

USING THE UNIX™ SHELL TO INTEGRATE A  
MANAGEMENT MODEL WITH A GIS

By

FENGXIA MA

Bachelor of Science

Wuhan Technical University of

Surveying and Mapping

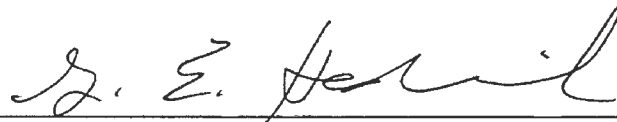
Wuhan, P.R. China

1985

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
December, 1993

USING THE UNIX™ SHELL TO INTEGRATE A  
MANAGEMENT MODEL WITH A GIS

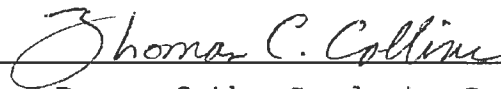
Thesis Approved:



Thesis Advisor

H. Lu





Dean of the Graduate College

## ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Dr. George Hedrick, my major advisor, for his encouragement and advice throughout my graduate program.

Many thanks also go to Dr. David Nofziger for giving me the opportunity to practice my knowledge, for providing the knowledgeable instructions I needed. Without his help and support, this project would not have been completed so well.

Thanks are also extended to Dr. Huizhu Lu for serving on my committee and providing helpful advice and instructions during my graduate study.

A very special thanks goes to the GIS specialist Mr. Mark Gregery at Agronomy Department of Oklahoma State University for patiently guiding me in learning and using GRASS. Only with his help, this project can be finished so smoothly.

My deepest gratitude goes to my parents for their caring and support.

Last, but not the least, great appreciation goes to my husband Gang Huang for his consistent support, encouragement and understanding. Without him, none of this would have been possible.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION . . . . .	1
II. LITERATURE REVIEW . . . . .	5
III. ENVIRONMENTAL BACKGROUND . . . . .	10
Introduction . . . . .	10
Basic UNIX™ Background . . . . .	10
UNIX™ Design Philosophy and Its Style . . . . .	10
The Components of UNIX™ . . . . .	11
Unique Features of UNIX™ . . . . .	11
Shell: The Enhancement of UNIX™ . . . . .	14
GRASS: A Powerful GIS Package . . . . .	16
CMLS: A Simulation Tool for Managing Agricultural Chemicals . . . . .	17
IV. DESIGN AND DEVELOPMENT OF GO & DRAW . . . . .	22
Introduction . . . . .	22
Functionality . . . . .	22
User Interface . . . . .	25
System Description . . . . .	33
Implementation Details . . . . .	38
V. DESIGNING GUIDELINES . . . . .	41
Reliability . . . . .	41
Maintainability . . . . .	42
Modularity . . . . .	42
Consistency . . . . .	43
Efficiency . . . . .	43
Change Directory . . . . .	43
Reduce Temporary Files . . . . .	44
Using Time To Measure the Speed . . . . .	44
Documentation . . . . .	45
VI. SUMMARY, CONCLUSIONS AND FUTURE WORK . . . . .	46
Summary and Conclusions . . . . .	46
Future Work Suggestions . . . . .	46
BIBLIOGRAPHY . . . . .	49

Chapter	Page
APPENDIXES . . . . .	51
APPENDIX A - GRASS COMMANDS USED IN THE MAP DRAWING TOOL . . . . .	52
APPENDIX B - PSEUDO-CODE OF THE PROCEDURES IN THE MAP DRAWING TOOL . . . . .	54
APPENDIX C - INSTALLATION INSTRUCTIONS OF THE MAP DRAWING TOOL . . . . .	74

## LIST OF FIGURES

Figure	Page
1. Components of UNIX™ . . . . .	12
2. CMLS Input Format . . . . .	20
3. CMLS Output Format . . . . .	21
4. Data Entered From the Command Line . . . . .	24
5. Dialogue Box 1 . . . . .	28
6. Dialogue Box 2 . . . . .	28
7. Dialogue Box 3 . . . . .	29
8. Data Entered in Three Forms . . . . .	29
9. Dialogue Box 4 . . . . .	30
10. Dialogue Box 5 . . . . .	31
11. Dialogue Box 6 . . . . .	31
12. Dialogue Box 7 . . . . .	32
13. Dialogue Box 8 . . . . .	32
14. Dialogue Box 9 . . . . .	32
15. Execution Sequence of the Map Drawing Tool . . . . .	36
16. Process CMLS . . . . .	37

## CHAPTER I

### INTRODUCTION

The shell scripts, *go & draw*, a map drawing tool developed as part of this thesis, automates the process of producing maps using the output of a simulation model, Chemical Movement in Layered Soils (CMLS).

CMLS is a simulation tool used in managing agricultural chemicals. This model is designed to use soil, chemical and weather parameters to evaluate the impact of pesticides on large areas.

Many government agencies, such as the Oklahoma Department of Agriculture, and scientists interface CMLS model with a Geographical Information System (GIS) package to produce maps from the CMLS output. The maps produced can be used in helping to evaluate the risk of groundwater contamination for specific soil-pesticide-water management systems. As a result, the high risk management systems are brought to attention immediately.

Commercially available GIS packages usually are powerful but expensive. The basic requirements for GIS users are relatively high. Users must have knowledge of the GIS as well as of the computer, including input, output devices, and the operating system.

The Geographic Resources Analysis Support System (GRASS), developed at the U.S. Army Construction Engineering Research Laboratory (USACERL) is selected as the GIS package to interface with CMLS model in this project, because it is a public domain package. GRASS was developed using federal money and distributed without charge. This GIS package includes a graphics production system with powerful map production capabilities. The package is written in C and is UNIX™ oriented. Like the other GIS packages, however, GRASS requires its users to have significant knowledge of not only the GIS, but also of computers, including the UNIX™ system as well as the input/output devices, the databases being analyzed, the areas under analysis, and the requirements of the analysis [22]. The ability to produce thematic maps is only a small part of GRASS's powerful functionality, but users still must go through a learning curve to accomplish this job. It is impractical to ask a casual user of GRASS to take a great deal of time to go through the entire learning process for occasional use.

UNIX™, the system that GRASS is designed on, is a portable multi-user, time-sharing operating system and can be used on a variety of platforms. It is designed for programmers, and it is distributed with a wealth of development tools. These development tools are the basis for creating more complicated, powerful and flexible utilities. Many UNIX™ programmers consider their work as the creation and use of tools. They find UNIX™ a helpful, productive,



and pleasant environment in which to work. To a casual user, however, UNIX™ has a terse, unfriendly, unforgiving, inconsistent user interface [20]. It is hard for people who do not use the system very often to adjust to the environment.

The map drawing tool designed and developed in this project interfaces CMLS model with GRASS via UNIX™ shell programming. It allows the CMLS users to produce maps with minimal effort. It is not required for the user of `go & draw` to have much knowledge in either GRASS or UNIX™. Instead of going to the workstation terminals, users can even get their maps from their desktop PC providing it is connected to the network and running the X-window interface. The map drawing tool provides the CMLS user with an effective, productive working tool.

The map drawing tool is written for Sun Microsystems' workstations using the C shell for use with GRASS version 4.0. Due to the restrictions of current available data bases, the project is tested only on Oklahoma state data. When the data bases from other states become available, the project should be enhanced easily to work on all the data bases of the country.

Chapter II reviews the shell programming research and practical work that have been done in the past. Chapter III presents the basic background involved in doing this project. This includes the UNIX™ operating system, the GRASS package and the CMLS model. Chapter IV describes the

project in detail, including its features, functionalities and its implementation details. The designing guidelines of the project are introduced in Chapter V. Finally the summary, conclusions and the future work suggestions of the project are given in Chapter VI.

Three appendixes are included for reference. In appendix A, the GRASS commands used in this project and a short explanation for each command are listed. Appendix B includes the pseudo-code of the project. The installation instructions are given in appendix C.

Throughout the thesis, "He" is used both in a masculine and feminine sense, i.e., as a generic personal pronoun.

## CHAPTER II

### LITERATURE REVIEW

Much literature about shell programming deals with using shell utilities to customize computing environments and using shell utilities to manage software or to build small, frequently used application tools. In most cases, the UNIX™ shell utility is the major foundation. The proposed shell scripts **go & draw** build a bridge between a simulation model and a Geographic Information System by means of shell programming. In this project, functions provided by the Geographic Information System GRASS is the working foundation. The UNIX™ shell utility is used as the bond to link all the proper functions logically.

With the help of **go & draw** the CMLS users are able to produce useful maps at greatly reduced cost and effort due to less programming involved. Therefore the software productivity is increased notably.

Shell utilities have long been treated as a programming tool. Kernighan from Bell Laboratories [10] described the history of shell programming as:

The use of the shell as a programming tool has evolved over the years. It began a decade ago with file redirection, but the real revolution occurred with the invention of pipes... . The programmability of the shell itself has increased,

reaching a plateau with the 7th edition shell written by S. R. Bourne.

Bourne [3] discussed shell programming in his paper, The UNIX™ Shell and stated that shell procedures are text files which do not need to be compiled. Therefore they are easy to maintain. Debugging is easier because of the capability of trying part of a procedure at a time.

UNIX™ utilities are written in the manner that each command is general-purpose and independent. It is possible for users to finish their jobs by combining these tools with minimal time and effort. In other words, many programming tasks can be accomplished by writing shell procedures instead of writing compiled programs. By organizing shell utilities, many jobs can be done much easier than by using compiled programming languages such as C or C++. The procedures written in the shell utilities usually are shorter as well.

Because the shell provides powerful tools for manipulating documents and building programs, it has been widely used in the related areas.

The most common case of shell programming is that UNIX™ programmers use shell utilities to tailor their log-in environments in order to fit their own needs. By changing the default prompt, users put their own identities on their log-in environments. By changing the default search path, users get more efficient working environments. Fiedler [7] described how users can reduce their user support time by

setting variables in the system.

An example given by Kernighan and Mashey [11] displayed the application of shell programming in supporting programming projects in Bell laboratories. In this application, shell procedures are widely used to assist in managing a management decision support system. Shell procedures provide a compact, integrated and highly automated working environment.

`Run`, a 400 line Bourne shell script, used to control the BYTE UNIX™ benchmark, is an example of shell tool usage as an application program. There are three different stages for each benchmark test. Setting up parameters, timing the execution of the tests and calculating, formatting operations. After determining the parameters for the benchmark test, `Run` sends a formatted description to the output file and then invokes the specific benchmark test with the help of the UNIX command `time`. The output of `time` and other commands is saved in a raw data file. After the completion of a set of tests, `Run` invokes a cleanup script to do statistical calculations on the raw data by means of `awk`. The benchmark fairly reflects the strengths and weaknesses of all the systems for which `Run` is expected to be used [19].

