3D Structured Spreadsheet

by

Qian Wu
B.Sc, Changsha Institute of Technology

A Thesis Submitted in Partial Fulfillment of the
Requirement for the degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

[redacted]
Dr. William W.Wadge, Supervisor (Department of Computer Science)

[redacted]
Dr. Michael Levy, Departmental Member (Department of Computer Science)

[redacted]
Dr. Michael Fellows, Departmental Member (Department of Computer Science)

[redacted]
Dr. Nikitas J. Dimopoulos, Outside Member (Dept. of Electrical and Computer Eng.)

University of Victoria

Supervisor: Professor William W.Wadge

ABSTRACT

The 3D Structured Spreadsheet (3DSS) is a new kind of spreadsheet that uses program organization concepts similar to those of block-structured languages.

A program developed in a block-structured language such as PASCAL contains a main unit and optionally many functions and procedures. The major unit such as blocks, procedures and functions can be recursively defined. References to variable names follow scoping rules. In the same way, a 3DSS program can contain many sheets: one **main** sheet, and optionally many **subsheets**.A cell in a sheet can contain different kinds of values: text, number, formula, or a subsheet or a group of timesheets that vary with "time". Every cell can be given a name, and the reference relationships among the cells of parent sheets and subsheets follow the block-structured scoping mechanism. Sheets in a 3D Structured Spreadsheet program are therefore organized into a tree structure. Any sheet can also vary in "time" (3D) using time formulas to construct a group of time frames-- **timesheets**. 3D Structured Spreadsheet combines subsheet and timesheet together, and can organize information into a complex dynamic hierarchical way. In addition, one can define functions and procedures using subsheets and timesheets through programming-by-example in the same spreadsheet programming paradigm

The new spreadsheet design is based on ideas used by conventional spreadsheets, but it has significant advantages over the conventional ones. The hierarchical organization mechanism allows top-down design. The 3D feature provides a more powerful and convenient method for organizing information that varies in "time", and a more powerful calculation ability for iterative calculations. User-defined functions or procedures through programming-by-example in the same spreadsheet programming paradigm extends the spreadsheet programming environment and provides an abstract way to organize blocks of information having similar structure.

Examiners:

[redacted]

Dr. William W.Wadge, Supervisor (Department of Computer Science)

[redacted]

Dr. Michael Levy, Departmental Member (Department of Computer Science)

[redacted]

Dr. Michael Fellows, Departmental Member (Department of Computer Science)

[redacted]

Dr. Nikitas J. Dimopoulos, Outside Member (Dept. of Electrical and Computer Eng.)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I am indebted to my supervisor, William W. Wadge, for suggesting the topic of this thesis and for his valuable guidance.

I would also like to thank my original supervisor, Bjorn Freeman-Benson, who offered me many important comments.

Panagiotis Rondogiannis and bryan Gilbert corrected my English and offered substantial remarks, and I am grateful.

Joerge Dyrkton gave me good English lessons. And my family in China gave me encouragement, always. I thank them too.

CHAPTER 1

INTRODUCTION

This thesis describes the design and implementation of a new kind of spreadsheet--the 3D Structured Spreadsheet.

Originally, spreadsheet programs were electronic replacements for the traditional financial modeling tools: the account's columnar, pad, pencil and calculator. The characteristics of the spreadsheet make it a powerful problem-solving tool for many applications.The first electronic spreadsheet was developed in 1978 by Roberston and Dan Brickly in Cambridge, Massachusette[1]. It defined the main concepts of spreadsheets. A spreadsheet is organized as a grid of columns and rows. Each intersection of a row and column is an entry position (cell) where the user can enter information. All the values are visible and the user's operations are interactive and local[1][6]. The spreadsheet is continuously being evaluated as the user is modifying the content of a cell, so that the computation is always up-to-date ("what if...?").

From VisiCalc to Lotus 1-2-3, spreadsheets have changed significantly by providing more powerful calculating capabilities, more powerful functions, integrated business graphics, word processing and data-base management[1]. However, when we study a conventional spreadsheet application, we find that there has been no significant changes in the structure

1. In spreadsheets the formula in a cell can only access other cells but can not modify the values of other cells. This is the local feature of spreadsheets.

of spreadsheet programs.

Sheets in a spreadsheet program could be treated as blocks or functions or procedures, since they are the major units of a spreadsheet program. Some spreadsheet applications need to establish relationships among different sheets. The first generation spreadsheet used only a large two dimensional sheet for an application[1]. Relationships among sheets could not be established at all. In current spreadsheet products, one can use links to establish relationships among different sheets, and organize information into a hierarchy for a relatively complex application. However, these sheets are like isolated blocks in a large conventional program which are connected by "goto" statements. In fact, we have many years of successful research and application in structured imperative languages, for example, PASCAL. Unfortunately, the benefits of the experience do not appear in spreadsheet organization and programming.

Conventional spreadsheets are static in "time". They can not form "time" frames that would allow a cell in one of these frames to dynamically reference a value in a previous "time" frame. Many applications not only involve a relationship among static 2-D sheets, but also involve a relationship that changes with "time". For example, in one store, the prices of fruit items are increased by 5% every month. We want to look at all the prices in different months in one year from Jan. to Dec. [2]. In this case, the prices from Feb. to Dec. will each depend on the prices in the previous month. We call this kind of relationship a **time relationship**. We believe that time relationships are useful to help organize information which varies with "time". They also help solve some scientific problems, as many applications in nature vary in time as well as space, and many problems use iterative technique.

The tremendous successes of spreadsheet programs are due to the fact that the matrix for-

mat leads to a natural representation of many problems and that spreadsheet programming is an end-user programming approach which abolishes the traditional distinction between program design and program testing. However, conventional spreadsheets do not provide a complete programming environment, mainly because they do not consider that functions or procedures are abstract units of spreadsheets[8]. They also do not allow the user to define functions or procedures in the same spreadsheet programming paradigm. If one wants to define a function or procedure, one has to use a Macro language, which is very difficult to use and debug, and it is a back step to traditional programming.

We introduce the 3D Structured Spreadsheet (the following we call 3D Structured Spreadsheet 3DSS) which provides a new program structure. It uses program organization concepts similar to those of block-structured languages. A program developed in a block-structured language such as PASCAL contains a main unit and optionally many functions and procedures. The major unit such as blocks, procedures and functions can be recursively defined. References to variable names follow scoping rules. A spreadsheet program can contain many sheets: one **mainsheet**, and optionally many **subsheets**. A cell in a sheet can contain different kinds of values: text, number, formulas, or a subsheet or a group of timesheets. Every cell can be given a name, and the reference relationships among the cells of parent sheets and subsheets follow the block-structured scoping mechanism.

