University of Edinburgh
School of Informatics

A GUI Interface for LCFG

4th Year Project Report
Computer Science

Craig Devlin

June 6, 2003

Abstract: It has long been realised within the UNIX community that there are no efficient tools for managing and maintaining the configuration of large numbers of machines. LCFG has been designed to address this problem and is currently used in the School of Informatics at Edinburgh University. This paper describes the process from conception to completion of creating a GUI to enhance user productivity and comprehension when writing LCFG source files.

Acknowledgements

Carwyn Edwards for his introductory lecture on LCFG.
Paul Anderson for his initial briefings on the problem definition and the architecture of LCFG.
Lex Holt for his advice and assistance throughout the year.
Martin Murgh for his ideas on how to define the problem.
Kings Buildings Support Staff for their appraisal of the system and for helping me to come up with a list of extended features.
Volunteers for appraising the 'look and feel' of the user interface.

Contents

1. Introduction

1.1 Introduction

Managing large networks of computers is a difficult and continuous process: networks constantly change and adapt to meet new requirements. This process not only encompasses the addition and removal of nodes but also the configuration - both software and hardware, of each individual machine. This task is difficult as it stands but as the number of machines increases how can a correct and up to date network be maintained without increasing the human resources available? Obviously there is need for some form of network management system that allows direct and precise control of nodes whilst at the same time automating the configuration process usually done manually.

Local **C**on**F**i**G**uration system (LCFG) [1] is a system designed for automatically installing and managing a large number of UNIX machines. Development started on this project in 1993 by Paul Anderson at the University of Edinburgh. This initial version ran under the Solaris operating system. LCFG was then ported over to the Linux[2] Currently LCFG is used in the **D**istributed **I**nformatics **C**omputing **E**nvironment (DICE) at the University of Edinburgh and also a variant of the LCFG system is used as part of the European DataGRID project. The above projects I will go on to explain in further detail shortly.

Currently machines are configured by receiving a set of configuration details from a central server, the configuration details are edited from this central point. Each machine has its own set of configuration details known as a source file. These source files detail the hardware, software and miscellaneous information about the machine, making LCFG particularly suited to networks with very diverse and continuously changing environments. Currently each source file is manually created through a text editor package (emacs seems to be the text editor of choice) then saved as a plain text file.

Since creating these source files can be difficult and confusing, the aim of this project was to develop a Graphical User Interface that will allow quick and easy creation/modification of source files even for users with little experience at developing these files. By structuring the way in which the user selects configuration

[1] http://www.lcfg.org

[2] A freely-distributable open source operating system that runs on a number of hardware platforms. The Linux kernel was developed mainly by Linus Torvalds. Because it's free, and because it runs on many platforms, including PCs and Macintoshes, Linux has become an extremely popular alternative to proprietary operating systems. operating system in 1998 and has since undergone several developments and changes [8].

details and by presenting the user with a list of viable options for each configuration details that the above aim can be realised.

1.2 Context and Background

1.2.1 Current Deployment

1.2.1.1 DICE

LCFG is currently used as part of DICE at the University of Edinburgh[3]. The DICE project is a recent development in the School of Informatics, taking the best parts of the previous system and merging them with state of the art infrastructure components. The DICE architecture grew out of discussions amongst the original DICE project group (Paul Anderson, Jeremy Olsen, Alastair Scobie, Simon Wilkinson, with Tim Colles, Lex Holt and George Ross joining the group later). The first stages of DICE integration started 2002, and with the full system to be put into operation in 2004.

The driving factors behind DICE development are (from [1]):

- Maintainability and low Total Cost of Ownership
- Mobile and disconnected operation
- Distributed and devolved management
- Improved internal security

Maintainability and low Total Cost of Ownership are especially important to the DICE system. The DICE system covers machines within the School of Informatics these range from desktops, servers, laptops student lab machines and a great deal more. With such a diverse range of machines a state of the art configuration management system is needed to keep machine configurations up to date. Also the configuration management system will need to make the process of initially configuring and updating these configurations as quick and efficient as possible.Reducing duplication of user and system data so that there is only one source for each type of information will drastically simplify management tasks. Mobile and wireless devices are becoming ever popular and as a result the DICE system induces giving both wired and wireless mobile devices access to the network.

Distributed and Devolved Management will allow more than just systems staff to manage machines without reducing the security of the rest of the machines. Multiple levels of trust are needed in such a system to allow the configuration

[3]http://www.dice.informatics.ed.ac.uk/

system to allow different users to be able to change parts of the configuration information whilst at the same time maintaining stability of both the machine and the network. To achieve the above will require a powerful internal security system and as such is a priority in the development of the DICE system.

The DICE architecture is based around 4 directory services with each service fulfilling a separate role. These are (from [1]):

- **LDAP**[4] contains anonymously accessible, locally available information about the DICE system. This includes the information formerly held in NIS, some host based information such as ssh keys, and other information which is required to be available to a majority of machines on the DICE system.
- **KDC** contains Kerberos[5] specific information. This includes usernames, their passwords (in a hashed form), and Kerberos management data. This directory must be maintained in a secure fashion, with only authorised access.
- **LCFG** contains machine configuration information. This is machine specific information (a machine only has access to information about itself). Currently read access to LCFG information is locally available, write access is restricted to authorised users.
- **DNS** contains globally, anonymously accessible, machine information. This may include items such as server location information.

1.2.1.2 European DataGRID

The idea of a data grid dates back to the first half of 1990. The vision behind them is often explained using the electric power grid metaphor - hence the name grid. The electric power grid delivers electric power in a pervasive and standardised way. You can use any device that requires standard voltage and has a standard plug if you are able to connect it to the electric power grid through a standard socket.

As explained by Ian Foster and Carl Kesselman in Chapter 2 of their (famous) book "The Grid":

> "The current status of computation is analogous in some respects to that of electricity around 1910. At that time, electric power generation was possible, and new devices were being devised that depended

[4]Lightweight Directory Access Protocol, a set of protocols for accessing information directories. access.

[5]Kerberos is designed to enable two parties to exchange private information across an otherwise open network. It works by assigning a unique key, called a ticket, to each user that logs on to the network. The ticket is then embedded in messages to identify the sender of the message.

> on electric power, but the need for each user to build and operate a new generator hindered use. The truly revolutionary development was not, in fact, electricity, but the electric power grid and the associated transmission and distribution technology"

The DataGrid project[6] is funded by the European Union and is aimed at enabling access to geographically distributed computing power and storage facilities spanning several different European institutions. The resources made available by this project is necessary to process the vast amount of data generated by the scientific disciplines of the following three areas:

- High Energy Physics (HEP), led by CERN (Switzerland),
- Biology and Medical Image processing, led by CNRS (France),
- Earth Observations (EO) led by the European Space Agency

No single institution could afford to buy and maintain all the computing power and storage on its own and demands on resources are always increasing. Distributed computing practices can solve this problem by utilising the combined resources of all the institutions thus providing the required performance and scale. The distributed computing environment also encourages the sharing of data throughout scientific communities throughout the world.

Distributed networks like this are no simple task to implement. Making all of the resources seem uniform and transparent to the operator pose many challenges. The geographical location of the resources is a consideration as is the varied hardware and software that each institution possesses. Each institution will have its' own security policies and somehow the Distributed system must be able to coordinate the local security procedures into one global transparent user protocol. Finally once the operator has access to these resources we need to utilise them in an efficient manner by coordinating each institutions resources and hence providing effective and dependable data.

1.2.2 Architecture of LCFG

To complete this project a thorough understanding of LCFG's architecture is required. The architecture itself is modular and extensible and so both the client and server sides are easily upgraded. This is an important aspect of LCFG as updates to the system will happen frequently. On the server side LCFG is used to specify machine configurations through the use of source files - more on this later. Additionally there are the compilers to do both error checking and conversion of the source files into a machine profile that can be understood by

[6] http://eu-datagrid.web.cern.ch/eu-datagrid/

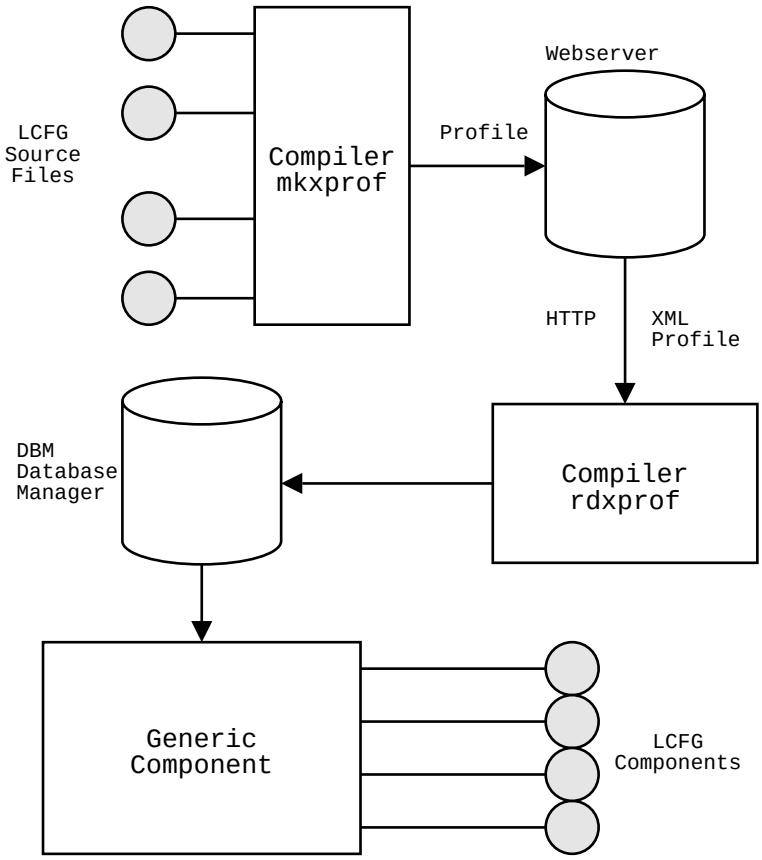

Figure 1.1: Overview Of LCFG Architecture

the client machine. On the client side LCFG initiates retrieval of profiles from the server, is responsible for updating software, booting and installing hardware and software. The client nodes contain LCFG 'components', these 'components' use the profile received from the server to create a local configuration. In other words (by the man who initially started the LCFG project):

> "LCFG provides a configuration language and a central repository of configuration specifications, from which individual UNIX machines can be automatically installed and configured. Changes to the central specification automatically trigger corresponding changes in the actual configuration of individual nodes. The system is particularly suitable for sites where the configurations are very diverse (ranging from large servers to laptops), and different aspects of the configurations may be changed frequently, and managed by many different people. LCFG scales to at least medium-size 1000 nodes."

[6]

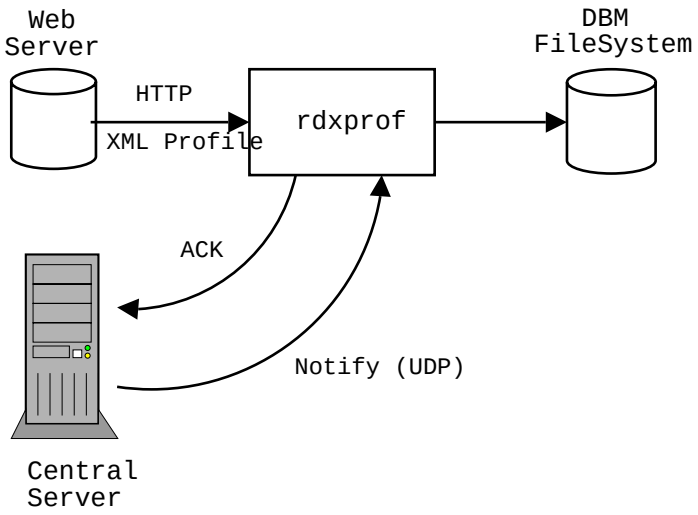

Figure 1.2: Detail of XML file transfer over UDP

LCFG works by keeping a central store of source files, these source files do not necessarily represent one machine they may instead describe an aspect of the overall configuration. These aspects may include things like parameters specific to student machines or parameters specific to machines within a single geographical location. There are also source file aspects for each of the types of hardware and software available. What this means is that at the lowest level a machine source file will call several other source files depending on the machines setup, these source files may then to go on to call yet more source files and so on. The benefits of this hierarchical approach are that because there is only one copy of each source file all machines with common aspect files are updated along with that single aspect file. Hence the machines will all use the same up to date configuration information and also the time spent updating machine configuration is dramatically deceased when compared to manually installing.

Once the source files are completed they are then compiled using the mkxprof (make XML[7] profile) compiler. This results in the creation of a profile - one per machine, this machine profile provides all the information the node needs to configure. The profile contains all the information that makes a machine unique. The profile is in XML format, this is to make transport of the profile to the client and subsequent interpretation easy. The XML profile is transported from the webserver over HTTP to the client, it also possible to implement security features via the use of the TLS (Transport Layer Security) layer of the IP (Internet Protocol) protocol.

[7]Extensible Markup Language used for enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

When a profile is updated the affected machine is sent a UDP (User Datagram Protocol)[8] notification. The client machine then retrieves the profile from the web server over HTTP and stores the information in the DBM (database file). The clients also periodically poll the server to check for new configurations, this is in case the UPD message is lost. The client will also send a UPD acknowledgement packet indicating that it has updated its state, this is so that the server can maintain up to date status information on all the client nodes. Lastly the component modules inside the client interpret and act appropriately on the received profile. Each component module has a corresponding section in the profile on which to act.

Not all machines have the same components installed. The components depends on the hardware, software and purpose of the machine. Currently there are around fifty components. Some examples of components are:

- Mailng - sendmail
- Xntp - time service
- Vmware - vmware
- DNS - DNS configuration
- Lpd - printer
- Apache - web service
- Postgres - database

1.2.2.1 Source Files

This project is based solely on the server side, the GUI will generate source files which will then be compiled into profiles and sent to the client. As a result a closer look at the structure and usage of the source files is required.

Appendix A shows some example source files.

