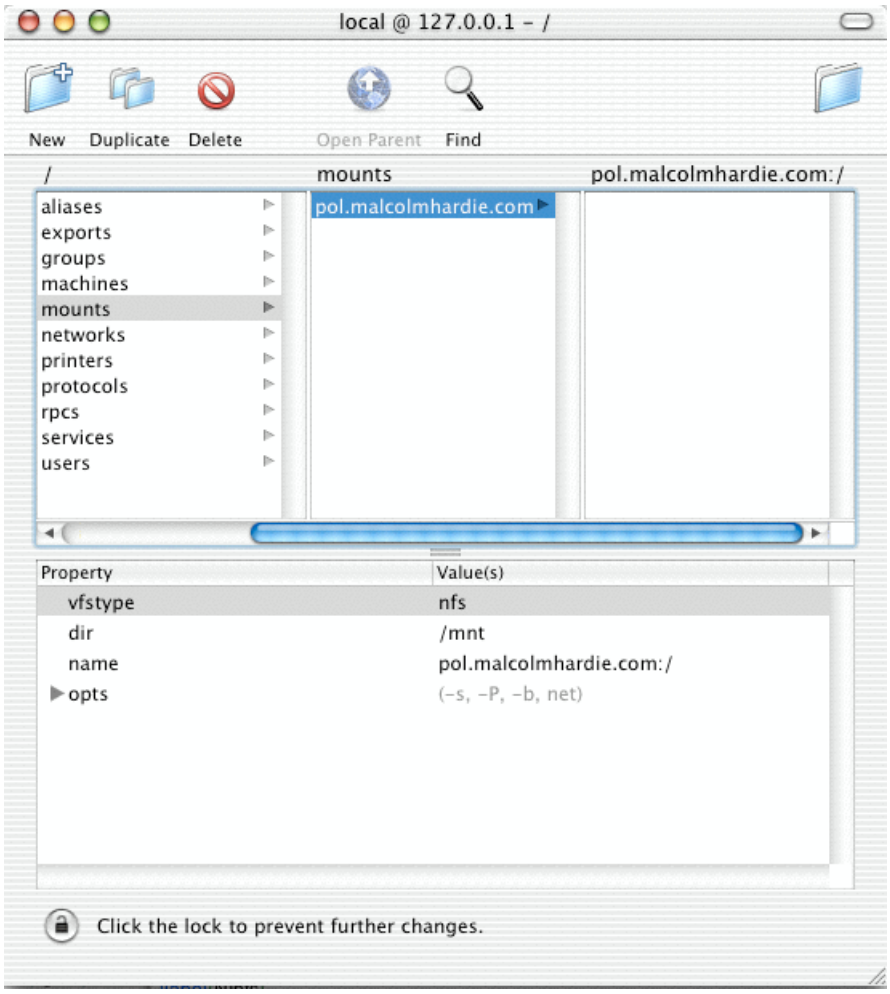

Figure 5.5: Netinfo Manager showing configuration for NFS mount

5.7 Preferences Component

The preferences component allows for arbitrary configuration of application preferences using XML or the defaults system. It is provided as a method of last resort and ideally specific components should be written for application configuration. However because of the way that applications are configured on the MacOS X platform it was possible to develop a generic component that could configure any application that uses the standard configuration system.

It is hoped that later components could be developed by sub classing this component to provide the advantages that a full LCFG component provides such as

correct inheritance and configuration reporting.

Fig 5.6 shows the contents of a typical plist based configuration file. The plist or property list is an XML based configuration format used by Apple and supported natively by cocoa, carbon and core foundation with simple API calls. The API is sufficiently simple that only applications with very specific needs use any other system.

The preferences component was rewritten during the project to move it from using XML files to using the main machine profile. This offered a number of benefits such as inheritance of settings and easier configuration. It also removed the requirement for a patched version of mkxprof which improves integration with a larger LCFG network.