A 3DSS sheet can also vary with "time" (3D) [1]using time formulas to construct a group of time frames-- **timesheets**. The tree structure of a 3DSS program can be changed by controlling the number of timesheets. So 3DSS can organize information into a complex dynamic hierarchy.

---

1. This is different from the current view of 3D sheets. For example, Lotus 1-2-3 for Windows is said to provide three dimensional worksheets. That 3D concept is static which takes advantage of Windows. Now the 3D varies with "time"[3], the system generates timesheets according time relationships defined by the user..

In 3DSS, one can define functions and procedures using subsheets and timesheets through programming-by-example in the spreadsheet paradigm. A user-defined function or procedure is a group of sheets which has the same tree structure as that of a 3DSS program.

In summary, the new spreadsheet design is based on ideas used by conventional spreadsheets, but it has significant advantages over the conventional ones. The hierarchical organization mechanism allows top-down design. The 3D feature provides a more powerful and convenient method for organizing information that varies with "time", and a more powerful calculation ability for iterative and nested iterative calculations. User-defined functions or procedures through programming-by-example in the same spreadsheet programming paradigm extends spreadsheet programming environment and provides an abstract way to organize blocks of information having similar structure.

In the next chapter, we give some necessary background on spreadsheets and block-structured language concepts. Chapter 3 describes the 3DSS features, programming and comparison between conventional spreadsheets and 3DSS through some examples. In chapter 4, the current implementation of the new spreadsheet is described. The last chapter gives the conclusions and further work.

CHAPTER 2

BACKGROUND

This chapter introduces the basic concepts of spreadsheet and program organization of block-structured languages that are used in 3DSS.

2.1 Spreadsheets

2.1.1 Overview

In general, a spreadsheet allows the user to solve problems involving numbers. It is organized into columns and rows like an accountant's spreadsheet, except that it is much larger. At each intersection of a row and a column are entry positions or cells into which the user can enter formulas. The formula causes the program to display the formula's value and recalculate the formula automatically whenever any cell on which it depends is changed so that the values of the cells are always up-dated and displayed. The characteristics of the spreadsheets make them become powerful problem-solving tools not only for business applications, but for scientific applications as well.

The screen for a typical spreadsheet usually consists of two basic areas: the control panel and the sheets [1]. Each sheet is organized into many cells by columns and rows. The num-

1. Lotus 1-2-3 for windows can display many worksheets at one time.

ber of the cells depends on different spreadsheet systems. The control panel provides facilities that let the user operate on the sheets. It includes a command menu (menu bar or icon palette), an input area to input commands and formulas, and a prompt area to display various interactive messages for system status or error information. Through the user interface of a spreadsheet, users can manipulate the sheets. Although various spreadsheets have different programming styles and different mahipulation facilities, they commonly provide some similar basic commands such as the *file* commands, the *edit* commands, the *range* commands, *the graph* commands, the *Data* commands, the *Style* commands etc. [3].

Into any cell, one can put a number, text, or a formula[1]. A formula is usually an arithmetic expression or a logical expression. The arithmetical expression is built up using numbers, references to other cells, arithmetic operators, and built-in functions. The logical expression is true/false tests. Mathematical relationships can be created between cells through formulas. A cell can be given a name. The reference to a cell in a formula can be a name, or an absolute address, or a relative address, or a combined absolute and relative address. An absolute address always points to the same cell. The relative address points to a cell related to the cell containing the formula. Either column or row address can be absolute or relative for a combined absolute and relative reference [3]. For example, absolute address R[1]C[1] always refers to the intersection cell at row one and column one. Relative address R[-1]C[-1][2] in cell R[2]C[2]'s formula refers to cell R[1]C[1]. The combined absolute and relative address R[1]C[-1] in cell R[2]C[2] refers to cell R[1]C[1]. Relative address is very useful for the copy operation[3]. A name address is always an absolute address. It is the formula's reference ability that allows spreadsheets to be continuously evaluated, so that the computation is always up-to-date. Table 2.1.1.1 (a)shows the calculation result of one's principal, interest, and bank balance within three months. Table

1. In fact, text and number are special case of formula, so later formula indicates all kinds of formulas including text and number.
2. In 3DSS, we use + or - to represent relative reference.

2.1.1.1(b) gives the formulas for each cell. From table 2.1.1.1(b), we can see the dependencies among the cells. Afterwards, if the bank interest is changed from 1% to 0.9%, the system will recalculate all the dependent cells and redisplay them again. Table 2.1.1.1(c) gives the result after the modification and recalculation.

	0	1	2	3	4	5
0						
1	principal	1000$				
2	rate	1%				
3	withdraw	100$				
4				JUNE	JULY	AUGUST
5			principal	1000$	990$	979$
6			withdraw	100$	100$	100$
7			interest	90$	89$	87.9$
8			balance	990$	979$	966.9$
9			total interest	266.90$		

Table 2.1.1.1(a). the spreadsheet for the calculation of interest

.. .	1	2	3	4	5
0					
1	1000				
2	0.1				
3	100				
4					
5			R[1]C[1]	R[8]C[3]	R[8]C[4]
6			R[3]C[1]	R[3]C[1]	R[3]C[1]
7			(R]5C[+0] - R]6C[+0]) * R[2]C[1]	(R[5]C[+0] - R[6]C[+0])+R[2]C[1]	(R[5]C[+0] - R[6]C[+0]) * R[2]C[1]
8			R[5]C[+0] - R[6]C[+0] + R7C[+0]	R[5]C[+0] - R[6]C[+0] + R[7]C[+0]	R[5]C[+0] - R[6]C[+0] + R[7]C[+0]
9			R[7]C[3] + R[7]C[4] + R[7]C[5]		

Table 2.1.1.1(b). the formulas for the calculation of interest

	0	1	2	3	4	5
0						
1	principal	1000$				
2	rate	0.9%				
3	withdraw	100$				
4				JUNE	JULY	AUGUST
5			principal	1000$	981$	960.29$
6			withdraw	100$	100$	100$
7			interest	81$	79.29$	77.43$
8			balance	981$	960.29$	937.72$
9			total interest	237.72$		
10						

Table 2.1.1.1(c). the recalculated spreadsheet for the calculation of interest

2.1.2 Recalculation

The recalculation of the values of spreadsheet cells occurs every time a cell is defined or modified. The value of the formula of a cell that references other cells depends on these referenced cell values. These dependencies among the cells define a **calculation natural order** of a spreadsheet, which defines the calculation sequence of cells in a sheet.