The #include(...) statement is called to include the relevant header file. Header files implement a simple form of inheritance and take all the configuration details from the relevant file and add it to the source files. Of course #include(...) statements may lead to source files being nested inside each other. The header files are grouped together by type (for example operating system or hardware platform) and placed in group directories. In machine source files there are certain selections that are mandatory, for example a machine must have an operating

[8]Abbreviated UDP, a connectionless protocol that runs on top of IP networks. UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagrams over an IP network. It's used primarily for broadcasting messages over a network.

system and as a result must include a header file defining this property. Information contained in a header file can be overridden in the source file through the use of mutation. Mutation can be used to transform any inherited resource value through the use of a regular expression that makes it easy to prepend or append items to a standard inherited resource without having to override the specification for the standard machine. Mutation is a very powerful tool but can lead to confusing and less central machine configurations and so its use is restricted to a few will-defined macros [3]

The MAC address, short for Media Access Control address is the hardware address that uniquely identifies each node of a network. The MAC address in source files is identified by the dhclient.mac statement. Inventory information is detailed in statements beginning inv.* these describe the various admin details for the machine such as who can access it and who is responsible for it. There are - understandably, a great deal of different options for use in source files. The above only touch on the most frequent.

Header and source files are very different in structure and contents. Machine files have more of a regular structure and somewhat predictable contents as opposed to header files which are very much different from one another. Machine files tend to be less complicated than header files, although comments throughout both sets are sparse.

Initially I didn't have access to the source files, they were made available (through the use of) the Rsync command. Rsync is a tool for efficiently transferring files across a network and enabled me to copy across the appropriate source files to examine. The directories I needed were:

- lcfg :: lcfgdefs
- lcfg :: lcfginf
- lcfg :: lcfgpacks
- lcfg :: edpacks

1.3 Need for a GUI

As mentioned above there are a great many options to consider and lot of information to remember when writing an LCFG source file. New Source files are written either from scratch or from an existing template file through a text editor such as emacs. Existing source files are again edited through an appropriate text editor. Experienced users find it sometimes difficult to alter or create source files.

Even with the help of the template file the job is not necessarily made any easier as the template only covers the most basic machine configurations.

Another issue is that through the years it is apparent that many different people have edited the source files and as a result there are many different files that do perform similar configuration operations but look completely different. Commenting throughout the source files is sparse and sometimes more confusing than helpful. There is also a common situation where files have been modified withoutthe comments being changed to match. Also it is easy to make mistakes when using a standard word processor and these mistakes will not become apparent until the mkxprof compiler detects the error.

Currently there is no way for someone other than an experienced user who is familiar with LCFG to make modifications to machines on the network, other than by using the limited template file. By designing and implementing a GUI for LCFG I will enable novice users to change machine configurations without having an in depth knowledge of how an LCFG source file is constructed, especially in terms of syntax and command options. The GUI itself should make creating a new source file from scratch easier by providing a structured sequence of steps for the user so that no mandatory details are left out and by getting the user to select item from a list rather than typing the number of mistakes will be reduced. An added advantage is that all source files created in the way will be uniform and similar in both style and structure and as a result will be easier to interpret when viewed at a later date.

Editing a source file will also be easier as the GUI will display a list of valid options for the user to choose from and hence will speed the editing process up and reduce the chance of error.

Creating this GUI will simplify the more mundane tasks of writing LCFG source files and will reduce the number of mistakes made. At the same time refining all of the source files into a common and easily understandable format.

1.4 Synopsis

This project was started October 2002 and ran throughout the academic year (ending May 2003). The purpose of this document is to describe not only the system I have developed but also the problems and intermediate stages that have been faced over the last year. There is obviously a great deal of work to be covered and I have presented it here in the form I think most suited to give the reader both a chronological sense of the developments that took place and also demonstrate the development process driven by the goals and aims initially set out in the next chapter.

Chapter 2 provides the reader with an in depth view of the problem and lists the aims and objectives that were key in the development process. This chapter also covers the first development phases and culminates in a detailed description of my initial design for the system.

Chapter 3 details the implementation phases of the project. The chapter contains a revision of goals and objectives that were realised in the later stages of the project and as a result of these goal changes the chapter includes a detailed description of the new design solution.

Chapter 4 details the later stages of the project and includes the results of testing and evaluation of both the system and the development processes used during the cycle of the project.

Chapter 5 is the concluding chapter of this report and summarises the lessons learnt and the degree of success of the project.

2. Initial Development

2.1 Aims and Objectives

To drive the project forward and to keep on track a clear definition of the task needs to be formed. The best way to do this for this type of project is to formulate a list of aims and goals ordered by priority. The aims are divided up into three categories: basic,extended and advanced. Those aims in the basic category are those that should be fulfilled for my system to work. Extended aims are those which i believe are attainable given the time and resources available. Advanced aims are those which I believe may not be possible given the current time and resource available but will be worked toward if all other aims have been satisfied.

At this stage there is no concrete vision of the form that the project will take but laying down a set of aims at this stage means that the project will have to be designed around these aims. Rather than writing the aims after a solution has been thought of and tailoring the aims to reflect that solution. However some aims may not be achievable once further investigated at a later stage, if this is the case the aims will be revised and a case for the revision will be made. Also some aims were added as new information came to light again the reasons for any revision of aims will be made clear.

At this stage it is clear that some form of graphical user interface will need to be developed but underneath this top layer there will need to be some way of interpreting the source files so the user can edit them and also a way formatting the data selected by the user into a source file and outputting it to the appropriate directory.

Aims and Objectives : (objectives being concrete milestones which can be verified, e.g. production of a program which can be demonstrated to perform a specific task)

Knowledge Based Aims:

1. To have an understanding of the architecture of LCFG
2. To have an understanding of LCFG source files
3. To learn to program in Perl

The measure of success of the above aims can be determined implicitly through the design and implementation of the GUI developed and also from the information contained in this report. Without achieving the above three aims it is impossible to complete this project.

Practical Based Aims:

1. The production of a program that will serve as a GUI interface for LCFG
2. To design the user interface so that it user friendly
3. To design the user interface so that it will enable a novice user to create/edit LCFG source files

The above aims are meant in a high level context, further goals and aims for measuring the success of the implementation will be presented shortly. The aims above will be satisfied if there is some system developed by the end of the project that will allow a novice user to create and edit an LCFG source file through the use of a GUI.

Evaluation Based Aims:

1. To gradually introduce the system developed and gain feedback from real users

The above will be satisfied if throughout the lifespan of the project regular evaluations are made of the system by both myself and real users.

All of the above knowledge, practical and evaluation aims are basic ones and should be completed for the project to prove successful.

2.2 Background Information

To complete the above knowledge aims information will have to sought out (and read). Below are the types of information that need to be understood along with probable sources.

What background information will be sought and from where:

1. How to program in Perl: Sources from books and Internet
2. How to program the GUI: Sources from Books and Internet
3. Understanding LCFG architecture: From LCFG documents, Staff familiar with the LCFG system
4. Understanding LCFG source files: LCFG documents, time spent examining the files and staff familiar with LCFG
5. How to design an effective GUI: Books and also advice from Staff familiar with Human Computer Interface design

At the start of the project there was a vast amount of information to digest. Certain documents from the LCFG website [1] were particularly helpful but took a long time to understand, others were less relevant but gave me a wider understanding of the structure of LCFG. I found the meeting I attended presented by Carwyn Edwards especially helpful in developing my understanding of the architecture of LCFG and also gave me a better understanding of the scope and ways in which to go about starting to design the GUI. Further meeting with Paul Anderson and Lex Holt of the School of Informatics allowed me to further grasp the architecture of LCFG and to begin to formulate several ideas as to how to go about designing the GUI. Initially when trying to gain an insight into the way in which LCFG works I spent some time trying to understand the client side component files an endeavour which was ultimately fruitless as my system is entirely based on the server side and studying the component files furthered my understanding of LCFG very little.

The decision was taken early on to use Perl as the language to develop the system with Perl/Tk[2] to be used for developing the GUI interface. With no prior Perl or Perl/Tk experience I began learning the language from scratch. The books [11] and [12] have been invaluable during the lifespan of the project. Initially I found it necessary to progress right the way through [11] to give me a sufficient background in Perl to be able to approach the problem. I also used several Internet based Perl tutorials which although not nearly as helpful still reinforced the lessons learnt from the book. I also spent some time developing sample programs to make sure that my understanding of Perl was adequate. This process took a long time but I felt that if I was to be developing a large amount of code in Perl it would save time later if my understanding of the Perl language was sound right from the start.

On the GUI programming side not only had no Perl/Tk experience and very little GUI programming experience in general with only a limited knowledge of Java Swing under my belt. The Internet was not nearly as helpful as I would have hoped it to be, the tutorials were often far too limited to be of any practical use beyond building an Interface with two buttons. The [10] book however is both an excellent way to learn Perl/Tk and also as a reference book. However given the size of the book it is not really possible to read all the way through as I had done with [11] and so I found myself constantly referring to it all the way through the project. Again it took a long time to familiarise my self both with GUI programming and additionally GUI programming in Perl/Tk.

I also did a lot of background reading as to how to go about designing the user interface. However I found later on that this material was less useful than I

[1] http://www.lcfg.org/doc

[2] Perl/Tk adds the Tk GUI application libraries onto Perl. Thus making it possible to develop graphical user interfaces

thought it would be as the design of this system has to come down to a functions grouped by type and then using a little common sense to decide on the placement within the GUI.

An outcome of the initial meetings was the decision that the size of the project would not be big enough to warrant a full scale software engineering approach to the software development cycle. However it was still necessary to follow several software engineering practises to better structure and document my code.

2.3 High Level Solution

The prospect of designing the system was overwhelming at first and the more I thought about designing a solution to the problem the more questions came up. My main group of questions to be answered were:

- What information do I need to represent?
- How am I going to parse the source files?
- How am I going to generate the source files once the user has chosen to save?
- What is the GUI going to look like?
- What differences will there be between editing machine source files and header files and how will these be represented?

The answers to these questions generated a great deal more questions and a long time was spent planning and fleshing out all the details. Essentially this brain storming approach allowed me to visualise the functions that my system needed to perform. Even if I didn't have a clear idea of how I was going to implement the system at least I now had a clear idea of the things that the system should be doing.

Examples of the types of questions that arose from the brainstorming: "our current LCFG system supports over 2000 parameters" [6]
Which ones should I think about changing first? I.e. which ones are the most heavily used?

"25% of which (parameters) routinely vary between different systems" [6]
OK so that cuts things down from 2000 to 500, that regularly change. Chance for code optimisation here?

The overall architecture for the system was established using the questions generated in the brain storming as guidelines for functionality. This architecture was very abstract in nature but then gave me a framework from which to work from.

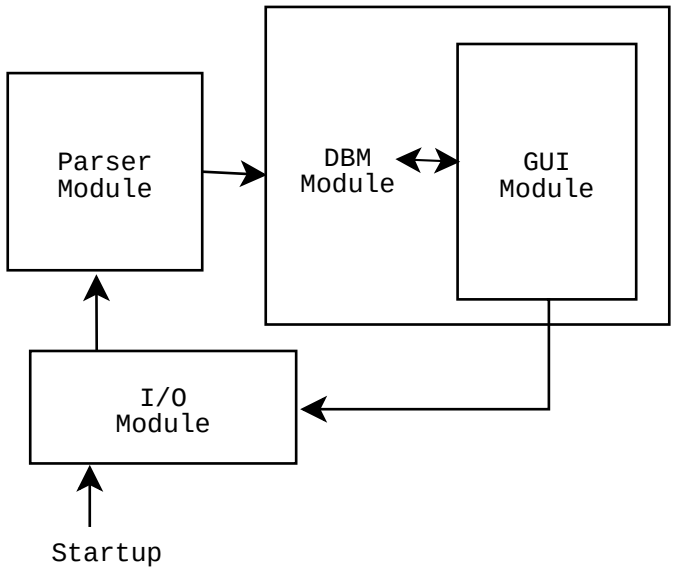

Figure 2.1: Overview Of Design Solution

The above diagram was the final design in this initial design stage and as a result is very likely to change as new information comes to light. It is clear that the architecture of the system can be divided up into four distinct areas.

- The GUI itself
- The source file parser
- The Database system (DBM)
- The Input/Output system

On start up the I/O module will scan through all the available source files and then to the parsing module which extracts all the information into a format that can be stored in the database. Once this information has been extracted - I am expecting this process to take some time as there are a great many source files, the data is passed onto the database manager which stores all the source file data. The source files will only have to be parsed once as the DBM is a persistent form of storage and so does not lose all its data when the program is terminated. There are several issues associated with database correctness which shall be covered shortly. At this point there is a store of source file data that is in an easily retrievable format.

Up until this point there has been no user interaction apart from telling the program to start, now the user must select what they want to do, this will be one of three actions:

1. Edit a machine configuration

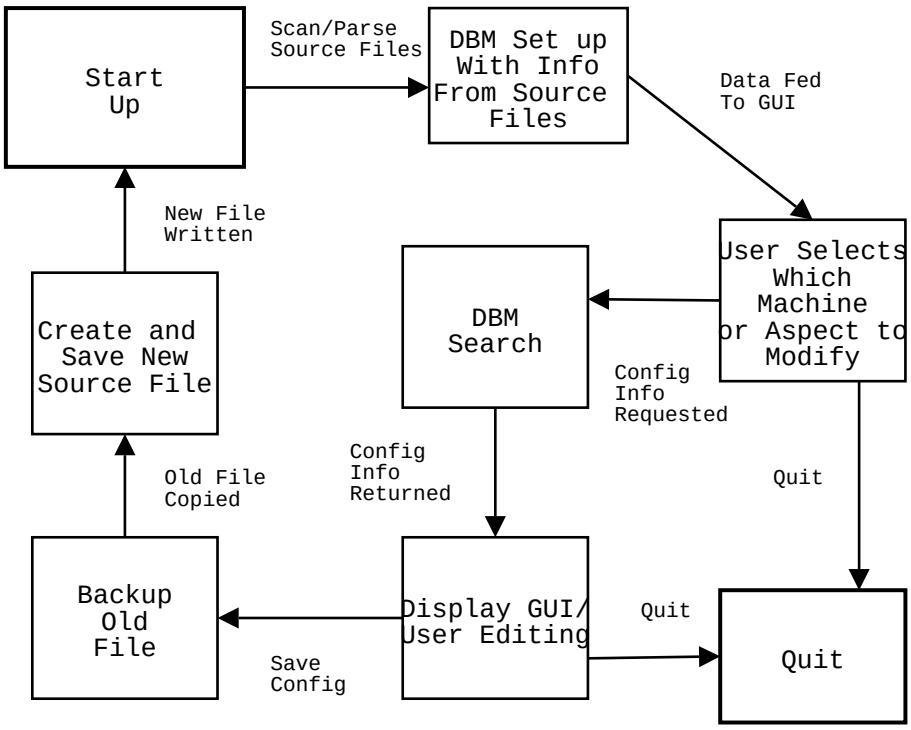

Figure 2.2: System Data Flow

2. Edit an aspect file

3. Create a new source file

If the user selects actions one or two then the will select which source file they wish to edit through some GUI representation the details for this source file will then be fetched from the DBM module. This information will then be fed back to the GUI and the details will be displayed in a way so that the user can easily edit them. If the user selects action three then a blank configuration screen is displayed. Once the user has finished editing or creating the source file they will want to save. Selecting save will cause the new source file details to be sent to the I/O module where the information is converted into a format that represents a source file and written into the appropriate directory. At the same time the DBM is also updated with the new source file details.