The raw XML for this example is shown in Appendix A

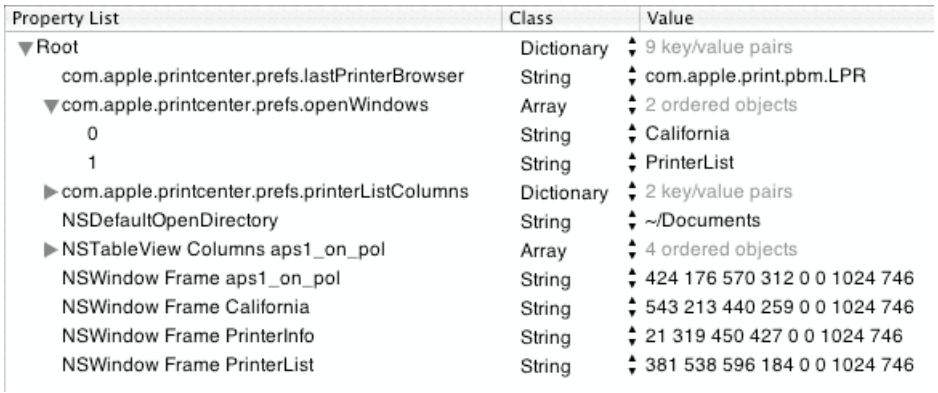

Figure 5.6: A example of the preferences stored in a plist

5.8 Package Management Components

Package Management is an important aspect for a configuration system. Files stored on the disk of a workstation or other machine often must be updated or kept synchronised with a central set of files. These files might be data files or applications or any other type of file.

The Linux version of LCFG uses RPM packages to update files which works very well. On Redhat Linux RPM archives are the normal way of distributing updated packages.

However for the Mac OS X platform this presented some difficulty. RPM utilities can be compiled on the platform but most people do not use the format for distributing archives. Had the software been directly moved across this would

have required administrators to recompile each Macintosh package into the RPM format which would add complexity and increase the possibility of errors.

So the requirement arose to use native packages. However there remained the question of which one?

There are several formats that are used to distribute software for Mac OS X machines and they have different advantages and disadvantages.

5.8.1 A Plug-In Approach

Given the difficulties of selecting a definite choice of formats to support it appeared that the most efficient approach would be to develop a plug-in architecture. Different package formats could be implemented as necessary and a management component could instruct plugins to manage different types of packages. Two formats were initially selected to provide base functionality although other varieties could be implemented with only a limited amount of extra time.

5.8.2 PKG - The Default Choice

The format that is most similar to RPM is the PKG format. It is used to install the operating system and most system updates are delivered in this format. It is well supported by graphical and command line tools. There are some problems with PKG files. In particular it has been noted that carelessness when building PKG files can cause disaster. [6]. The Mac OS X root directory is structured so that /var and /tmp and /var are located in /private and symbolically linked from the root directory. This means that files for /etc must be installed to /private/etc. However some package creators fail to do this and the PKG handling software is not sufficiently sophisticated to correct this. The result is that the symbolic link is overwritten with a directory and the machine configuration is severely damaged.

The PKG utilities track installed packages in a *receipts* directory on the disk. The package is copied to the directory but the compressed file data is removed leaving only the bill of materials (BOM) file and the versioning information. By parsing the receipts files the source of any installed file on the disk can be traced. (Although files that were not installed from packages cannot)

5.8.3 APP - The Application Bundle

Applications are usually delivered in a directory structure known as a bundle. The bundle may contain multiple executable binaries, resource files, help files and support files. An application bundle also contains versioning information. The current recommendation in Apple developer documentation is that an application should be complete and should store all necessary files within its own bundle. Applications that follow this recommendation permit drag and drop installs from a disk image or CD-ROM.

It would be possible to develop a system that used internal versioning information in installed applications to manage them without an external database.

There is still a difficulty in managing applications that have been installed from packages. So such items must be added in to the installed application list before any processing occurs. Failure to do this in early versions of the software caused applications that were installed from packages to be deleted.

5.8.4 The lcfgbundletool

Working with packages required constant access to dictionary based Info.plist files. These files store versioning information for .pkg packages and .app bundles. To make this easy to access from a command line environment a new tool was developed in Core Foundation and C++ called *lcfgbundletool*. This tool, when provided with a bundle and a key, locates the correct plist file inside the bundle according to Apple rules and then reads the value associated with the specified key. This is then printed to the command line. This tool is intended to be used from plug-ins that access bundle based packages and greatly simplifies the management of bundle information. The motivation for developing it in C++ was that the equivalent plist dictionary routines immediately available to Perl were immature[11] and the format of the bundle is variable depending on the method used to create it. It was concluded that given the assorted possibilities for file naming and location within a bundle, the ease with which the C++ tool could be written and the lack of an immediately available library to parse the files it was better to write in C++ using the carbon API than to develop a Perl tool. (However it should be noted that the plug-ins that call the tool are written in Perl as is the component that manages the plug-ins). As the development of the Perl bindings improve perhaps this could be reconsidered.