Facilities such as shell utilities and the pipe mechanism, provided by UNIX™ offer great possibilities for software reusability. In UNIX™, reuse of software is at the program level, that is, combining programs using the command

interpreter (shell). Thus, each user-invoked program becomes a building block analogous to a function in a programming language. Kernighan [10] defined software reusability as:

Previously written software can be used for a new purpose, or to avoid writing more software.

The necessity for reusing software becomes obvious as Horowitz and Munson [8] illustrated with their research. In 1980 the U.S. Department of Defense (DOD) spent more than \$3 billion on software and this cost is increasing at a rapid rate. They discussed the potential for increasing software productivity by addressing the concept of software reusability and concluded that cost, the bottle-neck of software development is because the entire software system is built up "from scratch". Applying the idea of software reusability would increase the software productivity.

By applying the concept of software reusability, one distinct benefit is that the cost is reduced, another is that less sophisticated users can gain access to many complicated packages or algorithms without knowledge of the internal workings.

A simple example for reusable software given by Kernighan [10] is the program `cal` (for calendar). This program can print the calendar for any specified year and month. The original program is written in C and is about 200 lines with messy computation involved. The program requests all the necessary arguments for month and year to be

supplied and month has to be in the form of number. If these restrictive conditions are not satisfied the program will not work. By using a simple shell program with the help of UNIX™ utility `date` and the logical structure `case`, the `cal` program becomes rather flexible without changing any internal C source code.

Combining CMLS with GRASS using the UNIX™ shell, CMLS users can produce maps from the CMLS output. The CMLS users get a more powerful tool at a greatly reduced cost because most utilities used already exist. No literature that describes using the UNIX™ shell to integrate a simulation model with the GRASS package has been gathered. It is really worthwhile to have this application software developed.

## CHAPTER III

### ENVIRONMENTAL BACKGROUND

#### Introduction

UNIX™, the work-horse of the project, along with GRASS and CMLS, the two software packages deeply involved in the development of **go & draw** are discussed in this chapter.

#### Basic UNIX™ Background

#### UNIX™ Design Philosophy and Its Style

UNIX™ was first developed by Ken Thomson in PDP-7 assembly language at the Bell Laboratories. Its original name was UNICS for UNiplexed Information and Computing Service and was changed to UNIX™ later.

After Ken Thompson and Dennis Ritchie rewrote the system in C, the system became easier to understand, maintain and easier to transport to other platforms. From the beginning the primary goal of writing this operating system was to create an effective environment for programmers to write and use programs. This determined the style of the UNIX™ operating system, that is, simplicity, generality and intelligibility. This environment cultivated the unique UNIX™ software style. As Mclroy, Pinson and Teagure outlined [13]:



A distinctive software style has grown upon this base [simplicity, generality, intelligibility]. UNIX™ software works smoothly together, elaborate computing tasks are typically composed from loosely coupled small parts, often software tools taken off the shelf.

### The Components of UNIX™

The Figure from page 12 of Sobell's book clearly shows the components of the UNIX™ operating system (see Figure 3.1), where software is built on hardware in different layers. Among the software layers, the inner-most part is the UNIX™ kernel, which contains the interface to the hardware and provides the most primitive of operating system features. It is a relatively small program. From the core, the second software layer is the system shell, which is a command interpreter and acts as an interface between the UNIX™ system and the user (programmer). There are many shell commands included within the shell and they are the basis for programmers to composite more powerful and complicated tools. The third layer consists of different application programs, such as compilers, editors, word processors, mail tools, and so on.

### Unique Features of UNIX™

UNIX™ has many unique features. Only the features closely related to this project are discussed here.

Simplified I/O. The reading and writing of files in UNIX™ are fairly simple. All the devices and files are

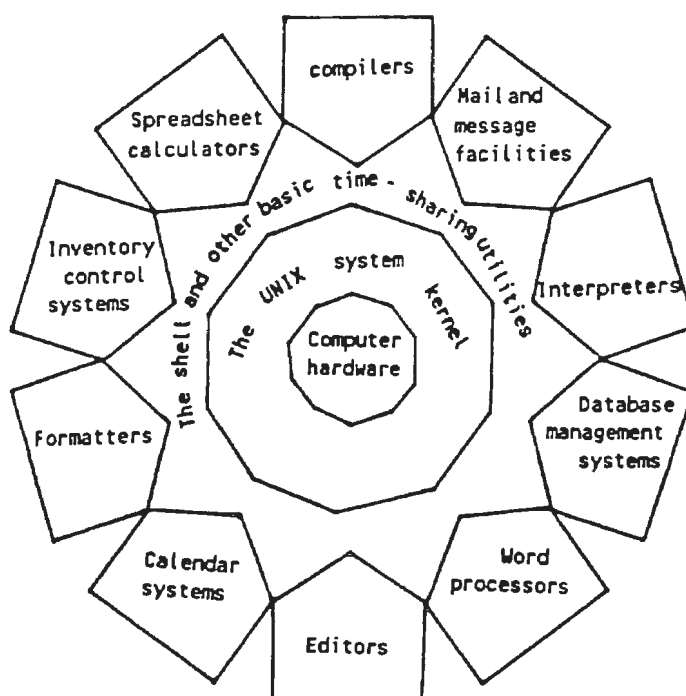


Figure 3.1. Components of UNIX™  
[source from 20]

treated in the same way. Input and output can be redirected to and from a disk file instead of the console by using the file redirection symbols (>, <). Files are stored as a stream of characters without any specific format. Multiple files can be appended together to form a new file by using the append symbols (>>, <<).

Pipe Usage. Both pipes and filters are allowed on the command line in UNIX™. A pipe is a construct that sends the output of one command to another command as input. A filter is another construction supported by UNIX™. It processes a stream of input data and yields a stream of output data. It is always used between two pipes. By combining commands using pipes and filters, a new functional utility is created. A UNIX™ programmer can use UNIX™ commands as flexibly as he uses his natural language. In a natural language, words can either have their own meaning or be combined together to form a new meaning. The commands are the vocabulary for the programmer to create more powerful tools.

There are many UNIX™ utility programs which are also called commands. The functions performed by these commands are required universally, such as creating, displaying, or moving files, creating, invoking, sleeping or terminating processes, and editing or formatting texts, etc.

Since UNIX™ was developed originally for programmers to manage their own projects, it is an ideal software development environment. The rich set of utilities provide

the software developers with powerful tools.

### Shell: the Enhancement of UNIX™

There are three major UNIX™ shells in existence today: The Bourne shell, the C shell and the Korn shell. The finished project `go & draw` uses the C shell.

The UNIX™ shell can be used as a programming language. It is an interface between the operating system and the end user. When a command is issued, the shell interprets the command and calls the related program. The UNIX™ shell is the key to increase the productivity and functionality of the UNIX™ system. It supports many high-level language constructs, such as variables, flow control structures, parameter passing or subroutine calls and interrupt handling. These capabilities are the primitives used to build more complicated programs. By composing shell commands, many tasks can be accomplished easier than by coding in other programming languages; i.e., C or C++. An example, given by Kernighan [10], indicates that the `news` program in their system, which is used to keep the user informed of current events, is 360 lines long when written in C but less than 20 lines in its shell version. Input and output redirection are also supported by UNIX™ shell, but in a slightly different way; that is, with an ampersand at the end of the redirection sign. The UNIX™ shell can customize the environment in which the command runs. Grouped commands are usually put into a file to be executed, and the

file is called a shell script. To execute the shell script, changing the file mode to executable is necessary. Once a shell script has been changed to executable, there is no difference between invoking a shell script and a compiled program. Shell scripts are powerful tools for invoking and organizing UNIX™ commands.

Bourne Shell. The Bourne shell is the most commonly used UNIX shell. It almost always comes with the UNIX™ system. It was written by Steven R. Bourne in about 1975. It has the smallest size and the best efficiency among the three shells. It allows exception handling uniquely by using the **trap** command. It has more versatile input and output redirection, allowing standard input and output to be redirected into and out of the whole control structure. It supports both local and global variables and offers **if-then-else**, **case**, **for**, **while**, and **until** control structures. The Bourne shell does not support direct expression evaluation. Expression evaluation can be performed only with the UNIX™ utilities **test** and **expr**. This can slow the process slightly. The Bourne shell does not have the advanced interactive job control features as the other two shells do; that is, when the job starts, it remains in the foreground or background. Status changes are not allowed in Bourne shell.

C Shell. The C shell is the second most popularly used shell in UNIX™. It was developed by Bill Joy at the University of California at Berkeley. Part of the C shell

syntax is borrowed from the C programming language. The commands **history** and **alias** and the job control mechanism are the most important improvements of C shell on Bourne shell. The history feature allows users to keep track of the issued commands, then go back to execute them without retyping the command. The job control feature allows job switching between background and foreground. The command **alias** allows users to create aliases for command names. C shell supports both local and global variables which are achieved by **set** and **setenv**. The control structures offered by C shell are **if-then-else**, **switch**, **foreach**, **repeat**, and **while**. Expressions in C shell can be evaluated directly.

Korn Shell. The Korn shell was created at AT&T by David Korn, with many features of both the C shell and the Bourne shell. It is compatible with Bourne shell and is the least popular shell among the three. The Sun Workstations on which this work was done does not support the Korn shell. Writing the project in the Korn shell never was considered.

#### GRASS: A Powerful GIS Package

GIS is a system which can collect, manage, manipulate, analyze and display spatial and attribute data [21]. It is a powerful, general purpose software package with great flexibility. The basic requirements to use the software are relatively high. Users must have knowledge of the Geographical Information System as well as the computer, including input, output devices, and operating systems.

GRASS was designed and developed by researchers at the U.S. Army Construction Engineering Research Laboratory. It is written in C and is UNIX™ oriented. GRASS was first released in 1985 and version 4.0 was released in August, 1991. In version 4.0, each command can be used either interactively or through the command-line interface. The command-line interface provides programmers with great working flexibility. GRASS is distributed with source code. Portability is the first priority in GRASS design. It comes before a friendly user interface and execution speed.

#### CMLS: A Simulation Tool for Managing Agricultural Chemicals

CMLS is a computer model designed to simulate the movement and degradation of pesticides in soils. Pesticide movement depends upon soil properties, pesticide properties, weather, and management. The model is designed for use by managers in selecting agricultural pest management systems which minimize risk of degradation of ground water quality. It is well suited for use with GIS systems since its input parameters are readily available for large areas. Monte Carlo techniques are used in CMLS to assess the range of future behaviors of the chemicals since future weather is unknown.

The input of CMLS is a text file. Information inside the text file can be multiple blocks. Each block consists of two groups. One group defines the overall simulation

parameters and the other defines all the soil chemical systems being simulated. Fig 3.2 shows the format of the CMLS input file, where the possible keywords in the input file are listed.