The above flow diagram shows the flow of data from start-up until the user is finished.

Other initial design decisions included taking a modular approach to the implementation of the system, to begin with a basic system is developed and then progressively more features will be added. The order that these features will be added will depend on the feedback from the real users.

From looking at the source files it was immediately apparent that there would be a huge amount of work to do if I hoped to parse all of the details as mentioned above there are around two thousand parameters. Writing a parser to deal with this seemed infeasible given the time constraints and the fact that there would be three other architectural sections to design and implement. After discussions with Paul Anderson and Lex Holt, it was agreed that a total parsing of the system would not be possible and so I should look to parsing the important details and find another way of representing the un-parsed information to the user.

From looking at the spectrum of tokens used in LCFG and in fact how they are used within the source files it was apparent that some features would beharder to implement than others in terms of both parsing and graphical presentation. The MAC address for example is only one line, will be quite quite easy to parse and will require only one entry field to alter. Disk partition will be difficult as there are a large range of commands on many different objects, it will also be difficult to represent in a graphical way - many user inputs will be required. Hard disk partition is far less regularly used with around 80 machines, as compared to MAC address in which almost every machine uses.

Still on the subject of the parser it was apparent that I would need to spend a lot more time familiarising myself with the structure and constructs if I was to write the parser and also output the user data in the form of a source file. Matters are not helped by the fact that source files have been edited by many different people over the years, most do not have any comments and those that do aren't necessarily helpful. At this stage the design and implementation of a parser for such a diverse collection of files in a language completely foreign to me concerned me the most.

Although the above was the final architecture decided upon, other strategies for solving the problem were considered. For example in the (very) early stages of designing a solution I considered perhaps reading and writing the XML profiles as a way of configuring the machines. However it was quickly realised that this wasn't possible and in fact would have not changed any of the data on the server side and so any changes that I did in fact make would be lost.

Using the above design it seemed feasable taht I could achieve my aims. Also the modular nature of the design means that it can be altered or extended as neccessary.

2.4 Testing and Evaluation of Proposed Solution

To measure the success of the project methods of evaluating the finished design need to be developed. This will not only help define the problem but also and more importantly judge whether the project has been a success. Testing is of course vital to any software project to ensure quality and to check that the system behaves as expected. Thus it is important to think of testing methods at this stage and carry out testing throughout the software development process.

The system will be evaluated throughout the development process by real users. Several iterations are planned and at each stage I hope to receive feedback as to what features should be added and which need more work. I also hope to test my solution on novice users and get feedback from them.

The aspects of evaluation will be:

- Initially method to change individual machines then progressing to aspects: enables me to check that the GUI is progressing in the right direction
- 1st round of feed back from Lex Holt/Paul Anderson, as is only basic implementation feedback from real users wouldn't be all that helpful
- Need to formulate a way of getting feedback so that it is useful and complete
- Trying to construct prototype system
    - Simplified set of source files
    - Configuration of one machine only

The initial prototype will be very much a limited system. The user will be able to alter the configuration of individual machines. From this the user will be able to select which header files they want to include,any additional specifications may be entered from a text box in the user interface. The text box will currently be held under the Extra tab in the LCFG Configuration window. This basic design is there so that I can make sure that all the basic elements of the system are working appropriately before I go on to add more advanced features.

Initial feedback will be from Lex Holt and Paul Anderson, it would have been possible to get real users to evaluate the interface at this stage however I felt that as it is such a basic implementation and as a result the functionality was limited that the feedback would not be helpful.

Once the initial prototype is running, I will question the real users of the system which features they regard as most important. From this list I will be able to order my list of advanced features to develop on top of the prototype system. At

several stages I will get the real users to evaluate what I have done and revise my design if necessary. At this stage some thought was spent thinking of how to best get the support staff to evaluate the system and hence receive structured feedback.

For testing I will not only need to test my system in a real set of LCFG source files but also use source files developed by myself that will test the limits (boundary constraints etc) of the system and give me an accurate indicator of performance throughout the development lifecycle. Also by making the system as modular as possible unit testing will enable fast error detection and correction.

2.5 Similar Systems

To help conceptualise the design problem a variety of other network management tools were investigated. Some designed in Perl othes not. Although none of the intefaces examined were for LCFG they still provided a valuable insight into how to design the user interface. None of the systems examined used a similar architecture to the system I planned to develop and so other than aiding me with interface design the review of other systems was of limited use.

LaptopNet[3] is a laptop network management tool and gave me some ideas on how to lay out the GUI. The interface is written in Perl/Tk and so was particularly interesting to examine. MOSIXVIEW [4] is a GUI for managing openMosix-Cluster networks and again gave me many ideas on GUI layout. NOCOL[5] is a popular system and network monitoring (network management) software that runs on Unix systems and can monitor network and system devices. It uses a very simple architecture and is very flexible for adding new network management modules. NOCOL is written in Tcl/Tk and so isn't as directly relevant as the above two examples.

Also [4] was extremely useful in gaining a backround knowledge of network configuration tools but had little direct bearing on the project.

2.6 HCI Research - Summary

Time was spent familiarising myself with guidelines and literature that would allow development of a GUI that was easy to use and intuitive. This section contains a very brief summary of some of the work that was carried out into

[3]http://www.imfc.edu/ daniau/laptopnet/
[4]http://www.mosixview.com/indexen.html
[5]http://www.netplex-tech.com/software/nocol/

interface design. The reason being that the research was not to prove directly useful and so as a result is only touched upon here.

Know the user and match the interface to the user's knowledge and tasks - this guideline has very much been kept in mind throughout the development process. Provide means of cancelling actions. Require confirmation of actions which have irrecoverable consequences. A good interface will have the following properties: [5]
Learnability:

- Predictability: the systems behaviour is observably deterministic
- Synthesisability: the user caan assess the effect of past actions
- Familiarity: match the interace to teh users expectations

Flexibility:

- Dialogue initiative: give user control of dialogue flow
- Multi-Threading: provide support for simultaneous tasks
- Task Migratability: negotiability of function allocation between user and system
- Substitutability: equivalence for different forms of input expression
- Customisability: interface iscapable of being adapted to suit different needs

Robustness

- Observability: relationship between the system state adn its presentation
- Recoverability: support for undoing errors
- Task Conformance: interface functionality should match common user tasks
- Responsiveness: feedback should be evident with action

2.7 Detailed System Design

First will be a general overview of the system, a description of how the four modules will interact and also how the user interacts with the system. The description below is similar to the one given in the preceding section but is here to remind the reader of overall architecture before the specifics of the system are given in detail.

The user will type in the command to start up the system or conceivably from a graphical menu. The system will then check to see if the database has been

previously constructed if not then the input module is run. The input module finds all the source files and feeds them to the parser which then parses the data into a format that can be transferred to the DBM, this data is then fed to the DBM. A GUI will now display on the screen enquiring what the user wishes to do, edit an existing machine source file, edit an existing header file or create a new source file - specifying whether header or machine.

A new screen will then display on screen allowing the user to edit the respective source file. Once the user has finished editing the source file they will press a save button and the GUI information will be fed to both the DBM and the I/O modules. At the DBM the entry for the corresponding source file will be updated. At the I/O module the information will be processed into a representation of a source file and written into the correct directory.

To begin with I drew several sketches of what I believe the layout of the interfaces should be like. There will be two main screens: the initial screen where the user selects what to edit and the configuration screen where the user edits the source file.

The configuration screens for header and source files will be very much different as they take very different forms. One of the aims of the project is to develop a system that is easily understood and easy to use. Therefore time was spent researching the ways in which best to layout and design a GUI.

Once the user has finished configuring the source file and clicked on the 'save' button the current source file will be written to a backup directory and the new one written in its place. This is a mechanism so if that the user selects wrong configuration options then they will be able to restore the previous settings.

This system is not intended to provide all the functionality that a user will need to be able to develop every type of source file, it is intended more to a framework that can be expanded upon as time goes on. Therefore the system code will need to be well commented and easy to bolt extra functionality onto. There are several advanced features that would be useful additions to the GUI, given the timescale of the project I will not have time to implement all of these but as mentioned above in section 2.4 the extra functions implemented will depend on what the users deem more important.

The main conceptual problem posed by this project is how to represent a LCFG source file in a graphical way (as a GUI interface).

2.7.1 Parser Module

Inputs: Source file data from I/O module
Outputs: Parsed source file data to Database module

Parser Module Aims and Goals:

- Parse the information from the source files
- Put this information into sensible data structures
- Efficiency both with time and memory
- Handle any source file, this means data in any order, and parser may come across unexpected tokens.
- Send source files data structures to the DBM module

From the start writing a parser seemed a daunting task, this was made a lot worse by the worry that it would have to cater for anything around two thousand different parameters. However initial discussions focused on the abstract level required for the project and that writing a parser to do the above would be infeasible. Also the amount of information that would need to be displayed on the GUI would not be possible. A medium ground had to be established and so the parser would evolve with the development cycle and only as much information as needed by the functions implemented in the GUI would need to be parsed.

My first somewhat lazy approach to the design of the parser was to not actually parse it all and simply send the file to the GUI and display it in a separate window. This way the user would still be able to reference the source file and so be able fill in the details into a blank GUI configuration screen. However this approach would have wasted a lot of user time especially if only one parameter was changing in a source file and with the time available I judged that it was feasible to implement a more complicated solution.

Initially I considered using a mark-up language so that a series of tags could be added to all of the various parameters that the system needed information for. That way the parser would simply have to scan through the source file identifying tags and then storing the information. The markup language would have to be designed in a way so that it was easy to understand and was easily expandable. However on further reflection it seemed that marking up source files would not be the solution. There are around 700 machine files alone and marking up each of these individually would take a long time and would be prone to error. Hence using a markup language of this sort wasn't feasible.

The third method was to use a pure parsing approach in that the parameters have a unique identifying command and so although it would be more work than using markup tags it would be possible to use pattern matching to find the pieces of information needed to feed the DBM information. The parser will read in one line at a time and scan that line for recognisable tokens, if found that token can be deleted and the information can be stored in a variable and sent to the DBM.

The parser will only run periodically after initialisation. So efficiency will not be the primary concern - correctness of the parser is. As there are so many files to parse I am expecting that the user will have to wait some time whilst the parsing is in progress as a result some form of progress bar should be displayed so that the user does not think that the system has crashed.

2.7.2 GUI Module

Inputs: Source file data from DBM, user configuration input
Outputs: Edited source file data to DBM module and I/O module
GUI module Aims and Goals:

- Use PERL/Tk toolkit to implement the GUI
- Use the parsed source file setails to set the defaults in the appropriate GUI entry fields
- Organise and structure the grouping of functions so that they aid user comprehension and ease of use
- Organise the layout of GUI objects on screen so that they aid user comprehension and ease of use

Initially trying to envisage what the GUI would look like was very difficult. There would be a lot of information to display and a lot of complicated data would have to be manipulated to set entry box defaults. To start with an initial screen will appear asking the user which source file they wish to change from here they would click on accept and proceed onto the configuration window. The configuration window is where all the editing will be done. It was difficult to think about how to design the configuration screen so the problem was broken down into smaller steps.

The layout problem was tackled first. As there was so much information to be displayed there was no hope of displaying all of the information on one screen this would be both overwhelming and unusable for the user. Displaying the information over multiple screen was a necessity. Several options were then open to me;

- Multiple windows could open simultaneously on screen and the user could iconify/de-iconify as needed
- Multiple layers could be used where the user would start from a main window and select options that would open up a further window leading to the information that they wish to edit - the layers may be nested.

- Tabs could be used where the user selects a heading tab and that information is displayed, essentially only one window is open but within that window are several others.

The first two options would lead to the screen being cluttered and may be confusing. The first approach especially would be overwhelming for the user. The second approach may be frustrating in that the user may have to search all the way down a series of windows to get to the information they are looking for and the number of windows generated may become unmanageable. The third option keeps the screen uncluttered whilst at the same time being highly visible and easy to navigate.

With the layout scheme chosen the next decision was how to divide up the parameters to put them into tabs. Initially I thought that it would be sufficient to divide up the parameters up into header files and text, so that all of the header files would be under one tab and everything else would be under another. After examining the source files again I decided to break it down further into software, hardware and text. Thus both the parameters that affected software would be placed in the software tab, hardware in hardware and everything else would go in text.

Using the machine template file (See appendix B) as a guide for initial prototype system functionality I was able to construct a rough idea of what the GUI would look like. However as mentioned in 2.8.1 it is not possible to parse all of the parameters that LCD caters for. Just because the parser module doesn't parse them doesn't mean that they can be ignored however. The parser will place any unrecognised lines into a series of scalars and the GUI will place the contents of these variables in some form of text widget that the user can edit. This way even if the functionality has not been implemented for a certain type of parameter that the user needs they will still be able to use the GUI to develop the source file. Hence another tab will need to be added to contain this text widget. Examples of the type of text that would be contained in the text widget are:
!auth.users mADD(All)
!xfree.monitor mSET(dell14u)

Also another item previously uncatered for was user comments within the source files, there was also no method for the user to add comments to the source file from the GUI. In much the same manner another tab was added that contained a text widget. This text widget would display any comments contained in the source files and from there the user could edit and add their own. This design decision had repercussions for the design of the parser that will be discussed later.

I looked into the options that PERL/Tk made available to me to design the GUI, there are a staggering range of widgets[6]. The main tools that I will work with are: tabs, scrollbars, keyboard shortcuts, autosave, drop down menu, radio boxes, check boxes, multiple windows, colour coding, greying out entry fields, layout design, grouping of common functions, menu bar.