5.8.5 Extending the Profile.

In order to provide enough data to the packaging component it was necessary to extend the format of the profile to support additional data. In particular the type of the package being installed had to be included and in early versions the destination path for Application bundles was included too.

In order to do this the possibility of changing the XML storage format was considered however this might affect server components and while the package format has been altered already to support netinfo it was considered better to retain as much of the format as possible.

The approach finally chosen was to embed further information within the package name and to parse the information upon processing. This is not an ideal solution and it may need to be reconsidered. In a Macintosh-only environment it would be better to rewrite the components of LCFG that handle packages to support the necessary additional information. However it has the significant advantage that it can operate under Linux LCFG servers without needing changes to the main tools.

6. Future Extensions

6.1 Future Directions

Current thoughts on extensions to the project include the following:

1. A mechanism to allow machines to discover the location of an LCFG profile server.

2. A component that manages Classic Environment applications.

6.1.1 Automated Profile Discovery with Rendezvous

Using Rendezvous[1] [17] a machine on a network can discover services and use them without a centralised directory system. Rendezvous support is built into Mac OS X and any machine can act as client or server. By advertising a machine as an LCFG profile server an administrator will be able to configure client machines by selecting from a list rather than having to remember a numerical location or machine name. It might also be possible that a machine could declare itself a server if it could not find one already on the network. As new machines joined the network it could automatically create minimal profiles for them. This would allow a disconnected network to be managed in a limited way using this software and could help administrators to deploy the software initially. The server would be able to provide a framework and the only manual changes required would be the actual settings required.

Zero-Conf support is being developed for Linux and it is probably feasible to port this technology back to the Linux LCFG framework. It is unclear if such a facility might be useful on Linux.

Implementing this component would allow smaller groups and organisations to use LCFG Mac OS X easily while allowing a later transition to a larger and more centralised architecture without changing the software in use. A machine on the network could be designated the server without noticeably impacting on performance, because the number of clients would be very small.

[1] Rendezvous is also known as ZeroConf by the IETF

6.1.2 Classic Environment Configuration

Some work has been done in investigating how the Classic Environment operates. The conclusions reached here are as follows:

1. The Classic Environment is almost but not quite a normal application
2. It stores preferences in a plist (XML) file in a fixed location
3. The Classic Environment makes changes to a standard Mac OS 9.x installation
4. There are some security concerns with running Classic in a shared Environment although it is unclear how serious these might be. There is also a trade-off on the part of the administrators of such an installation between security and functionality.

The Classic component has the potential to be developed to varying levels. It is conceivably possible that a future version might be extended to configure Mac OS 9.x systems as well although how the LCFG libraries might operate in such a system is unknown.

- At the most basic level the component treats the Environment as a normal application, configurable with arbitrary parameters. Packages that might need to be installed could be installed using the standard package format, although some sophisticated post-installation scripts might be required to prevent damage to resource forks for installed files.
- A more sophisticated level might offer greater control over the Environment and be able to configure some of the settings inside the Environment.
- The most sophisticated level would provide control over the entire Environment and all internal settings. This would be almost equivalent to writing a port of LCFG to Mac OS 9.x with the further difficulty that Mac OS 9.x is not a Unix-like operating system and hence does not have built in Perl support. However it would be possible and the Mac OS 9.x port of Perl does extremely well at bridging the gap.

7. Conclusion

7.1 Evaluation

At the beginning of the project a number of objectives were defined. On the whole and subject to some limitations the concrete objectives were met.

- *Research the feasibility of porting LCFG client technology to the Macintosh platform and investigate how this might be done*

  A significant amount of investigation was conducted into LCFG, Mac OS X and the possible interactions between the two. This formed the basis of the implementation that was carried out it in later steps. Without this research it is unlikely that the project implementation could have progressed efficiently. The conclusion was that the technology could be developed on the Mac OS X platform but that some aspects would have to be re-implemented, replaced or removed.

- *Create a framework that can configure a Macintosh client machine and supports extension via components; use LCFG client technology if the first objective is feasible*

  The framework was completed and is able to configure a client machine. There are some minor problems with some of the more advanced features when interacting with a server. It uses LCFG client technology and it is extensible using components. There are some additional Macintosh specific framework features that factor out common Macintosh specific functionality such as scheduling activities to occur when a User logs out.