The output of CMLS is written to text files, and the file names are specified by the user in the input file following the keyword `InputFile` in each block. The output from the CMLS can either be time-depth data pairs or time-depth-amount triples depending on the choice of the user; that is, when the user chooses to specify the output amount, the output will be time-depth-amount triples. Otherwise, it will be time-depth data pairs. The first column reports the simulation number. It should start from 1 and end at the number specified by the user in the same block of the input file after keyword `NumberOfSimulations`. The second column is the system index specified by the user following the keyword `SystemIndex` in the beginning of each system in the input file. The data following will be either the time-depth pairs or the time-depth-amount triples. An example of output data is given in Fig 3.3.

When the user wants to get the depths penetrated by the chemical during certain time periods, he needs to specify the times following the keywords `Output TravelTime`. When the user wants to get the travel time taken for the chemical to move to certain depths, he has to specify the depths after the keywords `Output Depth`, and if the output amount is preferred, the user has to put the keyword `Output Amount` in



the input file of that block. The number of simulations performed on the soil chemical systems of a block is defined by an integer following the keyword NumberOfSimulations in that block. This number defines the number of simulations the user wants to perform on the given soil chemical systems of the same block under certain weather condition.

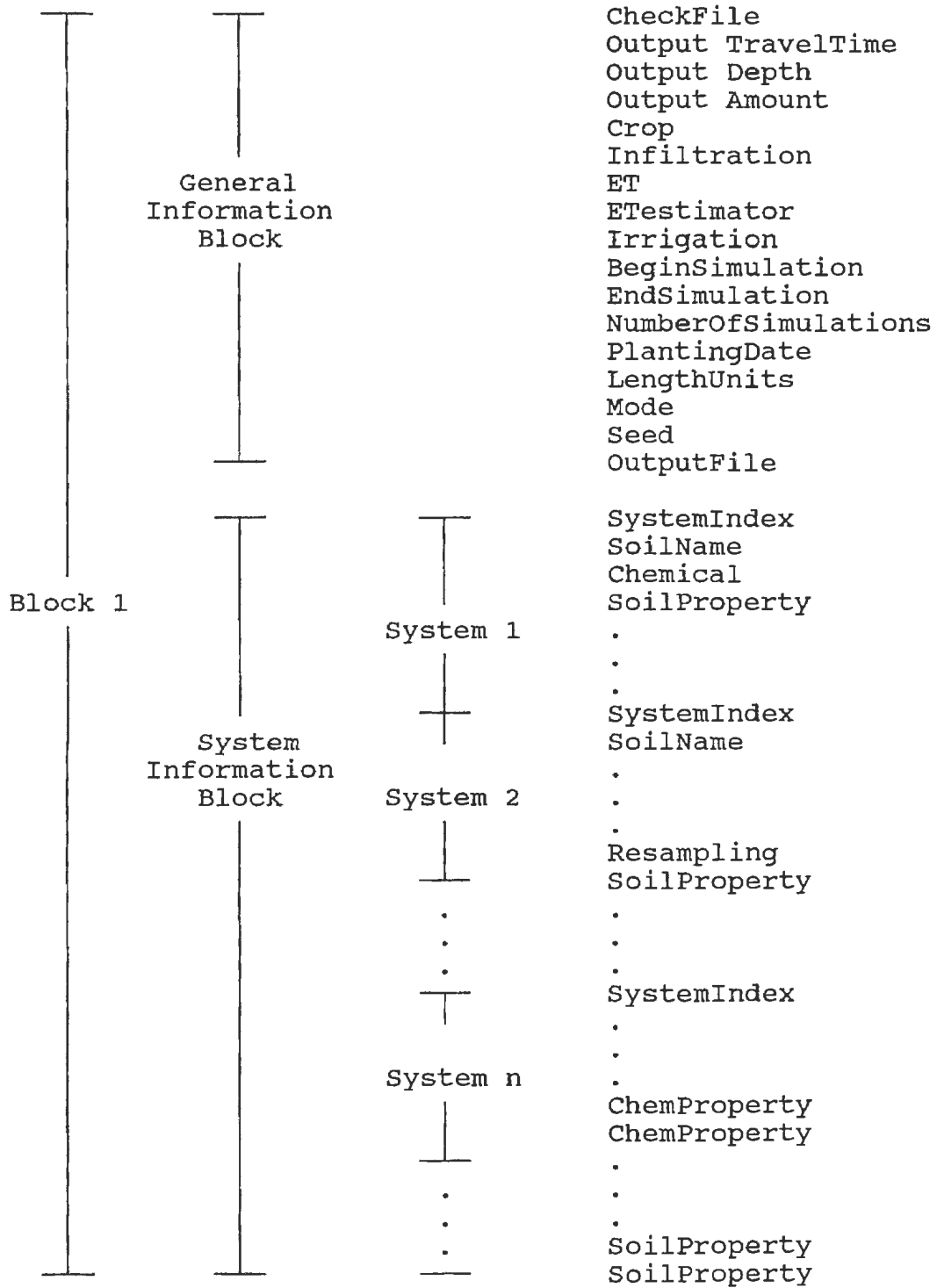


Figure 3.2. CMLS Input Format

sim. sys. time depth (amount) ... time depth (amount) ...

1	1	244	0.100	639	0.200	880	0.300	.....
1	2	10	0.100	61	0.200	61	0.300	.....
1	3	87	0.100	538	0.200	539	0.300	.....
1	4	148	0.100	626	0.200	848	0.300	.....
1	5	49	0.100	119	0.200	119	0.300	.....
2	1	43	0.100	108	0.200	185	0.300	.....
2	2	63	0.100	81	0.200	81	0.300	.....
2	3	462	0.100	1296	0.200	2106	0.300	.....
2	4	143	0.100	509	0.200	623	0.300	.....
2	5	39	0.100	131	0.200	151	0.300	.....
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Figure 3.3. CMLS Output Format (time-depth pairs five soil chemical systems are included in this block)

## CHAPTER IV

### DESIGN AND DEVELOPMENT OF GO & DRAW

#### Introduction

In this chapter, the features of the project are presented. This includes a detailed description of the functionalities that **go & draw** provides and the working environment that the user faces. Finally, a complete explanation of the designing details of the project is given.

#### Functionality

**go & draw** can help its user to create the CMLS input file and allows its user to run CMLS for a selected county and to produce maps from either the CMLS output or the soil and the land use data bases of that county. All the implementation details of both UNIX™ and GRASS have been concealed by using UNIX™ shell programming. After the user selects a county within Oklahoma and chooses to run CMLS for that county, the user only needs to supply some of the data in the general information block of the CMLS input file. They are entered under the prompt of dialogue boxes. The data to be entered by the user from the command line are shown in Fig 4.1. All the other required information to run

CMLS, like the soils and their properties, are extracted by `go & draw` automatically from the associated data bases. `go & draw` creates the input file for CMLS by combining these data via the UNIX™ shell command `awk`. The travel time and/or the depth data specified by the user from the command line are kept in a temporary file by `go & draw`. After `go & draw` invokes CMLS, it extracts the travel time and/or the depth from the temporary file and displays the data in a dialogue box to let the user select the data of interested in making maps. `go & draw` processes the relevant data to meet the requirements of the GRASS map producing programs. Not only can the user produce maps from the CMLS output, but the user can also produce soil and land use maps within the selected county with the help of `go & draw`. The user has the flexibility of displaying the CMLS output map and the soil map or the land use map alternatively by choosing the options provided in the dialogue box. To each displayed map, the user is allowed to zoom in on any particular part of the map to get a detailed map and to go back to the default geographical region of the county after zooming. All the maps can either be displayed on the screen or printed out on hard copy.

`go & draw` allows the CMLS users to accomplish these tasks in a highly user friendly environment. The detailed user interface description of this project is discussed in the next section.

```
Output TravelTime
Output Depth
Output Amount [optional]
Crop Name
Crop File Name
Evapotranspiration (ET) Source
ET File Name
ET Estimator Name
ET Estimator Parameter(s)
Infiltration Source
Infiltration File Name
Irrigation Management Strategy
Parameters for the selected Strategy
Simulation Begin Year and Day
Simulation End Year and Day
Earliest and Latest Planting Day
Number of Simulations
Name of the Output File
Seed
Measure Unit for the Depth
Same or Different Weather for Each System
Application Window
Application Depth and Amount
Chemical Name of Applied
Chemical Koc and half life
Root Depth
```

Figure 4.1. Data Entered from the Command Line

## User Interface

The user interface provided by **go & draw** is mainly through dialogue boxes. Except when certain actions need to be taken inside the graphics monitor where the mouse is used, all other actions are carried out by entering choices from the keyboard on the command line. A dialogue box is used to give clear instructions to the user whenever actions need to be taken by the user.

After the user starts **go & draw**, The graphics monitor is invoked automatically and the map of Oklahoma state is displayed on it. A dialogue box (see Fig 4.2) is used to instruct the user to choose the county of interested using the mouse. Another box (see Fig 4.3) continues to let the user decide if the selection is intended. In this dialogue box, [county] represents the selected county name. After selecting the county, the first dialogue box encountered by the user is the one that asks if the user wants to run CMLS (see Fig 4.4). If so, the user will be prompted to enter the necessary information to run CMLS. The data needed to be entered are shown in Fig 4.1. The dialogue boxes that instruct the user to enter these data are basically in three forms, answering yes or no, making a selection, or typing in the data directly. Examples for these three types of dialogue box are given in Fig. 4.5. When the user finishes entering data, a dialogue box is used to give the user the opportunity to modify the data that he has entered (see Fig

4.6). The user can keep modifying the data until he is satisfied by answering y to dialogue box 4. After this, **go & draw** invokes CMLS automatically. In the process of running CMLS, if an error occurs and if it is related to the data that the user entered, the user still has the chance to modify the data. That is, dialogue box 4 will be displayed to the user again with error messages to tell the user that the data entered need to be modified. **go & draw** will restart automatically to execute CMLS after the user is satisfied with his modifications. When CMLS ends, the user is prompted to select the actions he is interested in taking. This time, the choices given by the dialogue box include choosing the simulation number, displaying the soil or the land use map, zooming the current region, going back to the default geographical region of the county, or quitting this CMLS session (See Figure 4.7). In this box, [No] represents any number entered by the user from the command line for the number of simulations. When the user chooses to display the soil or the land use map or to go back to the default region, the actions followed will be taken by **go & draw** automatically. When the user chooses to zoom the current region, he will be instructed to use the mouse to select a region within the current region. When the user chooses a simulation number, another dialogue box with all the depths and/or travel times entered by the user prior to running CMLS will be displayed (see Fig 4.8 as an example) and the user is prompted to select a number which represents the



data in which the user is interested. After the user makes a selection in the dialogue box, **go & draw** will display the map made from the selected data and go back to dialogue box 5 to let this process continue. Thus, the user can switch back and forth between the CMLS output map and the soil or the land use maps. This process continues until the user selects **q** (quit) in dialogue box 5. Then the user is taken back to dialogue box 7 (See Fig 4.9). This is the same dialogue box that the user would encounter if he answers no to dialogue box 3. In dialogue box 7 and box 8 (See Fig 4.10), the user is given the options of displaying the soil or the land use maps of the county respectively. After that, the execution will go to dialogue box 9 (See Fig 4.11). This process continues until the user answers that he does not want to select another county.