Selecting mandatory header files would require some kind of drop down box, the list of header files to go in this widget is supplied from the DBM. This way the GUI will stay up to date as opposed to if the list of header files were to be hard coded into the GUI. Optional header files would require some kind of selectable list or checkboxes. Again the list of header files will come from the DBM. The other parameters would require some entry widget for the user to enter in text.

2.7.3 DBM Module

Input: Data from parser, update information from GUI
Outputs: Source file information to GUI
Aims of the DBM module:

- To provide quick access to the data contained within a source file without the need to parse it
- To provide persistence of the data generated by the parser.

A Database Management System is a collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMSs, ranging from small systems that run on personal computers to huge systems that run on mainframes. Perl provides access to the DBM through the use of a hash associated with the database through a process similar to opening a file. This hash (called a DBM array) is then used to access and modify the DBM databse.

The DBM will store all of the parsed source information, this is because I want to be able to access the source file data quickly without making the user wait whilst a source code is parsed. The DBM will have to update itself whenever a source file is updated, at the moment if the source files are changed out with the GUI there is no way of the DBM knowing it has been changed. This problem shall be covered in more depth shortly.

The next design decision to make was how many DBMs to use. Would one large DBM suffice or would several smaller ones be more efficient. This required investigating more into DBMs and the formats that I could use to store data.

[6]PERL/TK term for an object, this can be a button, some text or even graphics

From the options that the DBM provided storing the source file data in structured way would be possible. There were other options than using DBM, SQL could have been used instead and would have been a more powerful tool but doesn't interact as well with Perl as the DBM. From reports on newsgroups that I found there were several opinions that voices concerned as to whether DBM could handle over a thousand records reliably and so as it was not difficult to divide up the parameters into groups- having already done so for the GUI interface, I opted to have several smaller DBMs. The penalties incurred for having to spend time opening and closing links to multiple DBMs will be offset by the faster seek time for entries in the smaller DBMs.

Organising the data in an efficient, manageable and visible way is a difficult task especially given the nature of the source files. For some items such as the unparsed information it is impossible to refine this to be more efficient and so the DBM will have to store very long strings. Another design decision is the permission for DBM files. The DBM generates two files a .pag and a .dir. Leaving these open to anyone would pose a security risk and so the permissions for them will be set to the user only.

The worry with the DBM is that if a source file is changed out with the GUI then the DBM will be incorrect, this could lead to all sorts of problems. Methods have to be developed to solve this problem if the system is to be of any use. There are several ideas that I will put into practice

1. Every nth run of the system re-parse the source fikles and update the DBM
2. After a set timeperiod has elapsed since the last DBM update re-parse the source files.
3. A reparse button in the GUI so that the user can manually reparse the source files.

Also adding a remove source file button would be good to ensure DBM integrity although it may be difficult to get user to start up a GUI to delete a file when they can do it much quicker through a shell.

2.7.4 Input/Output Module

Inputs: raw source files, updated source file information from GUI
Outputs: raw source files to parser module
Parser Module Aims and Goals:

- To navigate the various LCFG directories and forward all of the source files to the parser module

- To receive the edited source file information from the GUI module and use the information to construct a source file
- To backup the old source file and then write the new one into the appropriate directory.

The first time the system is run the I/O module will create a backup directory for old source files. This is so that if a user makes a mistake whilst editing they can always revert back to the original. When requested by the parser the I/O module will scan through the various LCFG directories and locate all the source files and then pass these onto the parsing module.

The other task of the I/O module is to take information from the GUI and convert it into source file format. This format will be well commented, ordered and uniform. It is the hope that by using this system over time all of the source file will take on a similar format and so will be much easier to understand.

2.7.5 Timetable

The initial rough timeplan for the project is below. I was expecting this to change as gained a better idea of what the project involved and also as I ran into problems and set backs.

Term 1 Week 1: Receive project
Term 1 Week 2: Background reading and problem definition
Term 1 Week 5: Initial design for prototype
Term 1 Week 7: Begin implementing prototype system whilst still learning
Term 2 Week 1: Initial Prototype to be scrutinised by Lex/Paul
Term 2 Week 3: Partial Implementation given out to real users for feedback and advice on what features they think are most important
Term 2 Week 7: Implementation of GUI asked by for real users, get more feedback
Term 2 Week 10: Implementation of GUI asked by for real users, final round of feedback
Term 3 Weeks 4 - 7: Finishing Touches and Dissertation
Term 3 Week 8: Presentation

3. Implementation

Throughout the implementation phases of the project goals had to be refined as new information came to light and the design of the system also needed to adapt to meet these changes. This chapter lists the changes that took place over the implementation period and also describes the reasons for these changes. Also the final design for the system is presented in detail. There were several intermediate designs but the details for these are not included as a notion of these stages can be gained from the revision of aims and design decisions. These intermediate stages also do not differ enough from either the original design or the final design to warrant an in depth description.

3.1 Revision of Aims and Objectives

The development of the modules proceeded slowly at first as I got used to programming in Perl. The main problem I found with Perl when compared to java (the language I'm most familiar with) is that there are almost no supported libraries. Java provides a huge range of methods that can be called from your program which means that you can develop a program very quickly and easily without having to labour over minor programming tasks like finding the largest element of an array. This is not the case in Perl and although modules can be downloaded [1] it means that any machine that wants to run the program would have to download and install the same modules thus the portability of the program would be very much limited. Throughout the project the books [10] [11] [12] of Perl books have been invaluable. A lot of time was spent studying these books trying to gain an understanding of how to solve the various problems that I encountered. I think the pace of the project would have been a lot quicker if I had previous PERL programming experience.

The modules were developed independently of each other with stub classes representing the other modules. Once the module was ready it was transferred across into the main program, the main program being the GUI module - as this is central to the architecture of the system. To start with I had four fledgling systems that didn't provide much functionality, as the modules grew it became apparent that the design of the system needed to change.

The parser worked well and much to my surprise worked a lot faster than I had thought it would. This fact coupled with my concerns over DBM integrity and scalability issues led me to reconsider using the DBM module. The system would

[1] http://www.cpan.org/ is the primary site for downloading Perl modules

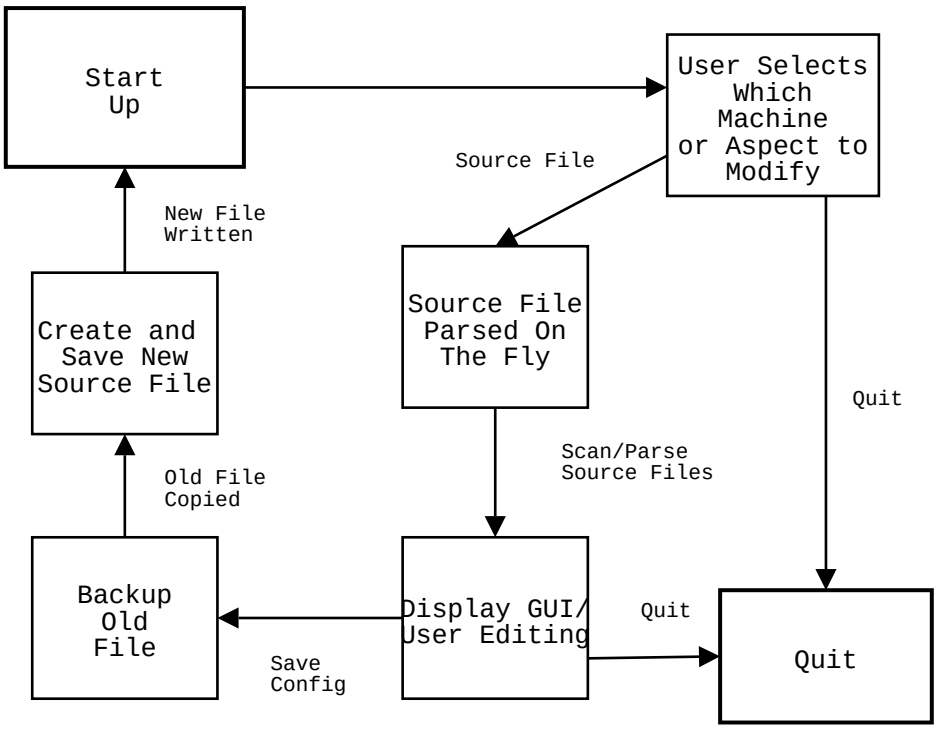

Figure 3.1: Overview Of Design Solution

not parse all the source files at start up but simply just parse the source file that the user wants to edit. This design would not have any less functionality than the original design but would require that the parser module was kept efficient. If later on in the design stages it was apparent that I needed some form of data persistence then I could use the already developed code from the DBM module to implement this, as it turned out I wouldn't be needing it.

The interface developed rapidly and evolved to take into account added functionality. This mainly stemmed from the template file (see appendix B) so that machine source files could be edited. Once basic machine functionality was implemented I turned to developing the header file interface. The old problem of header files being very complicated came back to haunt me. An oversight on my part was that I got carried away developing the machine side of things and focused less on the header files. As stated earlier there is very little structure and consistency throughout the header files as a result it is an extremely difficult conceptual problem to visualise an interface that would benefit the user. After discussions the decision was made to focus more on the functionality on the machine side as both parsing and displaying header file information was deemed infeasible.

I would not be ignoring the header files in my implementation again I stress that this is a framework for future expansion so at the moment there is a 'bare

bones' editing interface that will allow construction of header files but not in as structured an environment as machine files.

Any header files contained in the source file are parsed and pre-selected and all additional text is placed under the text tab, again any comments are placed under the comments tab. As justification of the above design header files should probably only be modified by experts and as a result shouldn't have to be guided too much when configuring a source file.

The completion of the basic system was achieved early in term two with evaluation from both Lex Holt and the support staff up at Kings Buildings. Feedback was good and from this I was able to order a list of extra features that could be added to the system. The details of these extra features will be presented later on in the chapter.

Throughout the project I was required to stand back and realise that the commenting had started to become quite sparse and that variable names had become a little obscure and so it was time to tidy up my code not only for the sake of my sanity but also anyone who wants to develop the system further at a later date. This did take up a lot time but were necessary otherwise the code would have become obscure and unintelligible.

3.2 Revised Timetable

To accomodate these design changes and also the slow progress at the start of the project the timetable was altered as follows:

Term 2 Week 1: Initial Prototype to be scrutinised by Lex Holt/Paul Anderson
Term 2 Week 3: Partial Implementation given out to real users for feedback and advice on what features they think are most important
Term 2 Week 7: Implementation of GUI asked by for real users, get more feedback
Term 2 Week 10: Implementation of GUI asked by for real users, final round of feedback
Term 3 Weeks 4 - 7: Finishing Touches and Dissertation
Term 3 Week 8: Presentation

3.3 Revised Systems Design

3.3.1 Parser Module

The development of the parser took a long time, the design had to be carefully thought out so that it was easily extensible as functionality was added. Initially the parser only dealt with finding header files in source files. Then was adapted to finding amongst other things the mac address and inventory information. The parser works by reading in each line of the source file and looking for a key token. If a key token is found on a line it means that data can be extracted from it and the information stored in a scalar or array. The next development stage for the parser module was enabling any text that wasn't recognised as having a key token and placing it in an array of scalars so that it could be displayed in the GUI.

Next the ability for the parser to recognise comments was added. This was particularly challenging as it meant that the parser had to accept the various forms that comments could take. These are /*...*/ and /* ...*/ and /**/ with nothing in between and /* ...*/ making sure that no comments were missed or that nothing that was supposed to be a comment showed up as text was quite hard work and required several test files full of comments to make sure that the system was working as expected. There were several bugs throughout the code that I have remedied such as vast amounts of blank lines getting saved as comments. When a source file is written through the GUI, the information output source file is written in a structured and well commented manner. This posed a problem though when parsing files that had already been edited with the GUI. Comments generated by the GUI would get picked up in the parser and stored in the comment array. This is fine but when the user saves the source file a second time another lot of generated comments are output to the source file.

The solution to solve this was to return to the idea of using a mark up language, only on a much smaller scale - one markup token in this case. Now when the source file is written by the I/O module a token "-+-" is included at the start of the comments. When the parser sees this token it discards the comment.

The only problem remaining with the parse that I would like to improve is that all the comments are written in one paragraph at the start of the source file. Hence all the comments written lose their place. This is obviously unsatisfactory but would require substantial effort to get the parser to track the position of the comment and then adapt that position as the source files grows and shrinks. As a result the comment tracker will be one of my advanced features that may be implemented at a later date.

One success story was the ease of integration of the parser and GUI module.

There were a few teething problems but other than a few mis matching variable names everything went well. It was good to see that the forward planning I had put into the design of the modules paying off. It was also good not to have to spend hours hacking away at code trying to find the source of the errors.

3.3.2 GUI Module

At the start of the project I thought I would have a lot of control over the layout of the GUI, this assumption was to prove false. Strictly speaking the layout of the GUI has been determined by how I have grouped parameters together. Initially there were to be three tabs; hardware, software and misc. A text tab was then added to this to allow the user to add in details that were not supported by source GUI. Then a comments tab was added in to allow the user to add and edit comments. The inf tab was put in to accommodate the the additional source files from the inf directory (the inf directory contains files that are used by informatics specific staff) and HD partition was added as a result of implementing one of my advanced features.

When designing the system I was tempted to use modules from CPAN, a Perl resource site that has a library of methods to suit a common purpose. I decided against using CPAN modules as it would reduce the portability off the system as every machine that wan't to use the system would have to have these modules installed as well.

Within these tabs it was a simple task of deciding whether the form of entry would be a check-box, drop down list or an entry box. The actual placement of these widgets was left as late as possible because by using the Perl/Tk pack manager it is very difficult to position the widgets exactly where you want them. So it was my decision to wait until the end so that all the widgets had been created so that I could spend time on the layout. So now the functions and widgets were split up over several tabs most of the layout work was done, all that was required was to place the widgets into a sensible layout. Drawing concept sketches for the layout was to prove no problem. However getting pack to place the widgets as required was.

There are three geometry managers in Perl/Tk: pack, place and grid. After reading up on the three I choose to use pack. However pack gives you very little control on the exact placement on widgets. So to get a widget where I wanted it to go required using lots of nested frames. It is possible to use two geometry managers at the same time but it can cause problems with the GUI if it is resized or if more widgets are added in the future.