  The framework has been reasonably integrated in the operating system. It can run without notice by users.

- *Develop a number of working components that allow basic configuration of the most important settings for a client Macintosh machine and represent examples for future expansion*

  Component development took up most of the development time during the project. A number of new components were developed. In particular the network component, the display component and the machine component were all entirely new. They depended heavily on Macintosh-specific features and software. Various tools were written to support the operation of these components which could be possibly be reused for future components.

  From this overall component objective a list of secondary objectives related

to components was developed. These are detailed on page 25. All of these objectives were met to a less or greater degree but in same cases with limitations.

The users and groups facility is insecure and has some problems. Using a download from a web site to configure the user list on a machine is probably not ideal. In a production environment this should be moved to LDAP. A planned extension to the Netinfo component supports encryption (the support files and schema have entries for these but no code has yet been written).

Whether to allow configuration of the root user is still an issue. The current version does allow this under the assumption that the transfer of data is secure. This allows better configuration at the cost of poorer security. It is optionally possible to prevent this by making a change to the component on the client machine.

Active service list changes currently require a restart because Apple does not support stopping of services in all cases. Some services can be stopped but for others the support is as yet unavailable. To get around this problem a restart is triggered at the next log-out if the service list changes. Apple documentation [21] indicates that this issue is a bug and as such should be resolved in future revisions to systemstarter and the boot system, however it is somewhat inconvenient at present.

- *Integrate this system with both LCFG server technology and the Mac OS X operating system software to the best extent possible*

  The objective of achieving a high level of integration meant that components often had to be redesigned after initial versions were produced because they did not match with one or other of the two technologies.

  The Netinfo component became more complicated than had been originally expected. The change from early versions where the Netinfo information was stored within the profile to the later versions where it could optionally be stored in a separate file (in XML or native formats) was significant in improving interaction with base LCFG installations. The original version required major changes to the mkxprof tool and making such changes to a tool as important as mkxprof might have proved problematic in a large system. However the Netinfo component is still the least LCFG like of all of the components included. Some work was done trying to move it to LCFG style lists instead of the current XML approach but this was quickly abandoned as being excessively complicated. Should it prove necessary then it might be possible but would require a more complicated component and possibly a new set of command line tools.

The preferences component was also redesigned to make it more compatible with LCFG. Unlike the Netinfo component, with the preferences component it was possible to move from the XML of early versions into an LCFG list structure stored directly in the profile. This was mainly due to the simpler and more structured nature of the preferences data compared to the Netinfo data.

The development of the network component underwent significant changes also but in the reverse direction. It became better integrated with the Mac OS X operating system. Early versions relied on using static files (see page 29) but later versions moved towards the correct Apple approved method of accessing the files via system libraries. This was a major improvement in compatibility, both with current and future versions of the operating system and also offered the advantage of being able to update settings immediately. One feature that worked well is the capability to change all of the settings without interrupting the User at all.

The display component is actually the most balanced component because it is relatively simple in structure. Early designs for it suggested it should be part of the machine component but the rich functionality of the display manager and core graphics suggested the development of a separate component. Although the current component offers only a subset of the total capabilities of the system this could easily be extended to support further options although this was limited here due to the non-core nature of some of these facilities.

Overall the level of integration is quite good. The average Macintosh User should not even notice the presence of the system except perhaps for the occasional restart as they log out to initiate changes in settings.

- *Document the system carefully to allow future expansion and further development of a Macintosh client as an ongoing effort*

  A number of Man pages were produced to document the code. It is hoped that these will be useful.

7.2 Improvements

There is still much scope for improvement.

The main areas in which the level of integration could be improved are the field of user interface integration. It would be useful to have a preferences pane that could display information about the machine and its profile, perhaps with a graphical

representation of services and an indication of where the profiles are being served from.

Supporting contexts as locations in the user interface would be another improvement. This would allow users to select the contexts from the standard system location menu Fig 5.2 and would make this feature much easier for Macintosh users to understand.

Package management could be switched to use the OSX-GNU package tools. Initial support for these is already in place but at the time of writing they are still somewhat immature and the benefits promised are not entirely delivered by the software. In the longer term this switch could yield important new features such as more reliable package formats and better package removal support. Support for RPMs could also be added. This would be relatively simple with the plug-in architecture developed for packages and might provide for easier package management in environments that already use RPM (such as default LCFG installations).