Set A_1, A_2..... A_n be sets of cells in a sheet (the number of cells in each set is equal to or greater than one). If cells in A_i reference the cells in A_{i-1}, and cells in A_1 reference no cells, then A_1, A_2..... A_n constitutes a calculation sequence. Cells in A_1 are the most fundamental cells which have the lowest calculation order. Cells in A_2 are the second most fundamental cells which have the second lower calculation order, and so on. The calculation of cells that have lower calculation order should occur before the calculation of cells

that have higher calculation order. If the formula of a cell references other cell values, but its calculation natural order is less than these cell calculation natural order; the reference is a **forward reference.** If two or more cells are defined by each other directly or indirectly, this reference is a **circular reference**.

There are three ways to do the recalculation. The simple way to do the recalculation is to use a linear form of recalculation. The model starts at the upper left corner of a sheet and proceeds either row by row or column by column through the sheet. The user must be careful to build dependencies in the sheet to avoid forward references and circular references that can create calculation nightmares.

Another method is to use the calculation natural order of cells in a sheet. The model starts from the most fundamental cells in the sheet, and then evaluates the second most fundamental cells in the sheet. This process continues until the whole sheet is recomputed. For example, in table 2.1.1.1(b), the most fundamental cells are: R[1]C[1], R[2]C[1] and R[3]C[1]. The recalculation sequence is: {R[1]C[1], R[2]C[1], R[3]C[1]}, {R[5]C[3], R[6]C[3], R[6]C[4], R[6]C[5]}, {R[7]C[3]}, {R[8]C[3]}, {R[5]C[4]}, {R[7]C[4]}, {R[8]C[4]}, {R[5]C[5]}, {R[7]C[5]}, {R[8]C[5]}, {R[9]C[3]}. Figure 2.1.2.1 shows the dependencies of these cells. The cells in the same set can be recomputed in any order. This model provides more powerful calculation ability: the user does not have to be careful about the calculation order of the cells, and the user can use circular reference and iteration functions to do iteration[3].

Another model also uses calculation natural order, but only calculates the modified cell and the cells directly or indirectly depending on this cell. For example, in table 2.1.1.1(b), if cell R[2]C[1] is modified to 0.9%, this model will only recalculate cells R[2]C[1], R[7]C[3], R[8]C[3], R[5]C[4], R[7]C[4], R[8]C[4], R[5]C[5], R[7]C[5], R[8]C[5],

R[9]C[3] according to their calculation natural order, and will not recalculate cells R[1]C[1], R[3]C[1], R[5]C[3], R[6]C[3], R[6]C[4], R[6]C[5]. It is clear that this model is the most efficient one.

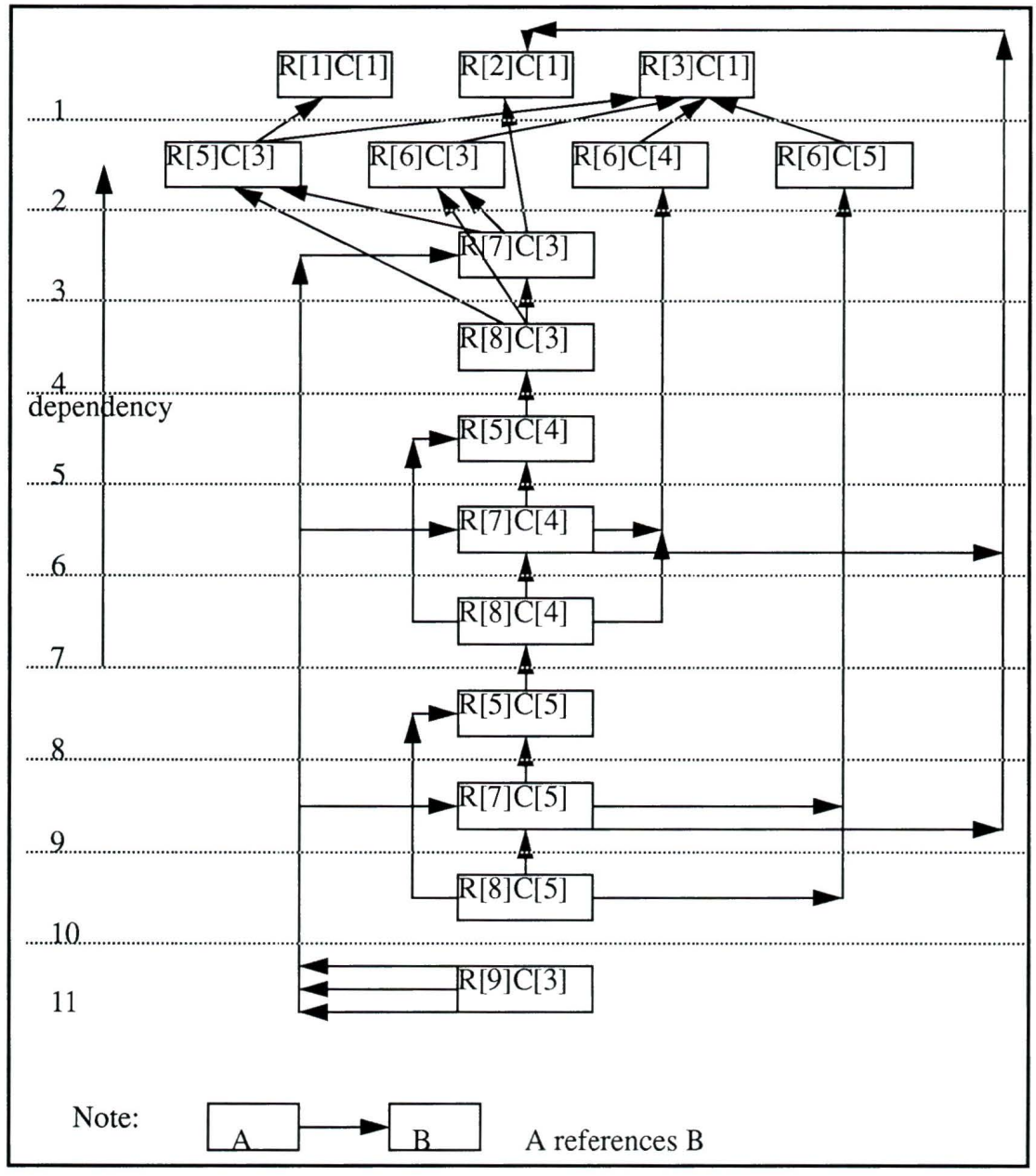

Figure 2.1.2.1 the cell dependencies in the calculation of interest

2.1.3 Links among Sheets

In the first generation spreadsheet, the user had to put everything into a large sheet for an application. Sometimes one needs only one sheet to analyze and store data. At other times, one may need to use multiple sheets to most effectively organize data. In a program which contains many sheets, a sheet is the major unit. For example, one application contains data for 12 months. It is natural to divide the data into 13 sheets. 12 sheets contain data for an entire year, the 13th worksheet represents the summation of the individual sheets. So a link facility is added to allow the user to create links among sheets. In a conventional spreadsheet, sheets are viewed as independent separate units. They are equal when they are created. The relationships are added later by the user.