Following the guidelines laid in section 2.6, I added a menubar at the top of both the initial screen and the configuration screen. These contain Save and Exit under

the File header and, Help and About under the Help heading, keyboard mappings have been provided for these functions. Although there is little functionality in the menu bar again I thought that it should be added so that more function may be bolted on at a later stage. The help system was developed mainly from the user guide (see appendix D) and may of course be revised at a later date but I felt it necessary to include some form of online help in case the user gets stuck. Not everyone will interpret the GUI in the way I have intended. It is the hope that the GUI will keep users on the right track but we can't always guarantee that they will.

Again according to good HCI guidelines widgets that can't be edited due to some combination of selected settings are greyed out, indicating to the user that they cannot change the contents of the box.

When the user is in the configuration screen they cannot open up another source file to edit, the accept button has been disabled as long as the configuration screen is open. On the initial screen the user cannot click on accept if a machine or header file has been chosen. When the user wishes to save the changes they have made to a source file a dialog box will pop up asking them to confirm that they want to overwrite the source file, this is in case the user accidentally presses the save button. If the user has entered the configuration screen and then presses cancel a dialog box will pop up asking the user if they want to lose all of their changes and exit. This is again so that if the user presses the wrong button they have a chance to go back.

When the user chooses a machine or header file the configuration window will open in addition to the old configuration window. This window displays the old source file so that the user can reference it whilst editing the source file. The text in the old configuration window has been made read only so that the user may copy and paste information into the GUI but cannot edit the file.

The above figure shows the state chart for all states that can be reached by the GUI.

3.3.3 Input/Output Module

There has been very little change to the GUI module from the original specification. The module itself was very simple to implement and I found directory manipulation to be a strength of PERL. Writing a source file out to file constantly changed as new features were added, with a large revision when the markup token was introduced.

Creating a format for a standard source file was another conceptual problem faced. I envisage the GUI as both helping the development of source files and

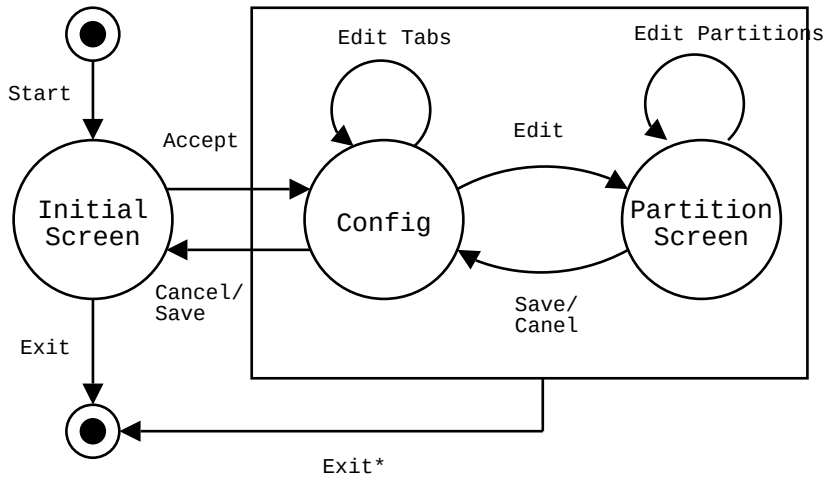

Figure 3.2: GUI State Chart

also going some way to standardising their layout and structure. Therefore it was necessary to think about how the I/O module would write the source file, what comments would be auto-generated and also leave the option for later expansion.

3.4 Procedure Description and Data Flow

Figure 3.2 shows the flow of data throughout the system. Below is a description of each of the procedures used and also a small sample of pseudocode (if required) for explanation.

Main
Not really a procedure and all programs must have a main method to run. At this stage the user ID and the system date are retrieved and stored. The directory_search procedure is called and once that has returned, fill_intial_window is called. The *mainloop* function is called at the end of main, this is necessary to start PERL/Tk functions running.

center
This routine is called by windows in the system to position them in the center of the screen.

directory_search
Generates a progress bar which updates the progress as the various directories are scanned and the names of the files are placed in the appropriate arrays. I have to make slight admission here: When I first implemented the progress bar the speed of the system was such that no sooner had the progress bar been drawn than it was destroyed because directory scanning had finished. I thought the progress

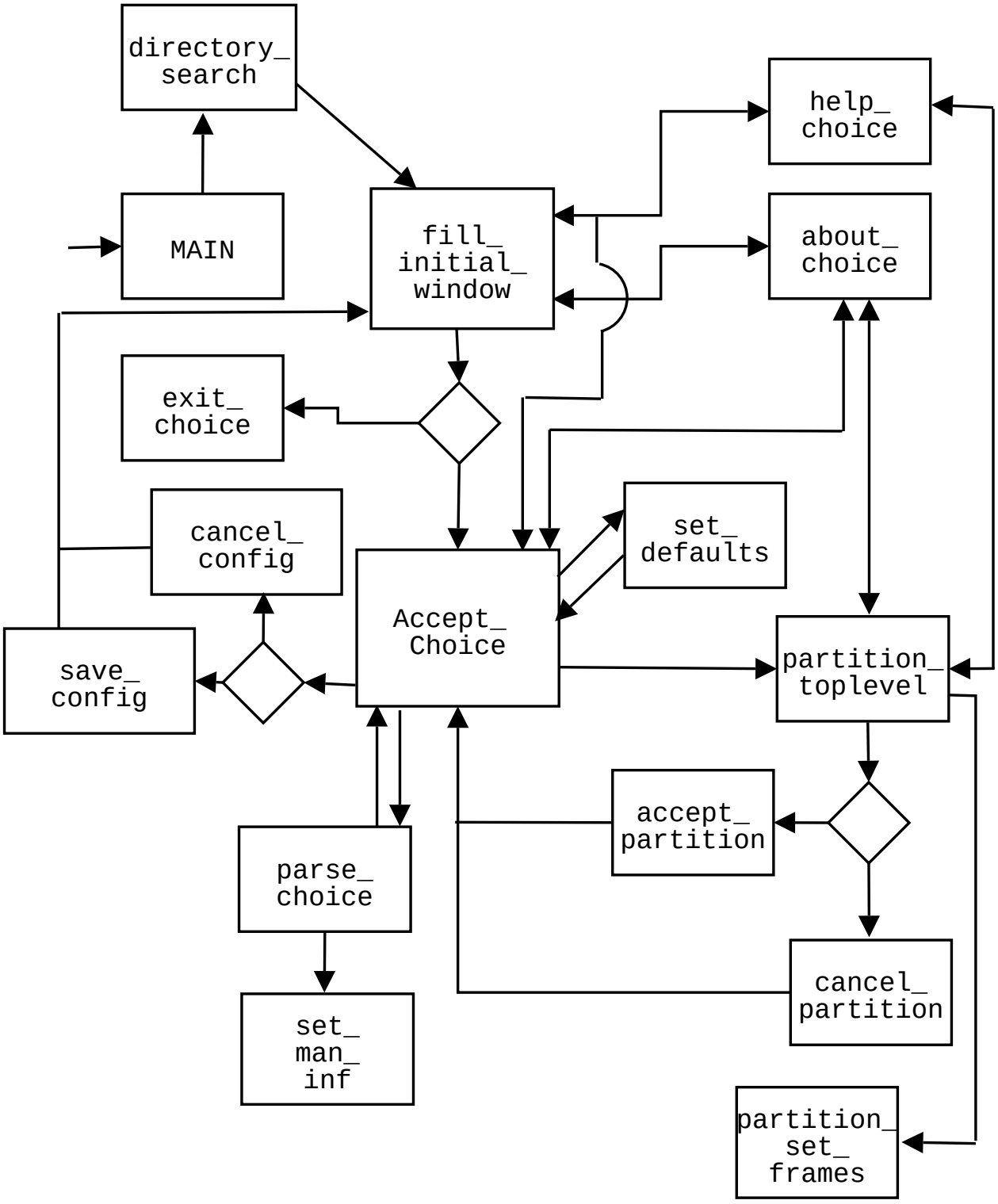

Figure 3.3: Overview Of Design Solution

bar looked quite good and so I added a couple of sleep statements[2], they can be removed very easily but I did spend a little bit of time developing it and so I thought it would be a shame not to include it.

Originally the progress bar was to appear to fill the time that was taken to parse the source file between the user clicking accept and the configuration screen displaying but the wait for this was even less than the time taken for directory_search to complete. So it was included in the directory_seacrh procedure

The main body of the procedure scans the relevant LCFG directories and stores all of the names of the available source files into arrays grouped by function. The method also creates the directory /old_source_files if it hasn't already been created.

fill_intial_window
The first thing that this method does is destroy the progress bar created in the directory_search procedure. The main job of this procedure is to use the information from directory_seach to produce the first GUI window initial_screen. see appendix A for screenshot. all of the machine files are placed into the machine list box and all of the header files are placed in the aspect entry widget. The user can either type in the name of the file they wish to edit or they can select the file from the drop down list.

The menu bar is created, from here the user can select exit from the file menu and about and help from the help menu. The menu bar is easy to expand upon if extra features are added. The widgets and labels are then added. The action taken by the system if the user presses the exit button is to call the exit_choice procedure and accept_choice is called when acept is pressed.

parse_choice
This method first calls the set_man_inf procedure. It next opens the filehandle to the user selected file. The parser then deals with comments, recognised tokens and unrecognised text. [3] The pseudocode for the parser is shown below:

```
declare @comments array;
set  $comment_flag == 0;
set  $comment_flag == 0;

FOR each line in the source file
{
 chomp line
 IF line isn't blank DO
```

[2]the sleep(*time*) method in PERL makes the system wait for a time indicated by *time*
[3]This PERL function removes the carriage return from the end if the line

```
{
    IF comment_flag == 1 DO //already in a comment block
    {
         IF line ends with */ DO
         {
          then this is the end of the comment block
  append this line to $comments[$comment_element];
  set $comment_flag == 0;
  set $comment_element += 1;
 }endif
         ELSE
         {
    still in comment block which doesn't finish on this line
    append this line to $comments[$comment_element];
         }endelse
    }endif
 IF comment_flag == 0 DO //not in a comment block
 {
  IF line starts with /* DO
  {
  IF line ends with */ DO
  //commemt begins and ends on the same line
            {
      chop off /* and */
      add comment to array of comments @comment;
      increment comment_element += 1;
      }
  }
  ELSE
  {
     //comment does not begin and end on same line
             chop off /* ;
     add comment to @comment;
     increment comment_flag += 1;
  }
 }
 IF recognise token DO
 {
   determine token and store data;
 }
 ELSE
 {
   store as text;
```

```
  }

 }endif
}endfor
```

exit_choice
The main task of the exit_choice procedure is to provide the user with a pop up reminding them that their changes will be lost if they exit from the configuration screen. If the user is on the initial screen then there is no information to lose and so the program exits normally.

about_choice
This procedure creates a pop-up containing information about the project

help_choice
This procedure creates a pop-up containing the user guide about the project. See appendix D.

accept_choice
The accept_choice procedure is launched when the user presses the accept button on the initial screen. First the procedure checks to see that there isn't already a configuration screen open - if there is then it does nothing. It then checks to make sure that a file has been selected on the initial screen. If a file has been selected then the parse_choice procedure is called when this returns the configuration window is created.

Next the configuration Top level[4] is created then the old_configuration Top level. The old_configuration window displays the old source file by opening a filehandle to the source file, reading it in and then closing the filehandle.

The menubar is then created and then the Notebook PERL/Tk package is used to create the tabs on the configuration screen. depending on whether the user choose to edit a header file or a machine file different configuration screens will be generated.

If machine was chosen the following tabs will be created:
software, hardware, inf, misc, fstab, text, comments
If aspect was chosen the the following tabs will be created:
headers, text, comments

The tabs themselves create appropriate widgets and fill drop down boxes and list boxes from arrays generated from the parser. Special options had to be enabled for the listboxes to allow more than one selection to be made at a time and also

[4]A TopLevel is similar to a new window but is created as the child of the parent window - in this case initial_window, this means that if initial window dies so does configuration window

to allow more than one listbox at a time to have a selected value: selectmode = multiple and exportselection = 0.

The only tab that requires special attention is HD Partition. When the user presses the configure button a call is made to partition_toplevel which is described below.

save_config
A pop up is created to ask the user if they want to overwrite the source file. If yes the procedure continues, if no then the procedure does nothing. If the user selects yes the old source file is written to the backup directory. Information is retreived from the GUI and written out to the source file. The procedure finaly destroys the configuration and the old_configuration windows.

cancel_config
This procedure destroys the configuration and old_configuratation window

set_defaults
This procedure pre-sets the entries to the list boxes, drop down menus, radio buttons etc in the GUI. The procedure will follow one of two routes:

1. The source file is a header file therefore match all included header files into the listbox under the headers tab and select those present using the selectionSet command.
2. The source file is a machine file therefore the process is broken down accross the various tabs to set entry widgets, radio buttons and listbox items.

set_main_inf
This procedure keeps a list of all the mandatory site wires and site specific details. They are kept in a separate procedure so that they are easy to change.

partition_toplevel
A new Toplevel is created - the child of the configuration Toplevel, the number of tabs created in this window is dependant on the radio button selection from the HD Partition tab in the configuration window. partition_set_frames is called. Then when this procedure returns the widgets are created.

cancel_partition
The cancel button will result in the partition window being destroyed.

accept_partition
The accept button will result in the window being destroyed and an edited flag being checked so that the I/O module knows that the information has been edited.

partition_set_frames
There is a quite a complex frame layout for partition window and so a separate

procedure has been used to set up all the frames that are required for the partition Toplevel.

3.5 Intermediate User Feedback and Actions Taken

It was very important as I had no experience developing source files when I began the project to receive advice and feedback from those who did have a lot of experience - those who would eventually be using the system as well hopefully. Those I sought advice from was principally Lex Holt followed by the support staff up at kb and also Paul Anderson. Their feedback on the initial prototype I had developed directed what features should be implemented and which wouldn't be used.

After discussions with the support staff at James Clerk Maxwell Building, Kings Buildings a list of useful features was obtained and also some advice on the layout of the GUI. The main topic discussed was the implementation of a function that could help them partition discs in LCFG. This required further research and after a brief evaluation as to whether the extension to the current system was feasible I began to implement.The details of this extension can be found below.

In addition the staff would have liked to have seen functions to encable them to edit auth, printer, and vmware parameters.

3.6 Additional Features Implemented

There have been many features and functions that were not included in the basic system and these have been covered in the previous design sections. The features here are the ones that took a lot of time or I felt were particularly important.