A series of procedures are involved in designing and developing this project. They are discussed in the following section.

Please use mouse to select a county  
left button to select  
right button to confirm

Figure 4.2. Dialogue Box 1 - Instruct the User to Select a County

Selected County is [County], ok ? (y/n)

Figure 4.3. Dialogue Box 2 - Let the User Confirm His Selection

Run CMLS batch ? (y/n)

Figure 4.4. Dialogue Box 3 -  
Give the User  
the Option to  
Run CMLS

Want to specify the output depth ? (y/n)

Enter the depths (float) in one line  
and when finished hit return key

Choose the source of infiltration, please  
a. actual  
h. historical  
g. generated

Figure 4.5. Data Entered in Three Forms

```
THESE ARE THE DATA THAT YOU JUST ENTERED

Output TravelTime 0.1 0.4 0.6 0.9
Output Depth 30 100 500 3000
Crop corn cropfile
ET generated ok3281.par
ETestimator SCSBlaneyCriddle 1.0
Infiltration generated ok3281.par
irrigation none
BeginSimulation 1 1
EndSimulation 10 365
NumberOfSimulations 10
PlantingDate 1 1
Seed -100
LengthUnits
Mode sameweathereachsystem

Any Modifications ? (y/n)
```

Figure 4.6. Dialogue Box 4 - Let the User Modify the Input Data

```

Please enter (1-[No]) for the
simulation preferred
or
s - look at the soil map
l - look at the land use map
z - zoom the area
b - back to the original region
z - quit

```

Figure 4.7. Dialogue Box 5 - Let the User Select an Action

```

Please choose the data that you are interested
in making maps (enter q to quit):
1. 0.1      2. 0.2      3. 0.6      4. 0.6      5. 0.9
2. 30       7. 100       8. 500     9. 3650

```

Figure 4.8. Dialogue Box 6 - Let the User Choose the Interested Data

Look at the soil map for this area ? (y/n)

Figure 4.9. Dialogue Box 7 - Give the User the Option of Looking at the Soil Map for the Same Area

Look at the land use map for this area ? (y/n)

Figure 4.10. Dialogue Box 8 - Give the User the Option of Looking at the Land Use Map for the Same Area

Another County ? (y/n)

Figure 4.11. Dialogue Box 9 - Ask If the User Wants to Work on Another County

## System Description

All the functions described earlier are performed inside GRASS. To produce maps using `go & draw`, the user must get into GRASS first. `go` is the procedure that carries out this task. By typing "go" from the command line, the procedure will guide the user to get into GRASS.

As long as the user is inside GRASS. `draw` becomes the procedure that initiates all the other procedures. By typing "draw" inside GRASS, the user will be instructed to take certain actions or to make certain choices in order to finish the job that he intended to.

Procedure `draw` implements all necessary processes by calling other procedures. The designing and selecting of a certain procedure is based on modularity. The relationship between `draw` and other procedures which are called by `draw` directly or indirectly are shown in Fig 4.12.

The functions that each procedure performs are presented below:

1. `draw`. Read in the directory where the script procedures are stored and the directories containing necessary database files. Start the graphics monitor which will be used to display maps during the session. Display Oklahoma state map on the graphics monitor. Procedure `sel_county` is called to continue the execution. The process will not end until the user explicitly answers that he does not want to choose

- another county.
2. **sel\_county.** Lets the user select a county within Oklahoma by using the mouse. Give the user the option to run CMLS. If the user chooses to run CMLS, procedures **get\_input**, **mod\_gen**, **draw\_map** are called, if the user selects to skip the option of running CMLS, display the soil map and the land use map options will be given and certain GRASS functions will be called.
  3. **get\_input.** The soils and their properties for the selected county are extracted from the soil attribute data base and the soil curve number for each soil are extracted from the soil data base. The extracted data are combined and processed to create one system information block of the CMLS input file. Procedure **gen\_infor** is called to read in data from the command line. Procedure **mod\_gen** is called to let the user modify the data that he has entered. The general information block data and the system information block data are combined to form the input file for CMLS.
  4. **gen\_infor.** Lets the user enter the limited data necessary for CMLS. Whenever certain data need to be entered or certain choices need to be selected by the user, a dialogue box will instruct the user to take the relevant action.
  5. **mod\_gen.** Lets the user modify the data that he has entered in procedure **gen\_infor**. The data will be displayed line by line. When there is no change in a



line, the user only needs to hit the return key. When there is any change in a line, the user must type everything in the line except the keyword(s).

6. **draw\_map.** Display dialogue box 5 to let the user select the choice that he prefers, which includes picking up a simulation number, looking at the soil map, looking at the land use map, zooming the current mapping area, returning to the original region from the zoomed area and quitting for this CMLS session. If the user chooses to pick up a simulation number, dialogue box 6 will be displayed to let the user select the depth and/or travel time for making a map. The procedure automatically processes CMLS output by first picking up certain rows and columns from the output file and then grouping the selected data. Finally, GRASS functions are called to set the region and display the map. Any other choices selected by the user will lead the procedure to take the corresponding actions.
7. **pre\_pmap.** Write a text file which specifies all the necessary information in drawing the CMLS output map. This text file is redirected to GRASS command **p.map**.
8. **pre\_soils.** Write a text file which specifies all the necessary information in drawing the soil map of the selected county. This text file is redirected to GRASS **p.map**.