Spreadsheets provide different facilities for creating links among sheets. For example, in Lotus 1-2-3 for Windows, the user can use the mouse to point to the referenced cell in the source sheet, then point to the reference cell in the destination sheet, or use a symbol address such as file name plus cell address directly[3]. There are two kinds of links among sheets, one is the links among the sheets in different files, the other is the links among the sheets in the same files. The first one is very useful when two or more applications share data, for example, two applications share data through a network; or when an application is too large to fit into memory at one time. It is more difficult to create and maintain than the second one. For the second one, it is useful for organizing information in one application in one model[1]. In 3DSS, we only consider the second one, which all the information can be organized into one file.

---

1. In Lotus 1-2-3 for windows, the user can create many worksheets in one file. These sheets are named by a sequence of characters, for example A, B, C.....Z. A cell in one sheet can reference any cells in other sheets.

2.2 Block-structured Languages

2.2.1 Program Structure

Usually a program developed in a block-structured language such as PASCAL using structure design methods contains a main unit and many functions and procedures. For a large program, functions and procedures can be organized into an even bigger unit or what is called module. The major units such as blocks, procedures and functions can be recursively defined.

A **procedure** is a unit or subprogram that changes the program state, either by changing the state of one or more data objects or by changing the state of the display screen. In general, to define a procedure one should provide: the name of the procedure and formal parameters (input data and/or output data). A **function** is a unit or subprogram that returns a single data value to its caller. In general, to define a function, one should provide: the name of the function, formal parameters (input data), and the data type of the returned value. A **block** is a unit that contains more than one statements.

2.2.2 Name Scope

It is important that the names introduced by a programmer to refer to data or subprograms should not necessarily apply, with the same meaning, over the whole program. An important technique for achieving this is known as block structure, originally introduced with ALGOL 60. The rules for scope of names in block-structured languages are as follows[4]:

1. The scope of a name includes the block, function or procedure in which it is declared but not the block, function or procedure surrounding it.

2. The scope of a given name includes any block, function or procedure contained within

its associated block, function or procedure. However, if the same name is used in a declaration within such a block, function or procedure, a new meaning for the name is thereby introduced.

In block-structured languages, blocks, functions or procedures can be recursively defined. The following gives an example of nested procedure definition in PASCAL:

```
program Nested (Input, Output)
var
   I, J, K : integer;
   procedure P1()
   var
      J, L, M : integer;
      procedure P2()
      var
         I, P : integer;
         begin {P2}
         ...
         end; {P2}
         procedure P3()
          var
            J, P, Q : integer;
            begin {P3}
            ...
            end; {P3}
      begin {P1}
      ...
      end; {P1}
   begin {Nested}
   ...
   end.
```

Figure 2.2.2.1 gives the contour diagram for the above definition. A **contour diagram** is a

set of nested boxes symbolizing the various subprograms (and main program) or blocks. It is a technique used to aid in the understanding of the scope of variables[16]. In this example, procedures P2 and P3 are defined within procedure P1, and P1 is defined within the main program. Within P1, J, L, M refer to the variables defined within P1, and I and K refer to the global variables defined in the main program. Within P2, I, P refer to the local variables defined within P2, J, L, M refer to the global variables defined within the surrounding P1 contour; and K refers to the global variable of the main program. Within P3, J, P, Q refer to the local variables defined within P3; M refers to the global variable defined within the surrounding P1 contour; and I, K refer to the global variables defined in the main program. From another perspective, I in the main subprogram, the scope consists of the main program, the body of P1, and the body of P3. Within P2, any references to I refers to the local variable. J in main, the scope consists of the main program. Within P1 or P2, any reference to J refers to the J of P1; within P3, it refers to the J of P3. The scope of K is the entire program. J of P1, the scope consists of P1 and P2. Variables J in P3 hides the variables of the same names from the P1 procedure. I in P2 hides the variable I of the main program.

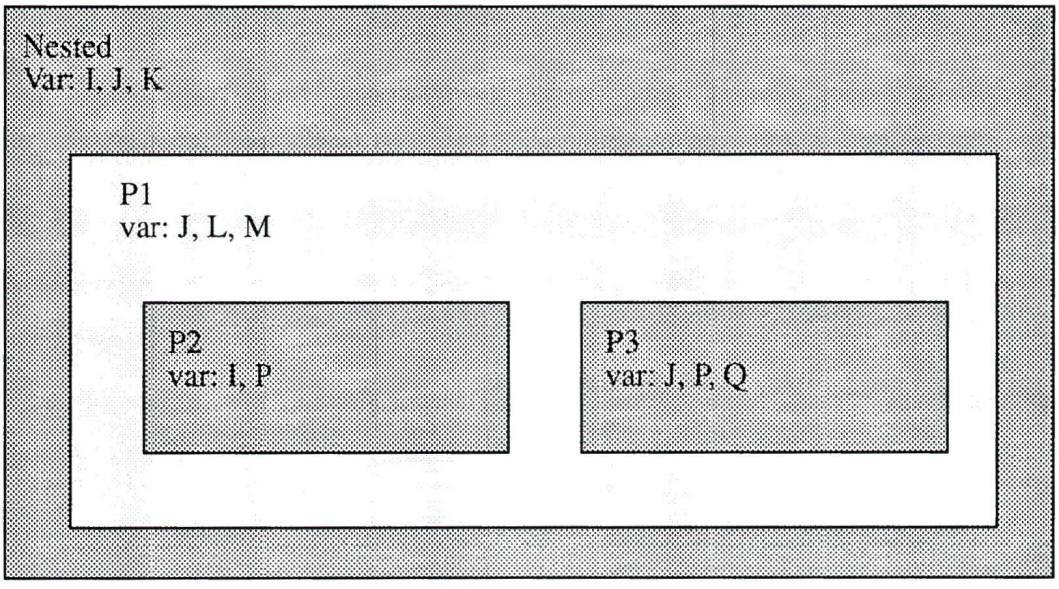

Figure 2.2.2.1. the contour diagram for the nested procedure definition

The next chapter will describe the features of 3DSS programming and comparison between conventional spreadsheets and 3DSS through some examples.

CHAPTER 3

3D STRUCTURED SPREADSHEET

This chapter illustrates the features of 3D Structured Spreadsheet (3DSS) with examples, and it also gives some comparison between conventional spreadsheets and 3DSS.