LCFG source files have the ability to override the disk partitions on a machine. (appendix B) Shows an example of a source file with this property.

There are currently 89 fstab overwritten source files out of 706 so with about 12% of LCFG files and the backing of the support staff I got underway developing the extension. The commands are quite difficult to understand and again it became apparent that the GUI would not be able to cater for every taste. Some of the fstab commands would have to entered manually. The number of fstab commands took me by suprise and suddenly I had a lot more work to do than I initialy thought. Each machine can have up to four physical hard discs and each of these can have up to four primary partitions. There are a range of commands that

can be used on each partition and so trying to decide on a detail level was very difficult.

Conceptually trying to visualise how I would lay out the information so that it made sense was a very difficult task.

The problems I faced in the design were only slightly more difficult than the actual implementation. With having four entry widgets for each virtual disc that meant 4*4*4 = 64 entry widgets for the fstab function alone, this doesn't include the checkboxes, menu and cancel save buttons. This was a lot to manage and so lot of frames were needed to allow me to place the widgets in a sensible arrangement.

Currently the hard discs are arbitrarily named(hda,hdb,hdc,hdd), this is because I don't want to clutter up the window further with further widgets for naming the four discs. Another small feature not implemented is a button that would automatically set the second partition of the disc to a swap space. This seemed to bea very common operation and so to save users time this button would be clicked on all the entry widgets for the appropriate section would be set.

Unlike the rest of the system the HD Partition details are not parsed and set as defaults in the entry widgets. I had to have a cut off point at which to stop programming and I reached this point whilst modifying the parser to do just that. Parsing the disc partition details was proving a particularly difficult problem due to the large number of commands and varied styles of usage. I believe though that with a little more time I would have been able to implement partitioning parsing. no parsing of fstab stuff

4. Testing and Evaluation

Throughout the lifecycle of this project there has been a focus on testing and evaluation. Throughly testing this system has been necessary so that it produces correct source files. Also there has been a focus on continuous evaluation so that the system developed to suit user needs.

A set of source files developed by myself was used primarily to test the system. These test files covered a wide range of the parameters encountered in LCFG. These source files were particularly concentrated on checking that the functionality implemented in the system worked. For example files would contain all the different types of quotes /**/, /* ...*/, /* ..., carriage return, */ etc to make sure that the parser caught all possible types of comments. These test files evolved as new functionality was imlpemented into the system.

Testing was also carried out in the real LCFG environment as soon as the protoype system was ready. During the full integration tests the system performed well. Even with a drastic increase in the number of source files available the system still performed as expected with almost no perceivable delay in the loading process or whilst parsing.

4.1 Testing Results

It is not possible for the user to progress to the configuration screen if a machine or aspect has not been selected. For ease of use I have allowed the user to type in the name of the source file they wist to edit rather than selecting it from the list. However this has the unfortunate side affect that the user can enter in anything they want and then press accept and the system will try to find the file and fail. Some form of error checking is needed here Similar problems can be found with all the entry widgets in the GUI and if the system were to be expanded a method of comparing the user entered field with the entries of the appropriate array would solve this problem.

During testing it was also discovered that if a configuration file is edited then another or indeed the same one is chosen and the user chooses to edit again some of the variables used in the previous run are not completely reset and so erroneus results were found in some of the test cases. This is the fault of some of the (necessary) global variables, these variables need to be set to null when the system returns to the initial screen to solve this problem.

Some comments don't get caught. comments with no space between the open

comment token and the text will not be found and as a result will be placed in the text section of the GUI. This is not a major problem as when written out to the source file it will still be encapsulated in its comment braces. A problem arises however if an open comment token that has a space between the token and text and if we have a close comment token that has no space beside the text. In this situation the system will place everything in the comment section. The reverse can also happen but with less disasterous affects. This was a feature found during testing and can be overcome by refing the parser to search for the regualr expression within words as opposed to comparing whole words as I have done.

The system was tested with double the number od source files and behaved slightly slower in the start up phase but otherwise unchanged. This proves that the system is scalable. I tested the GUI with a source file twice as large as the largest source file (sitedefs.h) and this worked fine as well. I tried to find the maximum number of characters that text widgets can hold but couldnot find the answer. I am confident though that if it scales to hold twice the largest source file it will be suficiently future proof. The same goes for the list boxes and entry boxes but again I am confident that they will be large enough to hold whatever the future brings.

During the first test runs with the full LCFG system it was noticed that some machine files did not have a hardware platform. This was expected of some source files like printers but these were normal machines. After investigation it seems that some of the functions have been transferred over to the informatics folder and have an entry there to cover the hardware base. This means that although the hardware base slot is mandatory sometimes the hardware base will be selected through the informatics tab. An extension to the system would be to take account of this fact and it will also be necessary to follow how LCFG evolves to see if more source files will migrate to other locations.

Another test performed is what happens if the the source file read in contains a header file that no longer exists. If the included source file is displayed in an entry widget then it will be displayed as the parsed option but it will not be included in the drop down list. Otherwise this error does not intefere with the system but is obviously unsatisfactory and a check needs to be performed that compares all the parsed in header files with the list of current header files. If the included header is to be displayed in a list box the problem is more serious and none of the options are set as a result. Again some checks need to be performed but these were very much validation and verification issues that I was trying not to focus on. Not ignore because they are problems with my system but these problems can be solved with a series of validation checks and so they have been noted here in case the system is to be upgraded.

should have used a hash instead of an array but initial testing was with a small

set of files so didn't notice when big file thing might make a difference not sure

Blank entries for compulsory files?? what happens?

4.2 Evaluation of the System

Apart from those situations described in the previous section the system produced satisfactory working source files under all other testing conditions. This was the aim of the project and so I can say that the system I have developed has achieved it. The user may develop any type of source file through the GUI, many of the lesser used parameters have not been catered for explicitly butcan still be developed throught the use of the text section. Source file creation has been aided by the structure provided in the GUI and user workload has been lessened as most of the code is auto-generated by the I/O module.

The GUI itself is easy to use and straight forward although I feel that with a little more time I could have added a slightly more professional feel to it. My main complaint is that there is a lot of wasted space, this is especially apparent in the hardware and software tabs. However the size of the window is dictated by the size of the text widget from the text and comment tabs so making the window smaller is not an option. The misc tab I feel has come together well and looks quite good as does the configure partition window but these are the only two pieces I am entirely happy with on purely asthetic basis.

The help system is minimal at best and I would have liked to have had the chance to develop this further using various methods such as pop balloons and a searchable help system. I feel that development of the system should have been one of my aims earlier on in the project. By including all I have learned about LCFG over the lifespan of the project and also a more detailed account of how to work the GUI would have made the system much easier for a novice to LCFG.

The aspects side of the system I feel is also slightly a let down but I feel that there is little I can do to develop it any further. As it stands it is not a particularly useful tool and only can be used as an aid to including header files. Using a text editor I feel would have been just as quick as using the GUI as there are additional features in the editor that would speed up the process. Further investigation may provide a solution to representing header files in a GUI environment but from my studies I believe that it is possible but I am not sure that it would make anything any easier.

The parser is one of my success stories and I am quite proud of the way that the source files are parsed - quickly, and the way that the options are selected within the GUI. There are some flaws in teh parser as pointed out above but I

believe these are fixable and that if there is a need the parser module is easily extendable.

The decision to drop the DBM module in the middle of the project I maintain was a good one. Especially after full integration testing where liveness of the system was obviously not a problem. I believe that the DBM would have created more problems than it may have solved. Its inclusion would have enabled some interesting functions to have been implemented such as changing the operating system header file across all source files which had a certain hardware base, this would have been achieved by searching across the hardware base field in the dbm and and then changing the os field of all the source fles that matched. There are certainly some exciting functions that could be developed and so perhaps in the future the DBM could be added and when needed for a special function, could be updated and then carry out the opertation. I still believe that parsing 'on the fly' is a better method as keeping DBM integrity intact would be too much of a headache.

The source files produced are accurate and are all in a standard format with auto-generated quotes. I think if all the LCFG source files were of a simpilar style and structure they would be a lot easier to manipualate and understand. Of course there is still the problem with the comment tracker and so not all comments will be as useful as the auto-generated ones. I would very much have liked to have tried to implement the comment tracker, I did enjoy developing the parser - far more than I did developing the GUI side of the system. It is an certainly an interesting problem that needs some further investigation.

Appendix C shows an example source file produced using the GUI.

4.3 Lessons Learnt

This is the first project of any practical size and use that I have had the opportunity to undertake and as such it has been an invaluable learning experience. I've learned a great deal about time management, there has been no one to stand over me to make me work and in a sense there has been no deadlines other than a start and finish date. It was also good to have a chance to put into practice what we have been learnng for the last four years about the software development process. It easy to see why it is such an important subject as even with a project this size I was sometimes sidetracked and it took me a while to realise that I ha deviated from what I should be doing.

This was the first large piece of code that has been solely developed by myself, I think that if I had the chance to redo the project I think that I would have made the code much more object orientated in nature. The books that I began

learing PERL from weren't in an OO style and so my inital code begun life in this way. Then all of a sudden I was 7 weeks into the project and it was too late to change the code around. Not to say that the code is unitelligable but after four years of OO methodolodgy getting taught to us I would have liked to have had the chance to develop a large system based on that paradigm.

I also learned that things will go wrong, no matter how much time is spent designing the system. I didn't really take into account how much time would be spent revising the system, taking account new information and developing solutions to problems that I hadn't anticipated. I think that I should have allowed more time in my intial timeplans to cover for unexpected errors that were bound to appear.

I feel that the amount of HCI research that I did at the start of the project was too much. Especially as in the end I in fact had very little control over the actual appearance of the GUI. With most of the widgets grouped together by function it was merely common sense that dictated where to place the widgets in the window. Good HCI practice such as 'greying out' unavailable boxes were already known to me through years of using GUIs. Still some research was neccessary and if the project were to expand perhaps the level of research would become useful. Although I believe that by doing so much research at the start of the project it made it easier to conceptualise how I was going to turn a source file into a GUI representation.

I have also learned to use a new language and also got my first taste of GUI development. Skills which I am sure I will use in the future.

4.4 Extensions, Improvements and Documentation issues

Firstly there are security issus with the system. Who gets access to the system and how do we restrict this access. Can the GUI be modified in such a way so that we can have access levels, this would mean that depending on security access level of the user thy would only be able to carry out certain functions or edit ceratin cource files.

In a similar vein but not quite a security issue if there are multiple instances of the GUI running on different machines may cause race conditions. Which means whoever saves first won't have any of their details stored as the second save will overwrite what they had done. Some provision of mutual eclusion could be used to allow only one instance of the GUI access to a source file at a time.

There are a lot of Validation and verification issues throughout the program and

these have not been ignored but as said previously in this document it was I did not focus on them as I wanted to concentrate my efforts on improving the functionality of the system. It was also assumed that any errors in the source file will be picked up by the mkxprof compliler and as a result v and v issues did not have to be dealt with at the GUI. However to make the system more robust and helpful data entries should be validated, numerical ranges should be checked to make sure they are sensible etc.

Improvements to the system have for the most part already been stated in the previous sections but in a nutshell: the layout and appearance of the GUI could do with some more work to make it look a little more professional. The text and comment tabs could be upgraded so that they appear much like a word processor so that development of large portions of source file text is made quicker[1].

Documentation issues, there is a user guide (see appendix D) which will need to be kept up to date as new functions are added. It is also important that with any functions added there is appropriate program code documentation and comments throughout. If there is to be more than one person working on a software project over its lifetime it is crucial that everyone can understand everyone else's code both so it can maintained and added to.

I would have liked to have developed error checking on the inputs but there really wasn't time. All of the inputs should be validated but it was a decision made early on in the project to focus on functionality rather than validation. However there is definetly scope for implementnig a lot of error checking within the system. The reason why this problem isn't high on the priority list is that the mkxprof compiler checks the source file first so in a way there is some indirect error checking in the system. Error messages will get fed back to the user at some stage but it is an inconvienance.

Another additional feature that I would liked to have developed is a graphical way of allocating disc space in the partitioning sections. Using a pie chart with a sliding scale it would be possible to see the partitions. This was quite low on the priority list though as it is hoped that the user has a clear idea of how they want the hard disc to be partitioned and also beacause it would have meant importing in modules from CPAN.

There is scope for the project to be extended from the framework that I have developed. Many of the extensions and improvements have been discussed in detail throughout this document.

[1]NB: there are already functions such as cut and paste already included as part of the text widget, a menu for these functions can be activated by right clicking the mouse on the widget

5. Conclusions

The basic aims have been achieved and also a number of the advanced ones. A framework system now exists that can allow novice userrs to develop LCFG source files. More time is needed to make the system completely sound (I have pointed out the flaws in the system in the preceeding chapters) and also to add validation to all user input.

I believe the project has gone well and am happy with what has been produced, the pace has been slower that I had predicted and so less functionality has been implemented than I would have liked. Namely the parsing of the disc partition commands and also modifying the parser to allow some form of comment tracking. More work is needed on the complicated functions before the GUI can start to be used by expert users as the functionality is not there to support the types of commands an user of this type would need.

the system has performed consistently well in testing. From the feedback I believe that (with a little more work on the functionality and validation) that the GUI would be able to compete against emacs as the top way to configure LCFG source files.