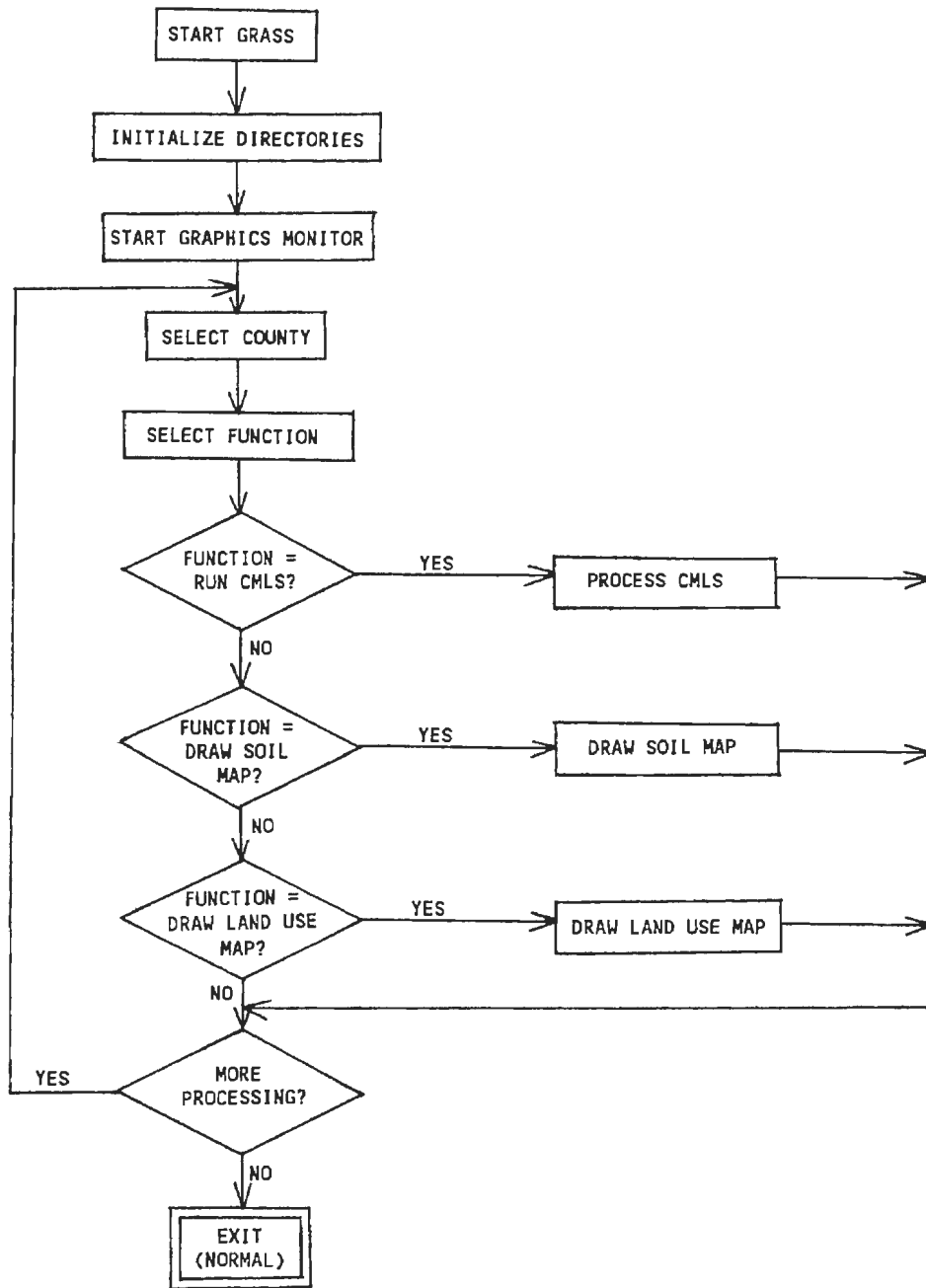


Figure 4.12. Execution Sequence of the Map Drawing Tool

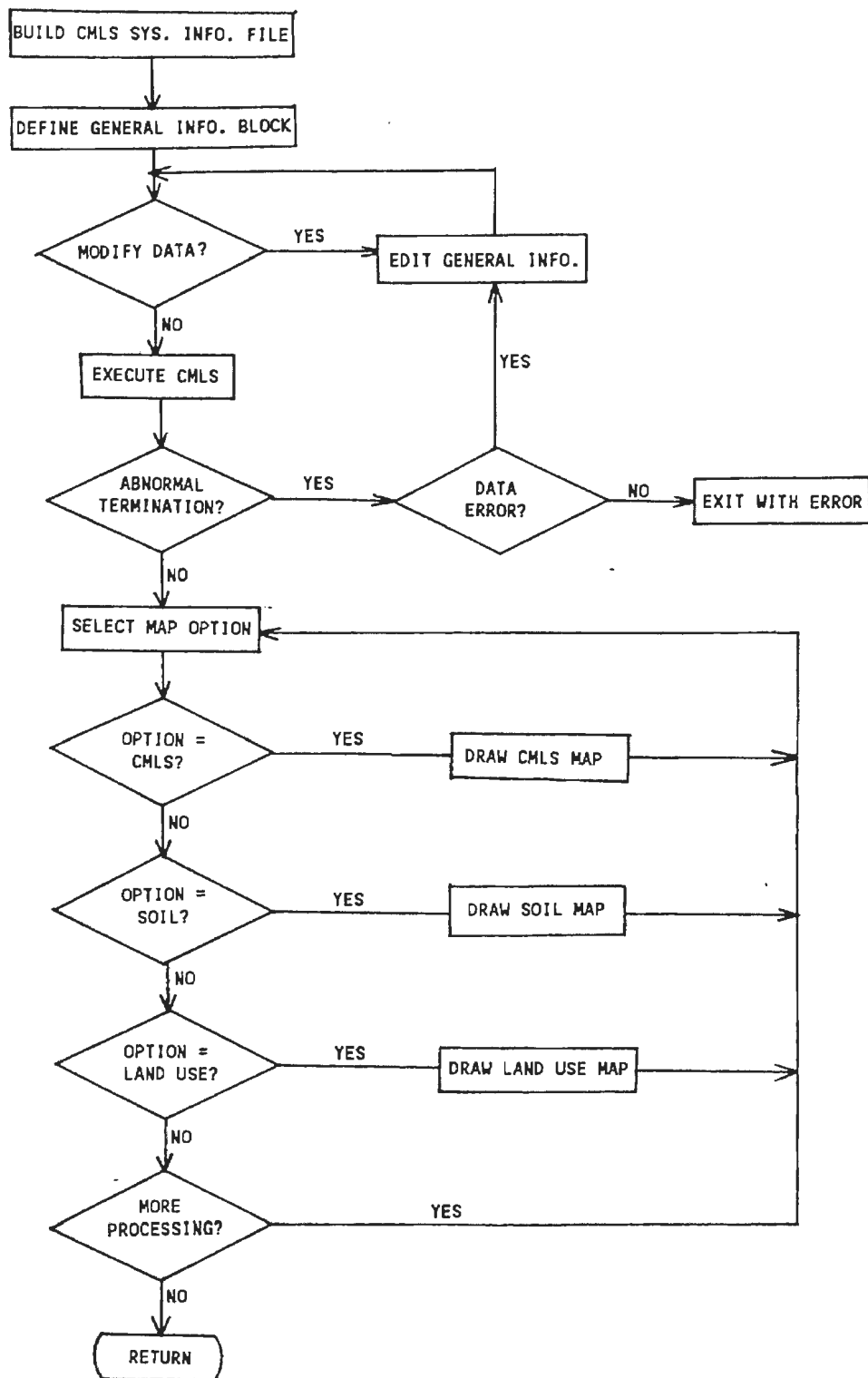


Figure 4.13. Process CMLS

9. **pre\_land.** Write a text file which specifies all the necessary information in drawing the land use map of the selected county. This text file is redirected to GRASS command **p.map**.

The procedures perform their functions with the help of the UNIX™ shell utilities and the shell logic constructs. A detailed discussion of selecting and organizing these utilities is provided in the next section.

#### Implementation Details

A large amount of text processing is involved during the implementation of **go & draw**. This is carried out by using the UNIX™ shell commands **awk**, **cut**, **paste**, **sed**, **tr**, and so on. The logical constructions offered by the UNIX™ shell are used as the glue to connect all the building blocks (the shell tools). Pipe, filter, append symbols (**<<**, **>>**), I/O redirection signs (**<**, **>**) are also widely used within the project.

The data in Figure 4.1 are entered by the user one by one under the prompt of dialogue boxes. As part of the input data, they are appended together by using the output append symbol (**>>**) and UNIX™ utility **echo**. The appended data are saved in a temporary file for later use. Logic constructs **while** and **if** are used to assure the proper flow logic of the project.

The process of letting the user continue to modify the data he entered is fulfilled by using UNIX™ utility **awk** and

the logic construct `while`. `awk` is used to check whether the line entered by the user is an empty line. That is, when the user does not want to modify anything in this line by choosing to hit the return key directly. The idea behind this is to use new line as the field separator for the `awk` program. The number of fields in a line will not be one unless the user hits the return key directly, that is, there is no change in this line. If there is any change in a line, the new data entered by the user replace the old. If the user hits the return key directly, the old data are kept. The `while` construct keeps the modify process continuing. When the user chooses not to modify any data, a `break` statement is used. The status of the CMLS execution is checked using the C shell built in variable `status`. A single procedure `mod_gen` is designed to let the user modify the data. When errors occur inside CMLS, by simply invoking `mod_gen` again, the user can perform the data modification process.

If CMLS terminates normally, the output file is split into multiple files by using UNIX™ utility `split`. Each file contains the output from a single simulation. For example, if the user entered 5 for the number of simulations in the command line, the output file will be split into 5 files. If there is only one simulation, the split process is skipped. Which file is used in the following step depends upon the number chosen by the user in dialogue box 5.

After the user chooses the data in making the map, `awk`

and `cut` are used to pick up certain columns from the selected single simulation output file.

Finally, the data are grouped and organized using `awk` to meet the requirements of the GRASS map producing commands. The correct region is set by using GRASS command `g.region` and GRASS command `p.map` is used to make the map.

The temporary files generated during the implementation of the project are put in the directory `/tmp`, a UNIX™ directory for temporary files. A temporary file is removed as soon as its functions are finished. Each temporary file name is made unique by naming it using the process ID of the running procedure with a different extension.

Modularity is of first priority when it comes to selecting and designing a procedure. Though the `goto` statement is allowed by the C shell, it is never used within the project.

## CHAPTER V

### DESIGNING GUIDELINES

The intention of developing this project (**go & draw**) is to provide a friendly working environment for the CMLS users who want to produce maps from the CMLS output but are neither GIS experts nor computer experts. By using **go & draw**, the CMLS users can produce maps from the CMLS output or some existing data bases.

The most outstanding feature of this project is its user-friendly interface. Since the motivation of developing this project is to let the users with limited computer skills perform complex tasks without great effort. The friendly user-interface naturally decides the project's success and becomes the heart of **go & draw**.

Other than the user-friendly interface, some other quality factors of shell procedures, introduced by Arthur [2] are also of great concern in designing process. They are the reliability, the maintainability and the efficiency. These quality factors are discussed respectively below.

#### Reliability

The reliability of the project is understood as it performs the correct, intended action and rarely fails. It has taken effect by several efforts, like always checking

the range of input data. When a dialogue box is displayed and the user is asked to enter data, the data entered by the user will be checked for range and value. If there is any mistake, the same dialogue box will be displayed to the user. Therefore the user must re-enter the data. Fault handling is enforced with the help of the `exit` statement and the C shell predefined variable `status`. When any abnormality occurs after the user typed "draw", and it is related to the CMLS input data which was entered by the user from the command line, procedure `mod_gen` will be invoked to allow the user to have the opportunity to modify the data. When errors other than that occur, the project will exit and give control back to the C shell with an abnormal termination signal. Interrupts are handled by using the interrupt handler. That is, if any interrupt occurs when a procedure is executing, the execution flow will be transferred to a place inside the procedure where all the temporary files and intermediate data files are deleted. Then control is transferred back to the calling procedure.

### Maintainability

The maintainability of the project is enforced through the following points:

#### Modularity

Small, modular shell procedures help to improve the maintainability of the whole project. These small procedures



are easy to understand. Each procedure carries out a relatively primitive task. When the same function must be performed repeatedly, or must be performed at different places, it is only required to invoke the procedure which carries out that function. Flow control is achieved with the help of shell control constructions, such as, **if**, **while**, **switch** and so on. Although the **goto** is allowed in the C shell, it is never used within this project.

### Consistency

Consistency is another quality factor upon which the maintainability depends. Applying a consistent programming style in each individual procedure is one way to ensure the consistency of the shell scripts. Other than that, choosing the simpler command when there are several choices available and indenting the shell control constructions to show the structure of the procedure are also helpful in keeping the consistency of the project in good shape.

### Efficiency

Time efficiency of the project is the major concern in the design process. The following items make the project run faster.

### Change Directory

When the user types "draw" inside GRASS, each procedure invoked by **draw** directly or indirectly changes to

the directory where the shell procedures are stored. Thus, the project does not need to search through directory after directory. The efficiency of the project can be improved.

### Reduce Temporary Files

The number of temporary files used can also affect the efficiency of the project. The number of temporary files can be reduced by using pipes whenever possible. Due to UNIX™'s rich set of commands, many tasks can be performed in several different ways. Carefully choosing commands is another way of reducing the number of temporary files.

### Using Time to Measure the Speed

The UNIX™ command `time` can be used to determine the execution time of the shell procedures. By typing "time" following the procedure name which needs to be measured. `time` will reports its data in seconds on the diagnostic output.

Because most of the shell procedures designed in this project ask the user to provide input interactively, the execution times for such procedures are measured by mainly measuring the single commands which are related to using temporary files or are time consuming text processing commands. No attempt has been made to measure the execution time of a whole procedure at once.

## Documentation

Since the map drawing tool will be commercially available, later improvement and enhancement are inevitable. Good documentation becomes the important key to assuring the future work. For each procedure inside the project, a companion text file is produced to document the functionalities and key points of that procedure.

## CHAPTER VI

### SUMMARY, CONCLUSIONS AND FUTURE WORK

#### Summary and Conclusions

As stated in Chapter I, the purpose of designing this project is to provide the CMLS user with a powerful map drawing tool. The CMLS user can produce maps with great efficiency but little effort. Early results of demonstrating the project have been very promising. The finished project displays this capability by user-friendly interface and reliability.

The whole designing process of the project follows the designing guidelines stated in Chapter V. As a result, not only does the project have the functionalities introduced in Chapter IV, but also are the later improvement and enhancement are assured.

The finished project provides the CMLS user with a powerful, convenient working tool. Due to time restriction, however, some features and functionalities that the project might have are not added in this thesis. They are stated in the next section.

#### Future Work Suggestions

1. After the user entered all the necessary data for

running CMLS and dialogue box 8 are displayed to the user, if the user chooses to modify the data, the modification steps followed are carried out by using UNIX™ tool **awk** (See Chapter IV for details). There is a lot of text processing involved each time the procedure tries to determine if the input is a new line. This can be revised by writing a small C program and install it in the user's working environment. Thus, the procedure can invoke this compiled C program just like a UNIX™ command. This will improve the response time.

2. CMLS program allows more than one block of information in the input file (See Chapter III for details). The current project only allows one block of data in the input file. If the capability of allowing more blocks of information in the input file were possible, the project would become more powerful.
3. Resampling is allowed in CMLS program, but the current project is not capable of handling this. Adding this feature to the project can also increase its capability.
4. The current project asks the user to supply one weather file from a single weather station for CMLS. If the project were able to automatically detect the weather stations that affect to a county and run CMLS based on the multiple weather stations, the maps produced based on such CMLS output would be more useful.