3.1 Subsheets

A 3DSS program can contain many sheets, including a mainsheet and optionally many subsheets which are organized into a tree. Similar to recursively defined blocks, functions, or procedures, a cell can contain a number, text, a formula and a subsheet, and sheets can be nested. The cell scope among the sheets follows the scoping rules of block-structured languages. The introduction of subsheet:

.helps the user to organize information into a top-down hierarchy.

.allows one to use the same names in different sheets without fear of name clashes.

.gives the spreadsheet programmer an opportunity to organize a spreadsheet application's user interface.

3.1.1 Mainsheets, Subsheets and Supersheets

In the 3DSS, the nested block concept is used for both cell reference and sheet organization. A 3DSS program contains a mainsheet and optionally many subsheets. The major

unit -- sheet can be nested, just like nested defined blocks. Any cell can contain a formula or another sheet or another group of timesheets[1].The sheets of a 3DSS program are organized into a tree. The root of the tree is called **mainsheet**, which is similar to the main routine in a PASCAL program. Any other sheet is called a **subsheet**. Any sheet which is directly or indirectly a parent of a subsheet is called a **supersheet** of the subsheet. The children of a sheet are called **sibling sheets**. A subsheet is similar to a conventional spreadsheet except that it is not independent: it relates to its supersheets; its cell references follow block scoping rules[2]. See figure 3.1.1.2.

.

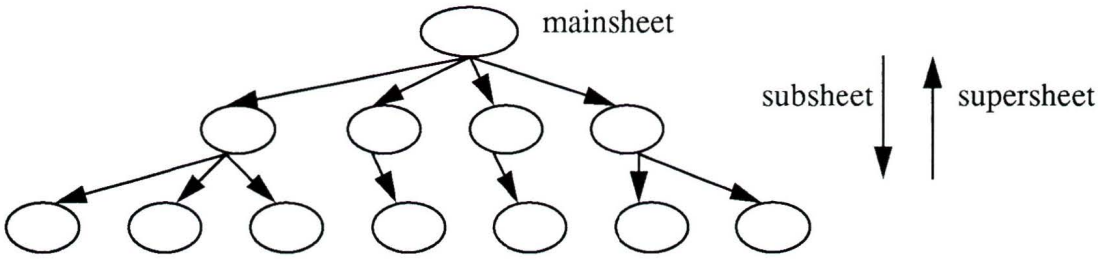

Figure 3.1.1.2 a hierarchical structure of mainsheet, subsheet and supersheet

At any time the user can select a cell that contains a subsheet and open it. At any time the user can close an open sheet. The system will close all the subsheets of a sheet when the user closes the sheet. The user can operate in any open sheet. The system automatically updates all related values in all the sheets which are affected by the user's operations in an open sheet.

3.1.2 Cell Scope

In 3DSS, the root sheet name is always **main** which is given by the system, and for every application it is the same, like the *main* routine in C program. Any subsheet has its own

1. This will be discussed in detail in section 3.2
2. This will be discussed in the next section.

name which is given by the user when it is created. Any cell can also be given a name. In 3DSS a cell **reference address** can be a name that is given to the cell, or an absolute address, or a relative address, or a combined absolute and relative address. When a cell references other cells in the same sheet, it can use any reference address. When a cell references cells in its supersheets, it can use a cell name, or a supersheet name plus any reference address; or @ plus any reference address[1]. @ refers to the direct supersheet, similar to **super** or :: in smalltalk or C++.

The cell reference obeys the following rules:

1. a cell can only reference the cells in the same sheet, or cells in the supersheets. In other words, supersheets are visible to their subsheets; a cell in a supersheet can not reference cells in its subsheets; cells in sibling sheets can not be directly referenced.
2. when a cell references other cells using their names, it follows the name scoping rules in block-structured languages.
3. a subsheet can use the **return** statement to return a value to its direct supersheet[2].

Table 3.1.2.1(a) ~ (d) shows the example given in chapter 2 section 2.2.2 reprogrammed using subsheets and cell scoping rules. In the mainsheet, cell names i, j, k are defined, and subsheet P1 is defined within the mainsheet. In subsheet P1, cell names j, l, m are defined, and subsheets P1 and P2 are defined within P1. In P2, cell names i, p are defined. In P3, cell names j, p, q are defined. Follows the rules described above, In P1, i, k reference the values in the mainsheet, and j, l, m reference the values in P1 itself. In P2, k references the value in the mainsheet, j, l, m reference the values in P1, and i, p reference the values in P2 itself. In P3, i, k reference the values in the mainsheet, l, m reference the values in P1, and j, p, q reference the values in P3 itself.

---

1. See appendix B for more information
2. This will be discussed in the next section.

	0	1	2	3	4	5	6	7	8	9
0										
1										
2				variable	I	J	K			
3					1.00	2.00	3.00			
4				subsheet	P1					
5										
6										

Table 3.1.2.1(a). the mainsheet

	0	1	2	3	4	5	6	7	8	9
0										
1										
2				variable	J	L	M			
3					4.00	5.00	6.00			
4				reference	I	J	K	L	M	
5					1.00	4.00	3.00	5.00	6.00	
6				subsheet	P2	P3				

Table 3.1.2.1(b). P1 subsheet

	0	1	2	3	4	5	6	7	8	9
0										
1										
2				variable	I	P				
3					7.00	8.00				
4				reference	I	J	K	L	M	P
5					7.00	4.00	3.00	5.00	6.00	8.00
6										

Table 3.1.2.1(c). P2 subsheet

	0	1	2	3	4	5	6	7	8	9
0										
1										
2			variable	J	P	Q				
3				9.00	10.00	11.00				
4			reference	I	J	K	L	M	P	Q
5				1.00	9.00	3.00	5.00	6.00	10.00	11.00
6										
7										

Table 3.1.2.1(d). P3 subsheet

3.1.3 Return Statement

In block-structured languages, the name scoping rules allow a lower level unit to use the variables in a higher level unit, and to modify the variables defined in a higher level unit in a lower level unit. The data can "flow" from top to bottom, and from bottom to top. However, one of the features of a spreadsheet program is that a cell can only draw values from other cells, it can not modify other cell values. If we apply the name scoping rules strictly to 3DSS, the data will only "flow" from top to bottom, and there is no way to "flow" from bottom to top. In fact, the user often does a subcomputation in one sheet, and wants the final result of the subcomputation in another sheet.