Bibliography

[1] Title: DICE Architecture
Author : Simon Wilkinson

http://www.dice.informatics.ed.ac.uk/development/doc/arch_stage1.html

[2] Title: Towards a High Level Machine Configuration System
Author : Paul Anderson

http://www.lcfg.org/doc/LISA8_Paper.pdf

[3] Title: Large Scale Linux Configuration with LCFG
Author : Paul Anderson, Alastair Scobie

http://www.lcfg.org/doc/ALS2000.pdf

[4] Title: Technologies for Large-Scale Configuration Management
Author : Paul Anderson, George Beckett, Kostas Kavoussanakis, Guillaume Mecheneau , Peter Toft

www.epcc.ed.ac.uk/gridweaver/WP1/report1.pdf

[5] Title: Human-Computer Interaction
Author : Dix, Finlay, Abowd, Beale

[6] Title: LCFG: The Next Generation
Author : Paul Anderson, Alastair Scobie

http://www.lcfg.org/doc/ukuug2002.pdf

[7] Title: LCFG Evaluation

https://wwwlistbox.cern.ch/earchive/hep-proj-grid-fabric-resource/ doc00000.doc

[8] Title: Webopedia

http://www.webopedia.com

[9] Title: Systems Analysis and Design
Author : Kendall, Kensall
ISBN: 0-13-042365-3

[10] Title:Mastering Perl/Tk
Author :Stephen O. Lidie, Nancy Walsh

[11] Title: Learning Perl
Author : Randal L. Schwartz, Tom Phoenix

[12] Title:Programming Perl
Author :Larry Wall, Tom Christiansen, Jon Orwant

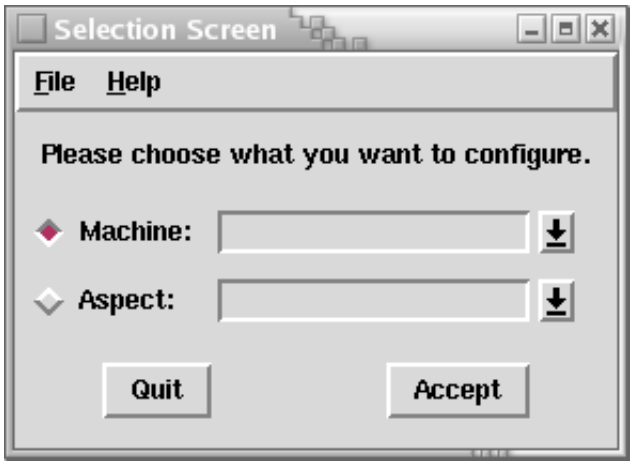

Figure 5.1: Initial Screen

Appendix A

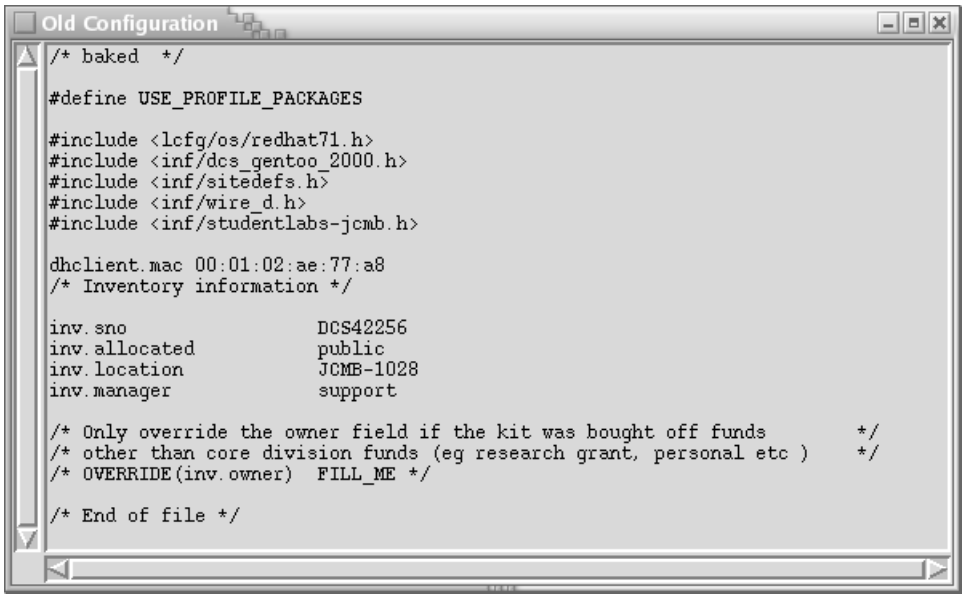

Figure 5.2: Old Source File Configuration

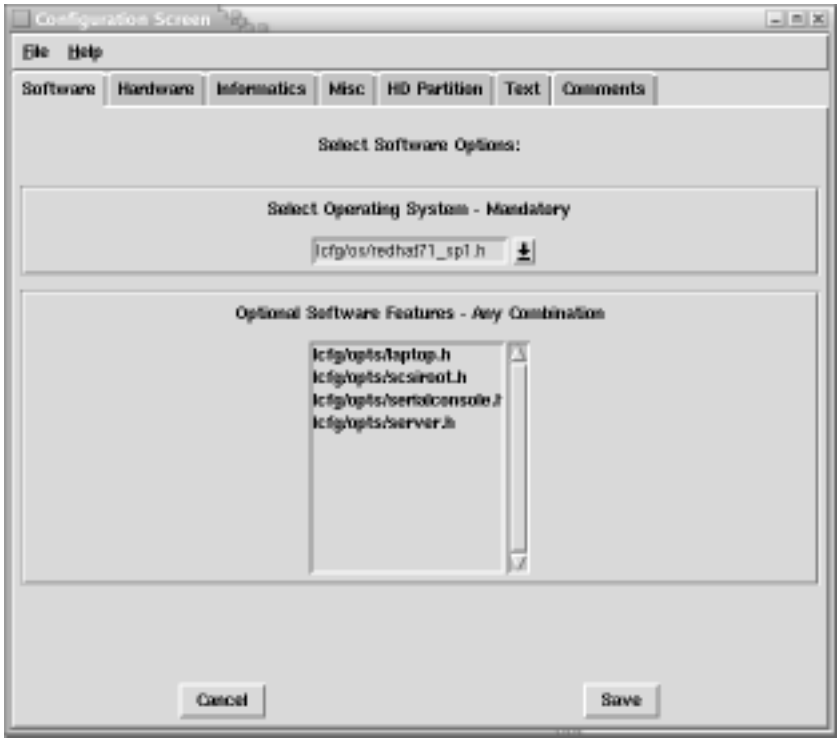

Figure 5.3: Hardware Configuration

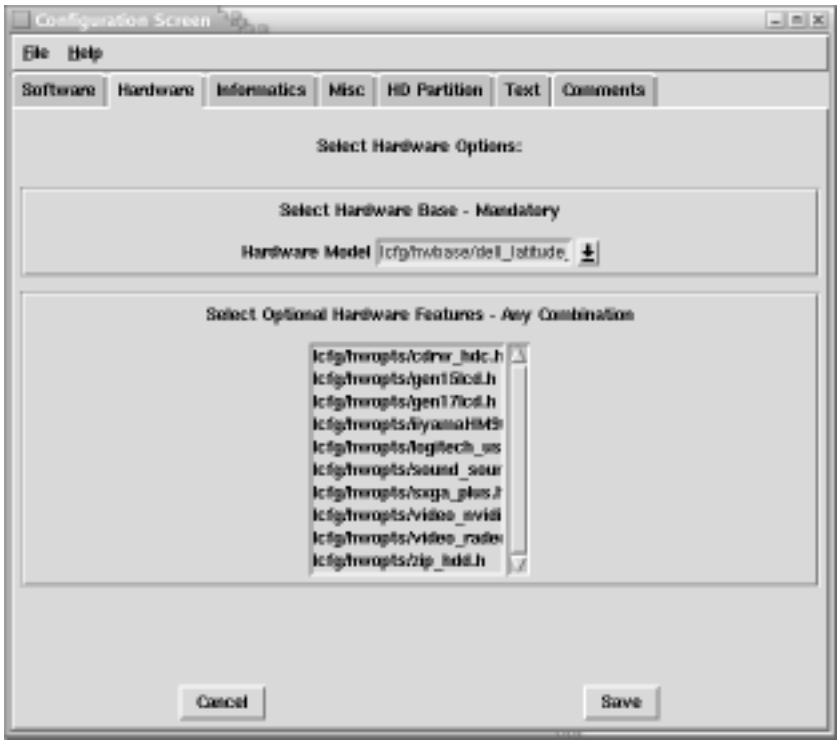

Figure 5.4: Hardware Configuration

Figure 5.5: Inf Configuration

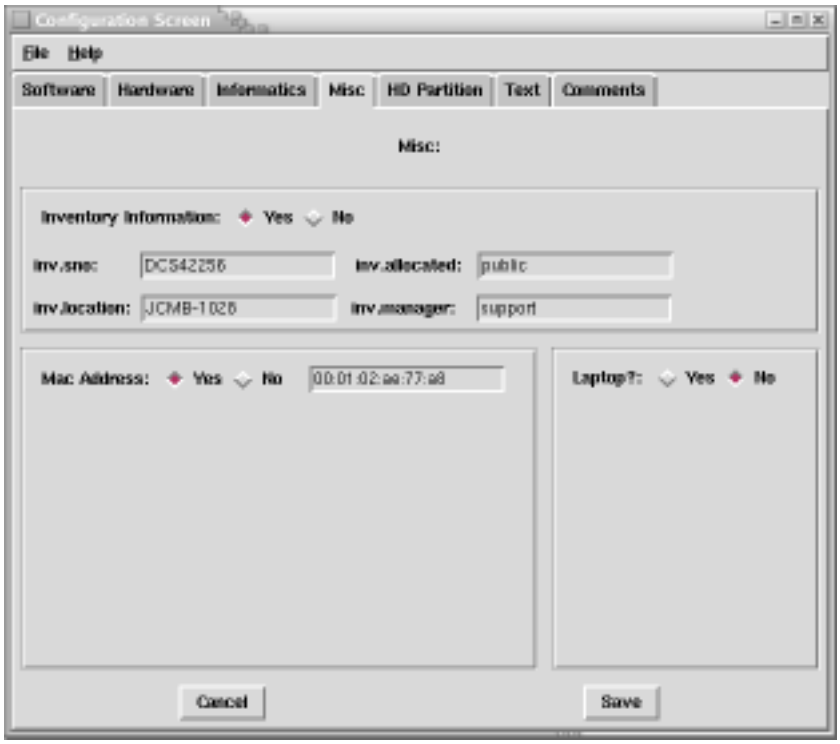

Figure 5.6: Miscellaneous Configuration

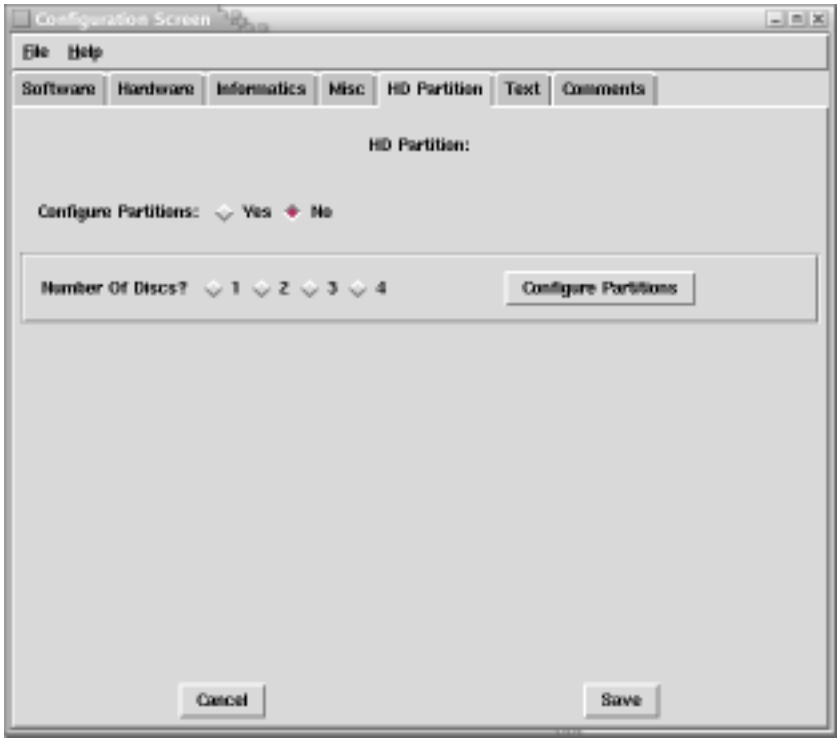

Figure 5.7: fstab Configuration

Figure 5.8: Additional Text Configuration

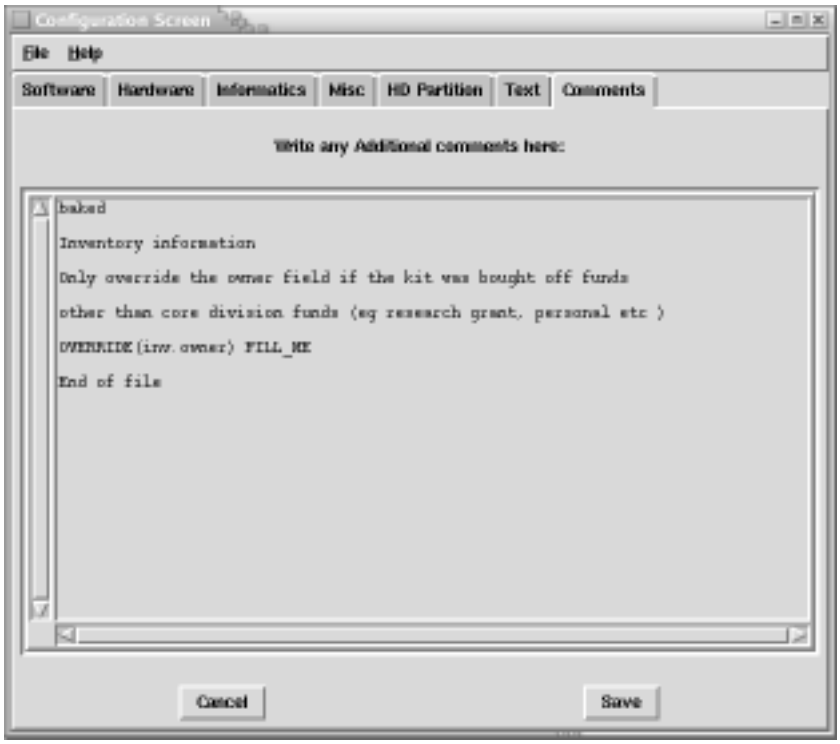

Figure 5.9: Comment Configuration

Appendix B

An example of a machine Source file:

```
#include <lcfg/os/redhat71.h>
#include <lcfg/hwbase/dell_optiplex_gx240.h>
#include <inf/sitedefs.h>
#include <inf/wire_c.h>
#include <inf/office-jcmb.h>

!auth.users mADD(+djr)

dhclient.mac 00:06:5b:be:e9:9c

/* Partition the Disk */
!fstab.disks mSET(hda)
!fstab.dopartition_hda mSET(no)
!fstab.partitions_hda mSET(hda1 hda2 hda3)

/* Root partition */
```