5. Each time the user typed "draw" in the command line, certain directory names have to be supplied (see Chapter IV for details). If a installation program could be written to ask for those directory names and the user did not need to be aware of those directories later, the project would be more easy to use.
6. The current project allows its user only produce CMLS output map based on current CMLS running session. If the output from the previous CMLS running sessions could also be used in producing maps, the project would be more flexible and powerful.
7. The current project can only produce CMLS output map on one single travel time or one single depth. If the project could calculate travel time or depth probability based on the CMLS output and produce the probability maps, the result would be more practical.

## BIBLIOGRAPHY

1. Anderson G. and Anderson P. The UNIX™ C Shell Field Guide, Prentice-Hall, Englewood, New Jersey, c1986.
2. Arthur L.J. UNIX™ Shell Programming, John Wiley & Sons, Inc, c1990.
3. Bourne S.R., The UNIX™ Shell, Bell Systems Technical Journal Vol. 57, July-August 1978, pp. 1971-1991.
4. Comeau G., The UNIX™ Shell, BYTE, September 1989, pp. 315-321.
5. Doane S.M., McNamara D.S., Kintsch W., Polson P.G, and Clawson D.M., Prompt Comprehension In UNIX™ Command Production, Memory & Cognition, 20:4, 1992, pp. 327-343.
6. Dolotta T.A., Haight R.C., and Mashey J. R., The Programmer's Workbench, Bell Systems Technical Journal, Vol. 57, July-August 1978, pp. 2177-2199.
7. Fiedler D., Customizing For Comfort, Byte, November 1989, pp 139-142.
8. Horowitz E., and Munson J.B., An Expansive View of Reusable Software, IEEE Transactions on Software Engineering, Vol. 10, September 1984, pp. 477-487.
9. Johnson S.C., and Lesk M.E., Language Development Tools, Bell Systems Technical Journal, Vol. 57, July-August 1978, pp. 2155-2175.
10. Kernighan B.W. The UNIX™ System and Software Reusability, IEEE Transactions on Software Engineering, Vol. 10, September 1984, pp. 513-518.
11. Kernighan B.W. and Mashey J.R., The UNIX™ Programming Environment, IEEE Computer, April 1981, pp. 12-24.
12. Luderer W.R., Maranzano J.F. and Tague B.A., The UNIX™ Operating System as a Base for Applications, Bell Systems Technical Journal, Vol. 57, July-August 1978, pp. 2201-2207.
13. McIlroy M.D., Pinson E.N. and Tague B.A., Foreword,

Bell Systems Technical Journal, Vol. 57, July-August 1978, pp. 1899-1904.

14. Nofziger D.L. and Hornsby A.G., Chemical Movement in Layered Soils: User's Manual, Circular 780, Florida Cooperation Extension Service, Institute of Food and Agriculture science, University of Florida, Gainesville. FL. 1992, pp. 44.
15. Ritchie D.M., A Retrospective, Bell Systems Technical Journal, Vol. 57, July-August 1978, pp. 1947-1969.
16. Ritchie D.M. and Thompson K., The UNIX™ Time-Sharing System, Bell System Technical Journal, Vol. 57, July-August 1978, pp. 1905-1929.
17. Ryan B., Scripts Unbounded, BYTE, August 1990, pp. 235-240.
18. Pike R. and Kernighan B.W., Program Design in the UNIX™ Environment, Bell System Technical Journal, Vol. 63, October 1984, pp. 1595-1605.
19. Smith B., The BYTE UNIX™ Benchmarks, March 1990, pp. 273-277.
20. Sobell M.G., A Practical Guide to the UNIX™ System, The Benjamin-Commings Publishing Company, Inc. 1989.
21. GRASS 4.0 Programmer's Menu. Engineers Construction Engineering Research Laboratory (Unpub.), August, 1992.
- ( 5 ] 22. GRASS 4.0 User's Menu, Engineers Construction Engineering Research Laboratory (Unpub.), July, 1991.



## APPENDIXES

APPENDIX A

GRASS COMMANDS USED IN THE  
MAP DRAWING TOOL

## GRASS COMMANDS USED IN THE MAP DRAWING TOOL

For reference, all the GRASS commands used in the map drawing tool are listed in this appendix, The source for the description of each command is from the GRASS Reference Menu.

- d.frame** - Manages display frames on the user's graphics monitor.
- d.erase** - Erase the contents of the active display frame on the user's graphics monitor.
- d.mon** - To establish and control use of a graphics display monitor
- d.rast** - Display and overlays raster map layers in the active display frame on the graphics monitor
- d.what.rast** - Display and overlays raster map layers in the active display frame on the graphics monitor.
- d.zoom** - Allows the user to change the current geographic region settings interactively with a mouse.
- exit** - Exits the user from the current GRASS session.
- g.region** - Program to manage the boundary definitions for the geographic region.
- p.map** - Hardcopy colour map output utility.
- r.patch** - Creates a composite raster map layer by using known category values from one (or more) map layer(s) to fill in areas of "no data" in another map layer.
- r.reclass** - Creates a new map layer whose category values are based upon the user's reclassification of categories in an existing raster map layer
- r.stats** - Generates area statistics for raster map layers.

APPENDIX B

THE PSEUDO-CODE OF THE  
MAP DRAWING TOOL

### Procedure **go**

1. Display the dialogue box to prompt the user to enter the directory name where GRASS is stored  
Read in the directory name
2. Change to the directory where GRASS is stored
3. Display the dialogue box to instruct the user to hit Esc when GRASS is started
4. Wait for 2 seconds for the user to read the instruction
5. Invoke GRASS.
6. End of **go**

### Procedure **draw**

1. Change to the user's home directory
2. Alias **rm** to **rm -f** for forcing removing files
3. Set variable **tf** to **/tmp/\$\$** for later naming temporary files
4. Prompt the user to enter the directory where all the shell procedures are stored  
Read in the directory name
5. Change to the directory where all the shell procedures are stored
6. Prompt the user to enter the directory name where all the soil files are stored  
Read in the directory name
7. Prompt the user to enter the directory name where all the soil property files are stored  
Read in the directory name
8. Check the environment (OpenWindows™ or SunView™) with the help of GRASS command **d.mon**
9. Display the dialogue box to instruct the user to adjust the graphics monitor size if the user is not satisfied with the default size when the graphics monitor displayed
10. If the graphics monitor is not started
  - A. If the environment is SunView™  
Start the graphics monitor for SunView™
  - B. If the system is OpenWindows™  
Start the graphics monitor for OpenWindows™
11. If the user enter continue for satisfying with the current graphics monitor size
  - A. Frame the graphics monitor using GRASS command **d.frame**
  - B. Set the current geographic region as Oklahoma state region
  - C. Clear the graphics monitor
  - D. Display Oklahoma state map on it
  - E. Call procedure **sel\_county**
12. While true
  - A. Display the dialogue box to ask the user if he wants to select another county

- Read in the user's answer
  - B. If the user selects yes
    - a. Set the geographic region to Oklahoma state
    - b. Clear the graphics monitor
    - c. Display the Oklahoma state map on it
    - d. Call procedure sel\_county
  - Otherwise
  - Break
- 13. End of draw

#### Procedure Sel\_county

1. Set tf to /tmp/\$\$ for later naming temporary files
2. Alias rm to rm -f for forcing removing files
3. While true
  - A. Display the dialogue box to instruct the user to use the mouse to select a county
  - B. Using GRASS command d.what.rast, UNIX™ commands tee and awk to get the name of the selected county
  - C. Remove the temporary files that from the previous step
  - D. Display the dialogue box to inquire the user if he is satisfied with his selection  
Read in the user's answer
  - E. If the user answers yes  
Break
4. Write the selected county name to a temporary file
5. Using the UNIX™ command tr to convert all the letters of the county name to lower case
6. Remove the temporary file
7. Display the dialogue box to inquire the user if he wants to run CMLS  
Read in the user's answer and store it in variable choice
8. While the user answers anything other than no
  - A. Call procedure get\_input
  - B. Invoke CMLS
  - C. Check the status of CMLS execution process
    - a. If the status is 1  
Terminate the shell tool with an abnormal termination signal
    - b. If the status is 2
      - aa. Display a dialogue box to tell the user that data entered has errors
      - bb. Call procedure mod\_gen
    - c. If the status is 0
      - aa. Call procedure draw\_map
      - bb. Set choice to no
9. Display the dialogue box to give the user the option of

- looking at the soil map  
Read in the user's answer
10. If the user answers yes
    - A. Set the geographic region to the selected county
    - B. Clear the graphics monitor
    - C. Display the soil map
  11. Display the dialogue box to give the user the option of looking at the land use map  
Read in the user's answer
  12. If the user answers yes
    - A. Set the geographic region to the selected county
    - B. Clear the graphics monitor
    - C. Display the land use map
  13. End of sel\_county

#### Procedure get\_input

1. Set variable tf to /tmp/\$\$ for later naming the temporary files
2. Alias rm to rm -f for forcing removing files
3. Copy the selected county soil property file to a temporary file
4. Using UNIX™ command sed to change the copied soil property file field separator from "," to ":"
5. Using UNIX™ command awk to make the first fields of all the lines in the soil property file two digits
6. Using UNIX™ command awk to remove the first four lines of the selected county soil file
7. Using UNIX™ command awk to make the first fields of all the lines in the soil file two digits
8. Using UNIX™ command join to do relational join on the soil file and the soil property file
9. Using UNIX™ command awk to convert the joined file to fit the requirements of CMLS input file system information block, name the file prop
10. Call procedure gen\_infor
11. Call procedure mod\_gen
12. End of get\_input

#### Procedure gen\_infor

1. Alias rm to rm -f for forcing removing files
2. While true
  - A. Display the dialogue box to ask the user if he wants to specify the output depth  
Read in the user's answer
  - B. If the user answers yes
    - a. Display the dialogue box to instruct the user to enter the

- depths
    - Read in the depths
    - b. Write the depths with keyword  
Output Traveltime to file gen
    - c. Write the depths with keyword depth  
to a temporary file mid for later  
use
  - C. Display the dialogue box to ask the user if  
he wants to specify the output travel time  
Read in the user's answer
  - D. If the user answers yes  
Display the dialogue box to instruct the  
user to enter the specified travel times  
Read in the travel times
  - E. If the user specified both depth and travel  
time
    - a. Append the travel times with  
keyword Output Depth to file gen
    - b. Append the travel times with  
keyword time to file mid
  - F. If the user only specified travel time
    - a. Write the travel times with keyword  
Output Depth to file gen
    - b. Write the travel times with keyword  
time to file mid
  - G. If the user specified neither travel time nor  
depth
    - Display error message
    - Otherwise
    - Break
- 3. Display the dialogue box to ask the user if he wants to  
specify the output amount  
Read in the user's answer
- 4. If the answer is yes  
Append keyword Output Amount to file gen
- 5. Display the dialogue box to let the user enter crop  
name  
Read in crop name
- 6. Display the dialogue box to let the user enter crop  
file name  
Read in the crop file name
- 7. If the file does not exist  
Display error message
- 8. Append the crop name and crop file name with keyword  
Crop to file gen
- 9. Display the dialogue box to let the user select the  
evapotranspiration (ET) source  
Read in the user's selection
- 10. Display the dialogue box to let the user enter the ET  
file name  
Read in the file name
- 11. If the file does not exist  
Display error message
- 12. Display dialogue box to let the user select the ET