3DSS provides the **return** statement which allows data to "flow" from bottom to top. The **return** statement returns the value of the cell that contains the statement to the cell that points to the subsheet. A subsheet can only contain one return statement. For example, let's say cell A contains a return statement, and cell B points to the subsheet, then cell B's value will be cell A's value. If a subsheet does not contain a return statement, the cell that

contains the subsheet will display the subsheet name.

Return statements also provide a way for sibling sheets to share data. For a hierarchical design, sharing data should be arranged in supersheets. So a subsheet returns a value to its supersheet which can be referenced by the subsheet's sibling sheets.

3.1.4 Examples Using Subsheet

The tree structure described above can make the user create a clear hierarchical design. For example, Mr. W. Talor wants to calculate his net worth in 1992 and 1993. The net worth calculation involves cash, convertible assets, personal possession, loan liabilities and unpaid bill. Each category has many subitems, involves subcomputation. Table 3.1.4.1~ 3.1.4.4 gives an example of using subsheets and return statement to do the net worth calculation. In the mainsheet of the program, cells R[3]C[1], R[3]C[2], R[3]C[4], R[3]C[5], R[4]C[1], R[4]C[2], R[4]C[3], R[4]C[4], and R[3]C[5] each contain a subsheet. These subsheets list the subitems, do the subcomputation and return the results to the mainsheet. Table 3.1.4.2 gives the open subsheet in cell R[3]C[1]. Table 3.1.4.3 and 3.1.4.4 give the formulas for the main sheet and subsheet for 1992 cash. cash1992, cash1993, con_ass1992, con_ass1993, per_pose1992, per_pose1993, loan_lia1992, loan_-lia1993, unpad1992, unpad1993, total_ass1992, total_ass1993, total_lia1992 and total__-lia1993 are the given names to the corresponding cells

.

	0	1	2	3	4	5	6	7	8
0	name:	W.Talor						time:	5 JAN
1					NET	WORTH			
2	year	cash	con-ass	per-pose	loan-lia	unpad-lia	total-ass	total-lia	networth
3	1992	7225.00	19800.00	60000.00	145850.0	1525.00	87025.00	147375.0	-60350.0
4	1993	8700.00	28900.00	68500.00	148300.0	1610.00	106100.0	149910.0	-43810.0

Table 3.1.4.1. the mainsheet for the net worth application

	0	1	2	3	4	5	6	7	8	9
0										
1										
2			cash	75.00						
3			saving	4300.00						
4			checking	2350.00						
5			bonds	400.00						
6			loans	100.00						
7			total	7225.00						

Table 3.1.4.2 the subsheet for 1992 cash in the net worth application

	1	2	3	4	5	6	7	8
0								
1								
2								
3	subsheet	subsheet	subsheet	subsheet	subsheet	cash1992 + con_ass1992 + per_-pose1992	loan_lia1992 + unpad_lia1992	total_ass1992 - total_-lia1992
4	subsheet	subsheet	subsheet	subsheet	subsheet	cash1993 + con_ass1993 + per_-pose1993	loan_lia1993 + unpad_lia1993	total_ass1993 - total_-lia1993

Table 3.1.4.3 the formulas for the mainsheet

	0	1	2	3	4	5	6	7
0								
1								
2				75.00				
3				4300.00				
4				2350.00				
5				400.00				
6				100.00				
7				return r[2]c[+0] + r[3]c[+0] + r[4]c[+0] + r[5]c[+0] + r[6]c[+0]				

Table 3.1.4.4 the formulas for 1992 cash subsheet

The system keeps all the records of the reference relations. Whenever a cell is changed, the system will calculate the necessary cells according the dependencies and update the cell values in all the sheets[1]. For example, if we change the first item 75 to 75000 in table 3.1.4.2, the system will recalculate the cells R[2]C[3],R[7]C[3] in subsheet 1992cash, and cells R[3]C[1], R[3]C[6] and R[3]C[8] in the mainsheet, and redisplay them again. The result is shown in table 3.1.4.5and table 3.1.4.6

	0	1	2	3	4	5	6	7	8
0	name:	W.Talor						time:	5 JAN
1					NET	WORTH			
2	year	cash	con-ass	per-pose	loan-lia	unpad-lia	total-ass	total-lia	networth
3	1992	82150.00	19800.00	60000.00	145850.0	1525.00	161950.0	147375.0	14575.00
4	1993	8700.00	28900.00	68500.00	148300.0	1610.00	106100.0	149910.0	-43810.0
5									
6									

Table 3.1.4.5. the up-to-date result for the net worth application

1. This will be discussed in chapter 4, section 4.3.2 reference chain

	0	1	2	3	4	5	6	7	8	9
0										
1										
2			cash	75.00						
3			saving	4300.00						
4			checking	2350.00						
5			bonds	400.00						
6			loans	100.00						
7			total	7225.00						

Table 3.1.4.6 the up-to-date result for 1992 cash

The tree structure of sheets and the facilities of 3DSS provide a natural way to organize information into a menu structure, and provide an opportunity for the user to organize a spreadsheet program interface.

For example, the admissions office needs to keep track of all the student registration information in the university. The information can be divided into many departments, and each department has graduate students and undergraduate students. Figure 3.1.4.7 gives the hierarchical organization of the information. Table 3.1.4.8shows the mainsheet of registration information application. It contains five menu items, computer, history, electronic, chemistry and physics. Each corresponds to a subsheet which contains the subitems. The user can select any menu item (a cell that points to a subsheet), and open that subsheet, go into the next level, open another sheet, until he gets to the information subsheet he wants. In the above example, if one selects a menu item using the mouse left button, then presses the mouse right button[1], a subsheet containing the submenu items: graduate and undergraduate will be displayed on the screen. Table 3.1.4.9. shows the subsheet contains sub-

1. See Appendix A.

menu for the computer department. Select one menu item, say undergraduate student, and another subsheet which contains computer department undergraduate student registration information will be displayed on the screen. See table 3.1.4.10