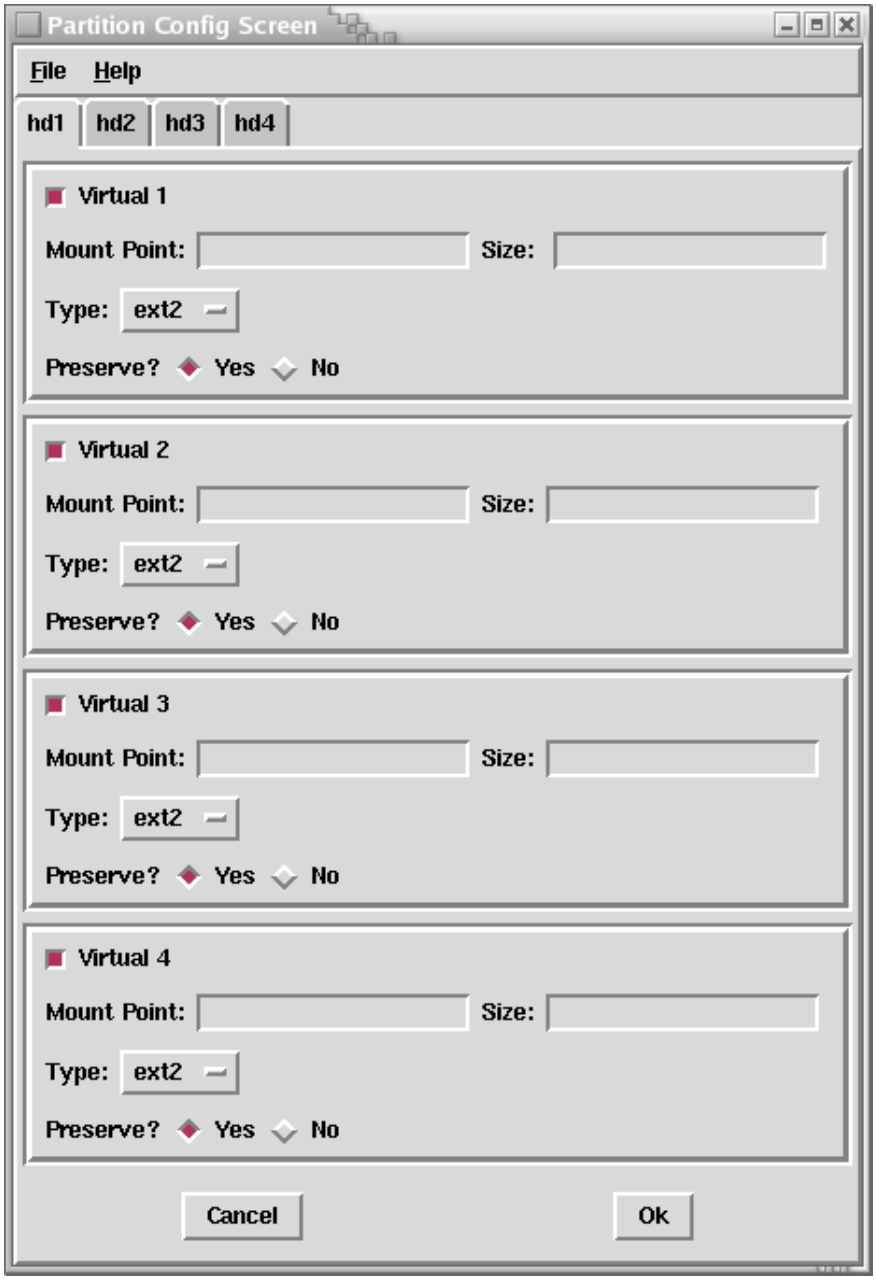

Figure 5.10: Disc Partition Configuration

```
!fstab.mpt_hda1          mSET(/)
!fstab.size_hda1         mSET(10000)
!fstab.type_hda1         mSET(ext2)
!fstab.preserve_hda1     mSET(no)

/* Swap partition */
!fstab.size_hda2         mSET(512)
!fstab.type_hda2         mSET(swap)
!fstab.preserve_hda2     mSET(no)

/* Personal home space */
!fstab.mpt_hda3          mSET(/disk/home/asparagus)
!fstab.size_hda3         mSET(free)
!fstab.type_hda3         mSET(ext2)
!fstab.preserve_hda3     mSET(yes)


/* Inventory information */

inv.sno 5W5PF0J
inv.allocated djr
inv.location JCMB-1510
inv.manager  support

/* End of File */
```

An example of a header source file:

```
/* Site specific defaults for inf.ed.ac.uk */
/* DCS [Datalink] machines bought in summer 2000 (purchase order a450162) */

/* based on dcs.ed.ac.uk RH62 header files
   dcs_gentoo.h, dcs_gentoo2.h, dcs_gentoo_hdc.h */

xfree.video atiragepro8
xfree.monitor idekpro410
xfree.displays 16bit 8bit 24bit
xfree.mouse  pilot
xfree.keyboard pc104
/* old mutation syntax, replaced below
OVERRIDE(xfree.modes_8bit)     1152x864 1024x768
OVERRIDE(xfree.modes_16bit)    1152x864 1024x768
OVERRIDE(xfree.modes_24bit)    1152x864 1024x768
```

```
*/
!xfree.modes_8bit     mSET(1152x864 1024x768)
!xfree.modes_16bit    mSET(1152x864 1024x768)
!xfree.modes_24bit    mSET(1152x864 1024x768)

/* old mutation syntax, replaced below
EXTRA(hardware.modlist) ether usbuhci     */
!hardware.modlist   mADD(ether usbuhci)

hardware.mod_ether      alias eth0 3c59x
hardware.mod_usbuhci    alias usb-controller usb-uhci

inv.model       DCS ATX Celeron

/* Suspend all clock-related differences as an experiment -
   maybe it's better under 7.1.  Chris 6 Aug 2002 */
#ifdef NOTUSINGTHISJUSTNOW

/* These machines seem to have very wonky clocks - they sometimes
   leap ahead by 25 or 27 hours.  So we correct them every hour with
   "ntpdate", and turn off xntpd to stop it getting upset about this
   (and anyway, ntpdate refuses to perform when xntpd is running). */

DELETE(boot.services,lcfg_ntp)
EXTRA(cron.additions) ntpdate
cron.add_ntpdate 17 * * * * /usr/sbin/ntpdate -s ntp1 ntp2
cron.owner_ntpdate root

/* ntpdate falls over if the clock is too far out!
   So first we set the clock approximately with rdate, then ntpdate
   comes in a minute afterwards to correct it more exactly.

   Good grief. */

EXTRA(cron.additions) rdate
cron.add_rdate 16 * * * * /usr/bin/rdate -s sligga.dcs.ed.ac.uk
cron.owner_rdate root

#endif
```

An example of the LCFG machine template file:

```
%
```

```
% Template linux machine profile
%
% * the ordering of header files categories is critical
% * you should provide values for all resources that have
%   the dummy value FILL_ME
%
%
% DELETE from this line upwards
%
% *** Please delete lines that you're not using ***
%
/* fill in hostname here  */

/* Operating system - mandatory */
#include <lcfg/os/redhat71.h>

/* Optional software features - choose any combination of */
/* #include <lcfg/opts/laptop.h>*/
/* #include <lcfg/opts/scsiroot.h>*/
/* #include <lcfg/opts/serialconsole.h>*/
/* #include <lcfg/opts/server.h>*/

/* Hardware model - mandatory - choose one */
/* #include <lcfg/hwbase/dell_latitude_c640.h>  */
/* #include <lcfg/hwbase/dell_optiplex_gxa.h>   */
/* #include <lcfg/hwbase/dell_optiplex_gx1.h>   */
/* #include <lcfg/hwbase/dell_optiplex_gx110.h> */
/* #include <lcfg/hwbase/dell_optiplex_gx150.h> */
/* #include <lcfg/hwbase/dell_optiplex_gx240.h> */
/* #include <lcfg/hwbase/dell_optiplex_gx260.h> */
/* #include <lcfg/hwbase/dell_poweredge_1400.h> */
/* #include <lcfg/hwbase/dell_poweredge_2500.h> */
/* #include <lcfg/hwbase/dell_poweredge_4600.h> */
/* #include <lcfg/hwbase/dell_ws340.h>          */
/* #include <lcfg/hwbase/dell_ws350.h>          */
/* #include <lcfg/hwbase/dell_ws530.h>          */
/* #include <lcfg/hwbase/hp_6000.h>             */

/* Optional hardware features - choose any combination of */
/* #include <lcfg/hwopts/cdrw_hdc.h>*/
/* #include <lcfg/hwopts/gen15lcd.h>*/
/* #include <lcfg/hwopts/gen17lcd.h>*/
/* #include <lcfg/hwopts/iiyamaHM903DT.h>*/
```

```
/* #include <lcfg/hwopts/logitech_usb_wheel.h>*/
/* #include <lcfg/hwopts/sound_soundblaster.h>*/
/* #include <lcfg/hwopts/sxga_plus.h>*/
/* #include <lcfg/hwopts/video_nvidia.h>*/
/* #include <lcfg/hwopts/video_radeon.h>*/
/* #include <lcfg/hwopts/zip_hdd.h>*/

/* Mandatory site specific defaults */
#include <inf/sitedefs.h>

/* Mandatory site wire - choose one */
/* #include <inf/wire_at1.h> */
/* #include <inf/wire_c.h> */
/* #include <inf/wire_d.h> */
/* #include <inf/wire_g.h> */
/* #include <inf/wire_m.h> */
/* #include <inf/wire_w.h> */

/* Optional site features - choose any combination of */
/* #include <inf/bp_machines.h>*/
/* #include <inf/console_server.h>*/
/* #include <inf/dcs_gentoo_2000.h>*/
/* #include <inf/dhcpd_server.h>*/
/* #include <inf/fhpkgs.h>*/
/* #include <inf/grub.h>*/
/* #include <inf/ipfilter.h>*/
/* #include <inf/iptables.h>*/
/* #include <inf/kdc.h>*/
/* #include <inf/netinf.h>*/
/* #include <inf/ntp.h>*/
/* #include <inf/officedesktop.h>*/
/* #include <inf/printer.h>*/
/* #include <inf/pxe_server.h>*/
/* #include <inf/studentlabs-at.h>*/
/* #include <inf/studentlabs-jcmb.h>*/
/* #include <inf/studentlabs.h>*/
/* #include <inf/studentlabs_bp.h>*/
/* #include <inf/vmwarews.h>*/
/* #include <inf/wireless.h>*/

/* Inventory information */

inv.sno  FILL_ME
```

```
inv.allocated FILL_ME
inv.location FILL_ME
inv.manager  FILL_ME

/* Only override the owner field if the kit was bought off funds */
/* other than core division funds (eg research grant, personal etc )
*/
/* OVERRIDE(inv.owner) FILL_ME */

/* End of file */
```

Appendix C

Example of the output source file produced by the system

```
/* -+- source_files/baked */

/*
baked

Inventory information

Only override the owner field if the kit was bought off funds

other than core division funds (eg research grant, personal etc )

OVERRIDE(inv.owner) FILL_ME

End of file


*/

/* -+- Mandatory Hardware Model */
#include <lcfg/hwbase/hp_6000.h>

/* -+- Mandatory Operating System */
#include <lcfg/os/redhat71.h>

/* -+- Optional Software Extras */
#include <lcfg/opts/serialconsole.h>
#include <lcfg/opts/server.h>

/* -+- Optional Hardware Extras */
#include <lcfg/hwopts/sound_soundblaster.h>
#include <lcfg/hwopts/video_radeon.h>

/* -+- Mandatory Site Specific Details */
#include <inf/sitedefs.h>

/* -+- Mandatory Site Wire */
#include <inf/wire_d.h>
```

```
/* -+- Optional Site Features */
#include <inf/pxe_server.h>
#include <inf/studentlabs-jcmb.h>
#include <inf/testing.h>

/* -+- mac Address */
dhclient.mac 00:01:02:ae:77:a8

#define USE_PROFILE_PACKAGES


/* -+- Inventory Information */
inv.sno         DCS42256
inv.allocated   public
inv.location    JCMB-1028
inv.manager     support

/* -+- End of File */
```

Appendix D

User Guide

This guide is to be used in conjuction with version 1.0 of the LCFG GUI front end designed and written by Craig Devlin 25/05/03.

Type perl LCFGinterface at the console to begin.

Source File Selection

You will see a progress bar please be patient whilst the program examines the LCFG source file directories. From here you will be presented with a window from which you can choose to edit a source file. If you are editing a machine file click the machine radio button and select or type the name of the file you wish to edit. For aspect files click on the aspect radio button and proceed as above. When you have selected the source file click the accept button.

Source File Configuration

Once accept has been pressed you will be presented with two new windows. One is the current source file in a plain text format this is for reference purposes whilst editing. The second window is where you may edit the source file itself.

There are two separate interfaces available, one for machine source file configuration and one for aspect source file configuration. I will describe the aspect interface first.

The aspect interface has three tab headings. The first headers contains a list of all the header files available under the LCFG environment. Those that are included in the current source file will be highlighted. To include a header file in the source file click on the entry in the list. Similarly to remove a header file from the source file click on a highlighted entry.

The second tab is where configuration details other than include statements may be entered. The contents of the text entry area will include all the details from the current source file and may be edited, deleted and added to as fits the situation.

The third and final tab in the interface is for the comments. You may add comments into this section and they will be output in comment format in the resultant source file. NB: all comments will be grouped together into a single large comment displayed at the top of the source file. For comments relating to

particular areas of source code information the details may be entered into the text tab with the normal comment braces (/* */) around the text.

The machine interface works in a similar method to that of the aspect configuration above. You will be presented with a window containing seven tabs. The first, second third and fourth tabs are edited by selecting entries for the drop down menus and lists as required. The sixth and seventh tabs are edited as for tabs two and three above. Tab five: HD Partition is used for configuring the disc partition of LCFG machines. To edit the partitions check the yes radio button on the screen and then choose how many physical hard discs are available then click configure.

A new window will open with a tab for each physical hard disc. Within each tab there are four section each of which represent a virtual partition. Each of these may be edited as the user needs. NB the checkboxes at the side indicate whether there is a virtual partition, it is not neccessary therefore to use all four partitions. Once editiing is complete click on the accept button to take you back to the main configuration window. Alternativly click the cancel button to cancel the partition configuration.

To save your work click on the save button, you will be prompted if you wish to save, click yes to continue which will save the file. You will now back at the original source file selection screen, you may edit another file or click the exit button to exit.

Thank you for using GUI-LCFG.