With respect to components it would be useful to develop components to manage the mime-type information on the system. It is relatively difficult to configure this information even on a single machine and so this is often left to the default setting. It is also ideally suited to a value-key structure. Automounting of NFS directories would also be useful, particularly in a lab setting. The auto-mount daemon on Mac OS X has some noticeable problems and it would interesting to solve these difficulties. A final additional component would a more sophisticated user and group component. The current approach uses the Netinfo component and relies on the system to create home directories. A more sophisticated approach would use LDAP and configure this information automatically from a server. It would also handle the creation of home directories more efficiently.

7.3 Level of Functionality Achieved

The system is fairly close to being deployable. Much of the basic functionality required for deployment is included and the additional functionality required is limited. The level of visible intrusion into the system has been limited (A concern from the design phase - page 14).

A network of machines could be deployed and configured relatively well using existing LCFG Linux infrastructure. The level of configuration does not yet meet the level of the Linux client however if in regular use this could be improved as additional features were required. Some care would have to be taken to test Apple software updates against a test machine to ensure that these do not delete or unexpectedly update software that is required by the LCFG client.

Whether this system could be used in a student lab is unclear. A student lab would have higher reliability and security requirements. While several groups have deployed labs with Mac OS X [28, 27]and LCFG has been extensively tested and deployed at Edinburgh [30] and elsewhere, the combination has not been tested beyond this project. It would be recommended to run a small lab under test conditions before deploying on a larger scale but it is likely to work successfully.

The main and obvious feature missing is printing support. Mac OS X natively uses CUPS printing [7] which should be able to discover printers automatically if print servers were configured to support it, one simple approach would to configure a single Macintosh in each subnet to act as a CUPS server and the remainder of the machines should automatically discover it and allow printing to all printers. [39] CUPS support could be developed as a cross platform component and this already a future direction of the LCFG Printing Group. [23]

Alternatively printing could be arranged via Samba [9] with minor alterations to the LCFG-Samba component

One point raised during the review process was how hardware driver installation could be performed. While this is not specifically included as a component, drivers for hardware are typically simple collections of files. These files are automatically discovered by the operating system when the device is discovered. This suggests that the package management component could be used for this functionality.

Ideally LDAP would be used in preference to the Netinfo component. It offers better scalability and a more proven technology. However the Netinfo component does work reasonably well and could be deployed in the absence of a suitable LDAP infrastructure.

7.4 Review

The project as a whole was reasonably successful. Some approaches and design decisions were in retrospect incorrect and some work had to be repeated using different approaches. This was a result of using a prototype driven development strategy although it did cause some delay.

Integration was perhaps one of the main challenges of the project, to get the disparate sections working together. This seems to have been achieved although perhaps the approach of tools and components being separated could be re-thought in later versions. It is unlikely that much more could be written in Perl without the writing of more Perl bindings. For this project it was easier to write tools directly in C++ and then access them from Perl.

Viewed as a complete system the LCFG for Mac OS X project provides a working and usable system that can configure Macintosh clients and should integrate fairly well into a Linux LCFG configuration fabric. Macintosh users should not notice any intrusion and the system stability and usability should not be affected.

It is believed that there is no currently available system that offers equivalent functionality to LCFG on the Mac OS X platform and it is hoped that the availability of such a system will prove useful.

In the future it is hoped that this client can be extended in future to support additional features and offer support for the extensions described.

Appendix A. Source code for plist example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>NSDefaultOpenDirectory</key>
<string>~/Documents</string>
<key>NSTableView Columns aps1_on_pol</key>
<array>
<data>
BAt0eXBlZHN0cmVhbYED6IQBQISEhAhOU1N0cmluZwGEhAhOU09iamVjdACF
hAErDHN0YXR1c1N0cmluZ4Y=
</data>
<string>97</string>
<data>
BAt0eXBlZHN0cmVhbYED6IQBQISEhAhOU1N0cmluZwGEhAhOU09iamVjdACF
hAErBG5hbWWG
</data>
<string>450</string>
</array>
<key>NSWindow Frame California</key>
<string>543 213 440 259 0 0 1024 746 </string>
<key>NSWindow Frame PrinterInfo</key>
<string>21 319 450 427 0 0 1024 746 </string>
<key>NSWindow Frame PrinterList</key>
<string>381 538 596 184 0 0 1024 746 </string>
<key>NSWindow Frame aps1_on_pol</key>
<string>424 176 570 312 0 0 1024 746 </string>
<key>com.apple.printcenter.prefs.lastPrinterBrowser</key>
<string>com.apple.print.pbm.LPR</string>
<key>com.apple.printcenter.prefs.openWindows</key>
<array>
<string>California</string>
<string>PrinterList</string>
</array>
<key>com.apple.printcenter.prefs.printerListColumns</key>
```