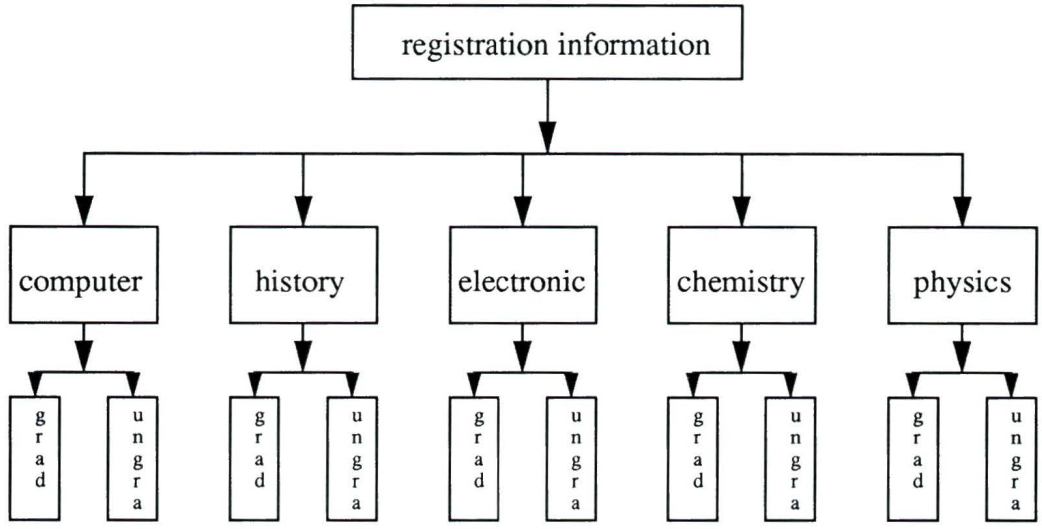

Figure 3.1.4.7. the hierarchical organization of the student registration information

	0	1	2	3	4	5	6	7	8	9
0										
1										
2				REGISTRA	TION	INFORMA	TION			
3										
4				computer	history	electronic	chemistry	physics		
5										
6										

Table 3.1.4.8. The mainsheet for the student registration information

	0	1	2	3	4	5	6	7	8
0									
1									
2									
3									
4				1	graduate				
5				2	undergraduate				
6									
7									

Table 3.1.4.9 the subsheet for the registration information in computer department

	0	1	2	3	4	5	6	7
0								
1			NAME:	NO.	STATUS			
2			R.IAN	9102123	yes			
3			J.Robert	9112134	yes			
4			W.Qian	9202516	no			
5			W.Darin	9201256	yes			
6			W.Qiang	9123348	no			
7			V.Jan	9234534	yes			
8			W.Mark	9102346	yes			
9								

Table 3.1.4.10. the subsheet for the undergraduate students in computer department

3.2 Timesheet

Some applications vary not only with space (2 dimensions), but also vary with "time". They describe problems consisting of many time frames, these frames have describable relations among them. In the 3DSS, any sheet can vary in "time" (3D) using time formu-

las. This, in effect, constructs a group of parallel sheets. We call this group of parallel sheets that vary with "time" **timesheets**. This timesheet feature in 3DSS is expected to help:

1. the user to organize information that varies in "time" by using the function *show time* to look at different timesheets simultaneously.
2. do iterative and nested iterative calculation.

3.2.1 Dimensional Operators

In a spreadsheet program, because of the 2D matrix representation, it often happens that a cell references other adjacent cells (for example, **up, down, left, right**). 3DSS adds the 3D feature that any sheet can vary with "time". The program will operate in space, and in "time" as well. So we believe dimensional operators like **left, right, up, down,** and **pre**[1] will help the user vividly operate the spreadsheet.

In 3DSS, we provide dimensional operators **left, right, up, down, here,** and **pre. Left, right, up** and **down** refer to the cell immediately to the left, right, up or down of the cell that contains the operator in its formula. **Here** refers to the cell itself. For example, if a cell already has the value 100, and the user modifies the cell's formula to **here + here**, the cell's value will become 200. **Pre** refers to the same cell in the immediately previous timesheet. Dimensional operators can be combined together to reference a cell nearby. For example, **left up**, **right right down**, **pre right down**. See figure 3.2.1.1.

1. Pre means previous.

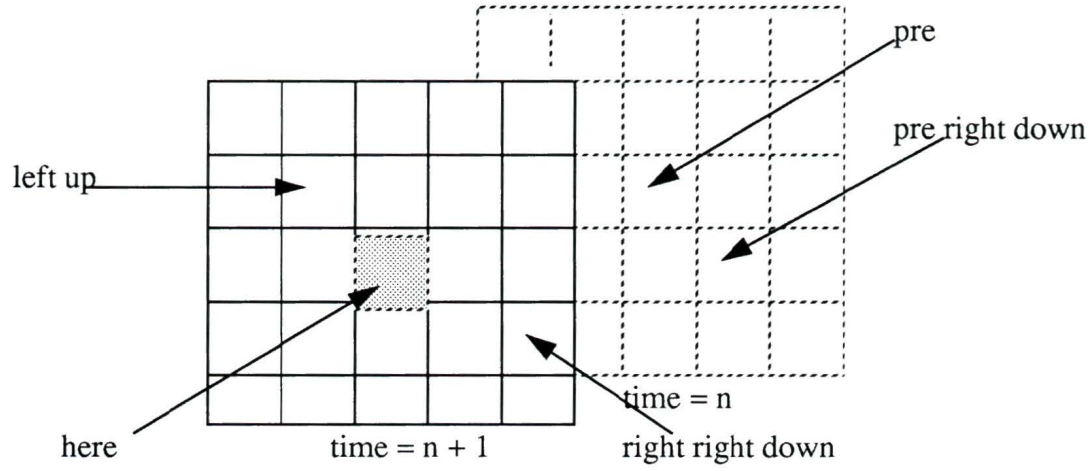

Figure 3.2.1.1. the meanings of the dimensional operators

3.2.2 Timesheet Definition

Timesheets are a group of sheets that describe relationships that vary with "time". The first timesheet in a group of timesheets is called a **starting timesheet**, others are called **following timesheets**.A starting time sheet defines the relationships how the sheet will vary with time. Following timesheets and all their descendants are special sheets: they are generated by the system. Following timesheets have the same parent sheet and the same definitions (cell name, formulas and subsheet) as those of the starting timesheet, but have different results corresponding to their time points respectively. The user can do the usual operations on these sheets except that the user can not modify the definition. The definition of relationships that vary with "time" can only be interactively defined in the mainsheet or a subsheet (described in section 3.1). If there is nested iteration, a descendant of a following timesheet can be a starting timesheet[1]. See figure 3.2.2.1.

In 3DSS, the relationships that vary with "time" are defined in a mainsheet or subsheet

1. See next section.

described in section 3.1 using **time formulas.** There are two kinds of formulas. One is the **initial formula**. The other is the **time formula** which defines how a cell's value will vary in "time". An initial formula is a general formula if its sheet is not defined as a starting time sheet. An initial formula gives the initial value of a cell that varies in "time". If a cell has only an initial formula, this cell will not change with time. That is to say, in each timesheet that varies in" time", it always has the same value as the one when time equals zero.