- estimator name  
Read in the user's selection
13. Display the dialogue box to let the user enter the ET estimator parameter(s)  
Read in the parameter(s)
  14. Append the ET source, ET file name, ET estimator name and the parameter(s) with keyword ET to file gen
  15. Display the dialogue box to let the user select the infiltration source  
Read in the user's selection
  16. If the user selects both the ET source and the infiltration source as not generated
    - A. Display the dialogue box to prompt the user to enter the infiltration file name
    - B. Append infiltration source and infiltration file name with keyword Infiltration to file gen

Otherwise  
Append infiltration source and ET file name with keyword Infiltration to file gen
  17. Display the dialogue box to let the user to select the irrigation management strategy  
Read in the irrigation management strategy, and if the irrigation strategy is
    - A. None  
Append None with keyword Irrigation to file gen
    - B. Actual
      - a. Display the dialogue box to ask the user for irrigation file name  
Read in the file name
      - b. Append Actual and irrigation file name with keyword Irrigation to file gen
    - C. Periodic
      - a. Display the dialogue box to prompt the user to enter irrigation begin day, irrigation end day, number of irrigation days and irrigation amount in one line  
Read in all the data at once
      - b. Append all the read in data with keyword Irrigation to file gen
    - D. Demand
      - a. Display the dialogue box to prompt the user to enter irrigation begin day, irrigation end day, critical water depletion level and minimum irrigation amount in one line  
Read in all the data at once
      - b. Append all the read in data with keyword Irrigation to file gen
  18. Display the dialogue to let the user enter simulation begin year

- Read in the year
19. Display the dialogue box to let the user enter simulation begin day  
Read in the day
  20. Append simulation begin year and day with keyword BeginSimulation to file gen
  21. Display the dialogue box to let the user enter simulation end year  
Read in the year
  22. Display the dialogue box to let the user enter simulation end day  
Read in the day
  23. Append the simulation end year and day with keyword EndSimulation to file gen
  24. Display the dialogue box to let the user enter the earliest planting day  
Read in the day
  25. Display the dialogue box to let the user enter the latest planting day  
Read in the day
  26. Append the earliest and the latest planting day with keyword PlantingDate to file gen
  27. Display the dialogue box to let the user enter the number of simulation times that he prefers  
Read in the number
  28. Append the number with keyword NumberOfSimulation to file gen
  29. Display the dialogue box to let the user enter the file name to which the CMLS output is going to be written
  30. Append the file name with keyword OutputFile to file gen
  31. Display the dialogue box to let the user enter a number for seed
  32. Append the number with keyword Seed to file gen
  33. Display the dialogue box to let the user select the measure unit for depth  
Read in the user's selection
  34. Append the user's selection with keyword DepthUnits to file gen
  35. Display the dialogue to ask the user if he wants to change weather for each soil chemical system  
Read in the user's selection and if the user answers yes  
    - Append newweathereachsystem with keyword Mode to file gen
    - Otherwise  
Append sameweathereachsystem with keyword Mode to file gen
  36. Display the dialogue box to let the user select the application day type  
Read in the user's selection
  37. Display the dialogue box to let the user enter application year  
Read in the year

38. Display the dialogue box to let the user enter application begin day  
Read in the day
39. Display the dialogue box to let the user enter application end day  
Read in the day
40. Display the dialogue box to let the user enter application depth  
Read in the depth
41. Display the dialogue box to let the user enter application amount  
Read in the amount
42. Display the dialogue box to let the user enter Koc  
Read in the value
43. Display the dialogue box to let the user enter half life  
Read in the value
44. Display the dialogue box to let the user enter chemical name  
Read in the chemical name
45. Append all the data read in from 36 to 44 with keyword Chemical to file gen
46. Display the dialogue box to let the user enter root depth  
Read in the value
48. Append the value with keyword RootDepth to file gen
49. End of gen\_infor

#### Procedure mod\_gen

1. Set variable tf to /tmp/\$\$ for naming temporary file
2. Alias rm to rm -f for forcing removing temporary files
3. Set nonomatch
4. Using UNIX™ command awk to get the number of lines in file gen
5. While true
  - A. Display the dialogue box to let the user browse the data that he has entered in procedure gen\_infor and ask the user if he wants to modify the data  
Read in the user's response  
If the response is no  
Break
  - B. Remove file mid and inf
  - C. Display the dialogue box to instruct the user to modify the data
  - D. Display the first line of file gen  
Read in the user's response  
If the response is not a newline
    - a. Write the entered data with the corresponding keyword to a temporary file temp
    - b. Write the data with the

- corresponding keyword to file mid
- Otherwise
- a. Write the line in file gen to file temp
  - b. Write the line with keyword to file mid
- E. Display the second line of file gen  
 Read in the user's response  
 If the response is not a newline
- a. Append the entered data with the corresponding keyword to file temp
  - b. If the keyword for the second line is Output Depth  
 Append the data with keyword time to file mid
- F. Display the third line of file gen  
 Read in the user's response  
 If the response is not a newline  
 Append the data with corresponding keyword to file temp
- Otherwise  
 Append the third line in file gen to file temp
- G. Display the fourth line of file gen  
 Read in the user's response  
 If the response is not a new line  
 Append the data with corresponding keyword to file temp
- Otherwise  
 Append the fourth line in file gen to file temp
- H. Display the fifth line of file gen  
 Read in the user's response  
 If the response is not a new line  
 Append the data with corresponding keyword to file temp
- Otherwise  
 Append the fifth line in file gen to file temp
- I. Display the sixth line of file gen  
 Read in the user's response  
 If the response is not a new line  
 Append the data with corresponding keyword to file temp
- Otherwise  
 Append the sixth line in file gen to file temp
- J. Display the seventh line of file gen  
 Read in the user's response  
 If the response is not a new line  
 Append the data with corresponding keyword to file temp
- Otherwise  
 Append the seventh line in file gen to

- file temp
- K. Display the eighth line of file gen  
Read in the user's response  
If the response is not a new line  
    Append the data with corresponding  
    keyword to file temp  
Otherwise  
    Write the eighth line in file gen to  
    file temp
- L. Display the ninth line of file gen  
Read in the user's response  
If the response is not a new line  
    Append the data with corresponding  
    keyword to file temp  
Otherwise  
    Append the ninth line in file gen to  
    file temp
- M. Display the tenth line of file gen  
Read in the user's response  
If the response is not a new line  
    Append the data with corresponding  
    keyword to file temp  
Otherwise  
    Append the tenth line in file gen to  
    file temp
- N. Display the eleventh line of file gen  
Read in the user's response  
If the response is not a new line  
    Append the data with corresponding  
    keyword to file temp  
Otherwise  
    Append the eleventh line in file gen to  
    file temp
- O. Display the twelfth line of file gen  
Read in the user's response  
If the response is not a new line  
    Append the data with corresponding  
    keyword to file temp  
Otherwise  
    Append the twelfth line in file gen to  
    file temp
- P. Display the thirteenth line of file gen  
Read in the user's response  
If the response is not a new line  
    Append the data with corresponding  
    keyword to file temp  
Otherwise  
    Append the thirteenth line in file gen  
    to file temp
- Q. Display the fourteenth line of file gen  
Read in the user's response  
If the response is not a new line  
    Append the data with corresponding  
    keyword to file temp

- Otherwise
  - Append the fourteenth line in file gen to file temp
- R. Display the fifteenth line of file gen  
 Read in the user's response  
 If the response is not a new line  
 Append the data with corresponding keyword to file temp  
 Otherwise
  - Append the fifteenth line in file gen to file temp
- S. If the number of lines in file gen is 16  
 Display the sixteenth line of gen  
 Read in the user's response  
 If the response is not a new line  
 Append the data with corresponding keyword to file temp  
 Otherwise
  - Append the sixteenth line in file gen to file temp
- T. If the number of lines in file gen is seventeen  
 Display the seventeenth line of gen  
 Read in the user's response  
 If the response is not a new line  
 Append the data with corresponding keyword to file temp  
 Otherwise
  - Append the seventeenth line in file gen to file temp
- U. Remove file gen
- V. Move file temp to file gen
- 6. Write all the lines in file gen that not start with "RootDepth" to file gen1
- 7. Write the line in file gen that start with "RootDepth" to file rd
- 8. Concatenate file rd with file prop and name the new file sys
- 9. Remove file prop and rd
- 10. Append the number following "RootDepth" to line start with keyword CurveNoRootDepth in file sys
- 11. Concatenate file sys and file gen1
- 12. Distribute line start with Chemical to each soil chemical system and name the finished CMLS input file inf
- 13. End of mod\_gen

#### Procedure draw\_map

1. Set variable tf to /tmp/\$\$ for later naming temporary files
2. Set nonomatch
3. Alias rm to rm -f for forcing removing files

4. Using UNIX™ command `awk` to get the CMLS output file name from file `inf`
5. Using UNIX™ command `awk` to get the number of simulations from file `inf`
6. Using UNIX™ command `awk` to get the number of lines in file `mid` and save it in variable `no_line`
7. Using UNIX™ command `awk` to get the first field of the first line in file `mid`
8. Using UNIX™ command `awk` to get the first field of the second line in file `mid`
9. Using UNIX™ command `awk` to get the number of data in the first line of file `mid`
10. Using UNIX™ command `awk` to get the number of data in the second line of file `mid`
11. Using UNIX™ command `awk` to calculate the number of soil chemical systems in file `inf`
12. Split the CMLS output file into multiple files and each file contains only one simulation output
13. Get the selected county soil map region from the coordinates data base file and set the geographic region to the soil map region
14. Clear the graphics monitor
15. Initialize counter `sim` to 0
16. Set `flag` to out
17. Set `zoom_flag` to off
18. While true
  - A. If the number of simulation is not 1
    - While `sim` is greater than number of simulation times or `sim` is smaller than 1
    - 1
      - a. Display the dialogue box to let the user make a selection among picking up a simulation number, displaying the soil map, displaying the land use map, zooming the current geographic region, going back to the original region and quitting the session  
Read in the user's selection and store it in `sim`
      - b. If the user selects to quit  
Do nothing
      - c. If the user selects to zoom
        - aa. If `flag` is out  
Call procedure `pre_soils`  
Display the soil map on the screen
        - bb. Call GRASS procedure `d.zoom`
        - cc. Clear the graphics monitor
        - dd. Set `flag` to in