```
<dict>
<key>SortOrder</key>
<string>name</string>
<key>TableColumns</key>
<array>
<string>name</string>
<string>374</string>
<string>status</string>
<string>199</string>
</array>
</dict>
</dict>
</plist>
```

Appendix B. Glossary of Terms

There is a considerable amount of terminology that is specific to the Macintosh platform. Other terms may also need explanation.

- Airport - Apple name for 802.11b Networking standard
- ADSL - Asymmetric Digital Subscriber Line - Broadband Access system over copper wire
- BOM File - Bill of Materials File, a file that lists the contents of a PKG format file package.
- Carbon - A library developed to support C++ development on Mac OS X and to ease transition from earlier operating systems
- Classic Environment - Mac OS 9.1 running inside a virtual machine on a machine running Mac OS X
- Cocoa - An object orientated application development framework
- CUPS - Common Unix Printing System - http://www.cups.org [7]
- Darwin - FreeBSD derivative that is the underlying operating system of Mac OS X
- DHCP - Dynamic Host Configuration Protocol, a method of configuring computers using network querying
- Display Mirroring - To Display the same information on more than one monitor at the same time.
- DNS - Domain Name Server, A server that converts from domain names to IP addresses
- IETF - The Internet Engineering Task Force
- Mac OS 9.x - Older Macintosh operating system that can be used on machines built before 2003. It includes versions 9.0 to 9.2.2
- Mac OS X - The complete graphical operating system as used on Apple Macintosh Computers (1998 and newer)
- Mac OS X Server - A version of Mac OS X that is optimised for server usage and includes additional server features.
- NetInfo - A centralised database of system settings for a Mac OS X, Darwin or Next machine

- NextStep - The operating system that Mac OS X was developed from.
- NFS - Network File System - A method of accessing disks attached to servers over a network, developed by Sun
- Quartz - The main graphics rendering system on Mac OS X
- PPP - Point to Point Protocol - Used to provide TCP/IP over a serial Link
- PPPoE - PPP over Ethernet. A standard used in ADSL Networks.

Bibliography

[1] Paul Anderson. Getting started with lcfg. http://www.lcfg.org/doc/pdf/lcfg-tutorial.pdf, 2001.

[2] Paul Anderson. LCFG adaptors for XML profiles. Internal Document, 2001.

[3] Paul Anderson. Tools for building and packaging modules from the dice cvs repository. Internal Document, 2001.

[4] Paul Anderson, George Beckett, Kostas Kavoussanakis, and Peter Toft. Technologies for large-scale configuration management. http://http://www.epcc.ed.ac.uk/gridweaver/WP1/report1.pdf, Dec 2002.

[5] Paul Anderson and Alastair Scobie. LCFG: The Next Generation. In *UKUUG Winter Conference*. UKUUG, 2002.

[6] Scott Anguish. Beware of installers bearing packages: Part ii. http://www.stepwise.com/Articles/Technical/Packages/InstallerOnX.html, 1999.

[7] John Berry, Toby Blake, and Neil Brown. Lcfg printing group progress report. http://www.dice.informatics.ed.ac.uk/groups/services/printing/progress_report.html, 2003.

[8] Kevin A. Boyd. http://www.macosxlabs.org/slides/RsyncX17d_preso_short.pdf, 2002.

[9] Cups.org Easy Software Products. Common unix printing system website. http://www.cups.org, 2003.

[10] Michael Egan. A simple analysis of mac os x net-booting. http://http://mike.passwall.com/macnc/, 1999,2000,2001,2002,2003.

[11] Brian D Foy. Perl modules by brian d foy. http://brian-d-foy.sourceforge.net/, 2001.

[12] Ofir Gal. Radmind manual for mac os x. http://www.gal.co.uk/software/radmind/radmind-manual.pdf, 2003.

[13] Simson Garfinkel and Michael Mahoney. Steve jobs and the history of cocoa. http://www.macdevcenter.com/pub/a/mac/2002/05/03/cocoa_history_one.html%, 2002.

[14] Apple Computer Inc. The cgdirectdisplay api technote. http://developer.apple.com/technotes/tn/tn2007.html, 2000.

[15] Apple Computer Inc. http://developer.apple.com/techpubs/macosx/Essentials/AquaHIGuidelines/%index.html, 2001-2003.

[16] Apple Computer Inc. http://www.apple.com/server/macosx/pdfs/L21952A_NetInstall_TB.pdf, 2002.

[17] Apple Computer Inc. Rendezvous technical briefing, 2002.

[18] Apple Computer Inc. Understanding netinfo. http://www.apple.com/server/macosx/pdfs/UnderstandingUsingNetInfo.pdf, 2002.

[19] Apple Computer Inc. Netboot and network install. http://www.apple.com/server/macosx/netboot.html, 2002 2003.

[20] Apple Computer Inc. Display manager reference. http://developer.apple.com/techpubs/macosx/Carbon/graphics/DisplayManag%er/Display_Manager/displaymgr/index.html, 2003.

[21] Apple Computer Inc. Mac os x: System overview. http://developer.apple.com/techpubs/macosx/Essentials/SystemOverview/in%dex.html, 2003.

[22] Apple Computer Inc. Morescf sample. http://www.apple.com/developer/, 2003.

[23] Apple Computer Inc. Open directory. www.apple.com/server/macosx/pdfs/L21953A_OpenDirect_TB.pdf, 2003.

[24] Apple Computer Inc. Xserve macintosh server hardware, the xserve cluster node. http://www.apple.com/xserve/, 2003.

[25] Terrasoft Inc. Terrasoft showcase - university of colorado, bolder. http://www.terrasoftsolutions.com/realworld/showcase/ucboulder/, 2003.

[26] Brian Jepson and Ernest E. Rothman. *Mac OS X for Unix Geeks*. O'Reilly & Associates, Sept 2002.

[27] Mac OS X labs. Participant list. http://www.macosxlabs.org/participants/participants_new.html.

[28] MacOS Labs.org. Login and logout hooks for mac os x. http://www.macosxlabs.org/slides/uef_tech-hooks_pres.pdf, 2002.

[29] Patrick M McNeal and The University of Michigan's Research Systems Unix Group. http://rsug.itd.umich.edu/software/radmind/, 2002 2003.

[30] University of Edinburgh. Dice project website. http://www.dice.informatics.ed.ac.uk/.

[31] Michael A. Olson, Keith Bostic, and Margo Seltzer. Berkeley db. In *Summer Usenix Technical Conference*. Sleepycat Software, Inc., June 1999.

[32] Graham Orndorff. Email servers and mac os x - article 1: Section 2. `http://www.stepwise.com/Articles/Workbench/eart.1.2.html`, 2001.

[33] Richard Rashid, Daniel Julin, Douglas Orr, Richard Sanzi, Robert Baron, Alessandro Forin, David Golub, and Michael Jones. Mach: A system software kernel. In *Proceedings of the 34th Computer Society International Conference COMPCON 89*, Feb 1989.

[34] Alistair Riddell. netbooting howto. `http://frank.gwc.org.uk/~ali/nb/`, 2001.

[35] John Rizzo and Matt Lake. `http://www.cnet.com/software/0-429669-8-6112332-7.html?tag=st.sw.429669%-8-6112332-1.txt.429669-8-6112332-7`, June 2001.

[36] John Siracusa. `http://arstechnica.com/paedia/f/finder/finder-1.html`, 2003.

[37] Charles Srstka. Bootcd software at charlessoft.com. `http://www.charlessoft.com/`, 2002 2003.

[38] Apple Computer Developer Support Staff. Installing perl 5.8 on jaguar. *Apple Developer Web Site*, 2001.

[39] Hi-Resolution Systems. Macadministrator web page. `http://http://www.hi-resolution.com/products_admin3_features.html`.

[40] SCL University of Utah. `http://www.macosxlabs.org/slides/uef_tech-radmind_details.pdf`, 2002.

[41] Felix von Leitner. divine - automatic ip configuration detection for laptops. `http://www.fefe.de/divine/`, 2003.

[42] www.macmaps.com. How to create a bootable cd in mac os x? `http://www.macmaps.com/bootcd.html`, 2002.