- ee. Set zoom flag to on
    - ff. Set sim to 0
  - d. If the user selects to go back to the original region
    - aa. Get the geographic coordinates for the soil map of the selected county from the coordinate data base file and set the region
    - bb. Clear the graphics monitor
    - cc. Set flag to in
    - dd. Set zoom flag to off
    - ee. Set sim to 0
  - e. If the user selects to display the soil map
    - aa. If the zoom flag is off
      - Get the coordinates for the soil map from the coordinate data base and set the geographic region
    - bb. Clear the graphics monitor
    - cc. Call procedure **pre\_soils**
    - dd. Call GRASS command **p.map**
    - ee. Set sim to 0
    - ff. Set flag to in
  - f. If the user selects to display the land use map
    - aa. If the zoom flag is off
      - Get the coordinates for the soil map from the coordinate data base and set the geographic region
    - bb. Clear the graphics monitor
    - cc. Call procedure **pre\_land**
    - dd. Call GRASS command **p.map**



- ee. Set sim to 0
      - ff. Set flag to in
    - g. If sim is quit
      - Break
    - h. If sim is greater than the number of simulation times
      - Set sim to 0
  - Otherwise
    - Set sim to 1
- B. If sim is quit
  - Break
- C. If the zoom flag is off
  - a. Get the geographic coordinates from the coordinate data base and set the region
  - b. Clear the graphics monitor
- D. Based on the number selected by the user for simulation number sim, copy the relative, split output file into file outfile
- E. Set variable choice to 0
- F. While choice is smaller than 1 or choice is greater than the number of data specified by the user for the output depth plus the output travel time
  - a. Display the dialogue box to let the user pick up a data from the specified data
    - Read in the user's selection into variable choice
  - b. If the number of simulation is 1 and the user selects to quit
    - aa. Set variable no\_line to 0
    - bb. Break
- G. If no\_line is 1 and the first field is depth
  - a. Set variable result to depth
  - b. Pick up column 2 and another column which depends on the value of choice from the file outfile, store them in file depth
- H. If no\_line is 1 and the first field is time
  - a. Set variable result to time
  - b. Pick up column 2 and another column which depends on the value of choice from the file outfile, store them in file time
- I. If no\_line is 2
  - Generate file part2 from file outfile by copying simulation number column, system index column, and the columns for specified travel times
- J. If no\_line is 2 and choice is not greater than the number of depths specified by the user
  - Pick out column 2 and another column

- which depends on the value of choice
- K. If no\_line is 2 and choice is greater than the number of specified depths
    - a. Set variable result to time
    - b. Get choice2 by subtract the number of specified depths from choice
    - c. Pick up column 1 and another column which depends on the value of choice2 from file part2
  - L. If choice is quit
    - Break
  - M. Set sim to 0
  - N. If result is depth
    - Group the output of specified travel time and save them in file rule
    - Otherwise
      - Group the output of specified depth and save them in file rule
  - O. If file rule exist
    - a. Call GRASS procedure r.reclass
    - b. Call procedure pre\_pmap and redirect the output to file pmap\_file
    - c. Call GRASS procedure p.select
    - d. Call GRASS procedure p.map
  - P. Remove all the temporary files
19. End of draw\_map

#### Procedure Pre\_pmap

1. Set variable tf to /tmp/\$\$ for naming temporary files
2. Alias rm to rm -f for forcing removing files
3. Set nonomatch
4. Write "raster rastfile"
5. If result is time
  - Write "setcolor 1 gray"
  - Write "setcolor 2 green"
  - Write "setcolor 3 aqua"
  - Write "setcolor 4 yellow"
  - Write "setcolor 5 magenta"
  - Write "setcolor 6 red"
- Otherwise
  - Write "setcolor 1 gray"
  - Write "setcolor 2 red"
  - Write "setcolor 3 magenta"
  - Write "setcolor 4 yellow"
  - Write "setcolor 5 aqua"
  - Write "setcolor 6 green"
6. Write "colormode approximate"
7. Write "scale 1:2200000"
8. Write "vector [county].outline" ([county] represent the selected county name)
  - Write "color black"

- ```

Write "width 1"
Write "end"
9.  Get the map title east and north coordinates from the
    coordinate data base
    Write "text $east $north"
    If result is depth
        Write "TRAVEL TIME"
    otherwise
        Write "DEPTH"
        Write "color black"
        Write "width 2"
        Write "ref upper center"
        Write "size 3500"
        Write "border none"
        Write "end"
10. Using UNIX™ command tr to convert all the letters of
    the county name to upper-case
    Get the county name position (north and east
        coordinates) from the coordinate data base
    Write "text $east $north [county]"
    Write "color black"
    Write "width 2"
    Write "ref upper left"
    Write "size 1500"
    Write "border none"
    Write "end"
11. Using UNIX™ command tr to convert all the letters of
    the chemical name to upper-case
    Get the chemical name position (north and east
        coordinates) from the coordinate data base
    Write "text $east $north [chemical]" ([chemical]
        represents the name of the chemical)
    Write "color black"
    Write "width 2"
    Write "ref upper left"
    Write "size 1500"
    Write "border none"
    Write "end"
12. Get the travel time or depth title position (north and
    east coordinates) from the coordinate data base
    Write "text $east $north"
    If result is depth
        Write "DEPTH value"
    Otherwise
        Write "TRAVEL TIME value"
    Write "color black"
    Write "width 2"
    Write "ref upper left"
    Write "size 1500"
    Write "border none"
    Write "end"
13. Get the beginning position of the legend box (north,
east1 from the coordinate data base
    Write "point $east1 $north"

```

- ```

Write "color gray"
Write "width 2"
Write "ref center left"
Write "size 1500"
Write "border none"
Write "end"
14. Get east2 by adding 5000 to east1
Write "text $east2 $north nodata"
Write "color black"
Write "width 2"
Write "ref center left"
Write "size 1500"
Write "border none"
Write "end"
15. Subtract north by 400
Write "point $east1 $north"
If result is depth
    Write "color red"
Otherwise
    Write "color green"
Write "icon box.fill"
Write "size 2.0"
Write "masked n"
Write "end"
16. Write "text $east2 $north"
If result is depth
    Write "< 1 yr."
Otherwise
    Write "< 0.5 m"
Write "color black"
Write "width 2"
Write "ref center left"
Write "size 1500"
Write "border none"
Write "end"
17. Deduct 400 form north
Write "point $east1 $north"
If result is depth
    Write "color magenta"
Otherwise
    Write "color aqua"
Write "icon box.fill"
Write "size 2.0"
Write "masked n"
Write "end"
18. Write "text $east2 $north"
If result is depth
    Write "< 1 - 2 yr."
Otherwise
    Write "< 0.5 - 1 m"
Write "color black"
Write "width 2"
Write "ref center left"
Write "size 1500"

```

```

Write "border none"
Write "end"
19. Deduct 400 from north
Write "point $east1 $north"
Write "color yellow"
Write "icon box.fill"
Write "size 2.0"
Write "masked n"
Write "end"
20. Write "text $east2 $north"
If result is depth
    Write "< 2 - 5 yr."
Otherwise
    Write "< 1 - 2 m"
Write "color black"
Write "width 2"
Write "ref center left"
Write "size 1500"
Write "border none"
Write "end"
21. Deduct 400 from north
Write "point $east1 $north"
If result is depth
    Write "color aqua"
Otherwise
    Write "color magenta"
Write "icon box.fill"
Write "size 2.0"
Write "masked n"
Write "end"
22. Write "text $east2 $north"
If result is depth
    Write "< 5 - 10 yr."
Otherwise
    Write "< 2 - 4 m"
Write "color black"
Write "width 2"
Write "ref center left"
Write "size 1500"
Write "border none"
Write "end"
23. Deduct 400 from north
Write "point $east1 $north"
If result is depth
    Write "color green"
Otherwise
    Write "color red"
Write "icon box.fill"
Write "size 2.0"
Write "masked n"
Write "end"
24. Write "text $east2 $north"
If result is depth
    Write "> 10 yr."

```

```

Otherwise
    Write "> 4 m"
Write "color black"
Write "width 2"
Write "ref center left"
Write "size 1500"
Write "border none"
Write "end"

```

25. End of **pre\_pmap**

#### Procedure **pre\_soils**

1. Alias **rm** to **rm -f** for forcing removing files
2. Set variable **tf** to **/tmp/\$\$** for later naming temporary files
3. Write "raster [county].soils"
4. Write "colormode approximate"
5. Write "scale 1:22000000"
6. Using the Oklahoma county name data base to find the short form of a selected county name
7. Get the soil map title position (north, east) from the coordinate data base
 

```

Write "text $east $north [county] SOILS"
Write "color white"
Write "width 2"
Write "ref upper center"

```

 Get the font size from the coordinate data base and store it in variable **size**

```

Write "size $size"
Write "border none"
Write "end"

```
8. End of **pre\_soils**

#### Procedure **pre\_land**

1. Alias **rm** to **rm -f** for forcing removing files
2. Set variable **tf** to **/tmp/\$\$** for later naming temporary files
3. Write "raster [county].landuse"
4. Write "colormode approximate"
5. Write "scale 1:22000000"
6. Using the Oklahoma county name data base to find the short form of a selected county name
7. Get the land use map title position (north, east) from the coordinate data base
 

```

Write "text $east $north [county] LAND USE"
Write "color white"
Write "width 2"
Write "ref upper center"

```

 Get the font size from the coordinate data base and store it in variable **size**

```

Write "size $size"

```

```
Write "border none"  
Write "end"  
8. End of pre_land
```

APPENDIX C  
INSTALLATION INSTRUCTIONS OF  
THE MAP DRAWING TOOL



## INSTALLATION INSTRUCTIONS OF THE MAP DRAWING TOOL

Include the directory name where all the shell procedures are stored in your path variable. This can be done in one of the two following ways (assume dir\_name is the directory name where shell procedures are stored):

1. Type set path = (dir\_name \$path) under the C shell.
2. Add the dir\_name to the path variable directly in the .cshrc file via any text editor. Save the modified file. Type source .cshrc in the command line of your home directory to recompile the .cshrc file.

VITA 2

FENGXIA MA

Candidate for the Degree of  
Master of Science

Thesis: USING THE UNIX™ SHELL TO INTEGRATE A MANAGEMENT  
MODEL WITH A GIS

Major Field: Computer Science

Biographical:

Personal Data: Born in Lanzhou, People's Republic of  
China, January 22, 1964, The daughter of Mr.  
Yiting Ma and Ms. Huanzhen Li.

Education: Graduated from Northwest Teachers  
University high school, Lanzhou, P.R. China, in  
August 1981; received Bachelor of Science Degree  
in Cartography from Wuhan Technical University of  
Surveying and Mapping, Wuhan, P.R. China in July,  
1985; completed requirements for the Master of  
Science degree at Oklahoma State University in  
December, 1993.

Professional Experience: Student programmer,  
Department of Agronomy, Oklahoma State University,  
January, 1992 to Present. Instructor, Cartography  
Department of Wuhan Technical University of  
Surveying and Mapping, Wuhan, P.R. China, July,  
1985 to February, 1989.

Professional Societies: Student Member, Association  
for Computing Machinery