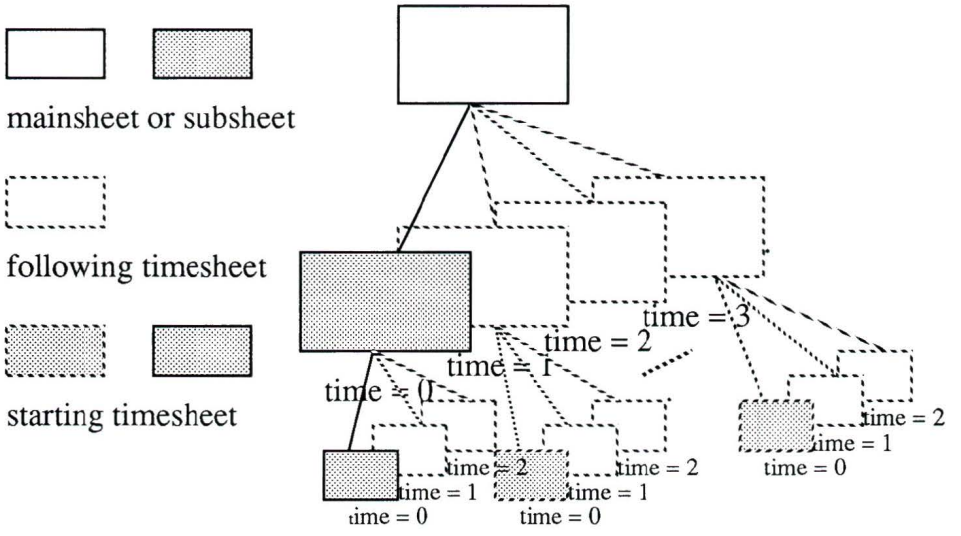

Figure 3.2.2.1 the part of the structure of a 3DSS application with timesheets

The timesheets are counted by number. Time zero refers to the sheet that has cell values are the values of initial formulas. Time one refers to the sheet that has cell values that vary ahead one "time" step and so on. The number of timesheets depends on the **end time formula** defined by the user, or the maximum number defined by the system. **End time formula** is a decision statement with the **stop** keyword[1]. The **stop** tells the system to stop generating the next timesheet. By default, the system will iterate the maximum times defined by the system if the end time formula fails to go to the stop branch, or if there is no

1. See appendix B.

end time formula. There should be only one end time formula in a starting time sheet. If the user defines more than one end time formulas, the iteration will stop whenever one of them goes to a stop statement, or reaches the maximum times defined by the system.

By default, the system displays the result of the last timesheet using the starting timesheet. The user can use the *show time* command to look intermediate timesheets as many as the user wants, as long as the memory is available.

The following example uses timesheets and dimensional operators to calculate and display retail and wholesale prices over a series of months, for a hypothetical store. Assume that prices of all items increase by 5% every month. We want to know all the retail and wholesale prices over twelve months. Initially, the prices of oranges, bananas and apples are 0.6, 0.7, 0.8. Table 3.2.2.2 shows the results at time = 0, 1 and 2. Table 3.2.2.3 gives the initial formulas and time formulas for the example.

	0	1	2	3	4	5	6	7	8	9
2				ITEM	PRICES		month	1		
3										
4				orange	banana	apple				
5			retail	0.60	0.70	0.80				
6			wholesale	0.54	0.63	0.72				

Table 3.2.2.2(a) the retail and wholesale in Jan. (time = 0)

	0	1	2	3	4	5	6	7	8	9
2				ITEM	PRICES		month	2		
3										
4				orange	banana	apple				
5			retail	0.63	0.73	0.84				
6			wholesale	0.57	0.66	0.76				

Table 3.2.2.2(b) the retail and wholesale in Feb. (time = 1)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2				ITEM	PRICES		month	3		
3										
4				orange	banana	apple				
5			retail	0.66	0.77	0.88				
6			wholesale	0.60	0.69	0.79				
7										

Table 3.2.2.2(c) the retail and wholesale in March. (time = 2)

.....	3	4	5	6	7
2					1
3					
4					
5	0.6	0.7	0.8		
6	up * 0.9	up * 0.9	up * 0.9		
7					

Table 3.2.2.3(a) the initial formulas for the retail and wholesale

.....	3	4	5	6	7
2					if pre > 11 then stop else pre +1
3					
4					
5	pre * 1.05	pre * 1.05	pre * 1.05		
6	up * 0.9	up * 0.9	up * 0.9		
7					

Table 3.2.2.3(b) the time formulas for the retail and wholesale

The end time formula is **if pre > 11 then stop else pre + 1**[1], it gives the condition that when the generation of the next timesheet is to be stopped. If the condition **pre > 11** is true, the system goes to **stop** which tells the system to finish the current timesheet calculation, stop the next timesheet generation and display the last timesheet result. Otherwise it goes to **pre + 1** which adds one to the previous cell value, generates a new timesheet and does the whole calculation again based on the cell values in the previous timesheets.

3.2.3 Iteration

Iteration is a very powerful tool in computation. In some conventional spreadsheets one can use circular references or iteration functions to do iteration. The system like Multiplan calculates the sheet over and over again using calculation natural order until some condition is satisfied. Unfortunately, this variation of time is limited. The next iteration depends on the previous iteration, a cell's value can be calculated incorrectly if the cell's calculation natural order is after other some cell natural order[3]. It is difficult to solve the problem that relates to more than two time periods, such as the calculation of Fibonacci numbers.

Actually, we can think of every iteration step as a variation of the sheets with "time". In 3DSS, we use timesheet to do iteration. Each timesheet represents an iteration step. Using the **pre** dimensional operator, one can correctly get the value in the immediately previous time without worrying about the cell's calculation natural order. Recursively using **pre**, one can solve the iteration problem involving more than two time periods.

The following example gives the calculation of Fibonacci numbers. Table 3.2.3.1 shows the formulas for the calculation of Fibonacci numbers using timesheets. If f > 2, we use

1. See appendix B.

pre pre + pre as the time formula to calculate a Fibonacci number. We use the end formula **if pre > 9 then stop else pre + 1** to define the maximum number of iteration which is ten. Table 3.2.3.2 gives the results at time 5 and 10.

.....	3	4
3		0
4		1
5		1
6		1

Table 3.2.3.1(a) the initial time formulas for Fibonacci number

.....	3	4
3		if pre > 9 then stop else pre+ 1
4		
5		
6		if n < 3 then pre else pre pre + pre

Table 3.2.3.1(b) the time formulas for Fibonacci number

	0	1	2	3	4	5	6	7	8	9
2				FIBONACCI	NUMBER					
3				N	10					
4				if N = 1	1					
5				if N = 2	1					
6				if N > 2	144					

Table 3.2.3.2(a) the result for Fibonacci number when time = 10

	0	1	2	3	4	5	6	7	8	9
2				FIBONACCI	NUMBER					
3				N	5					
4				if N = 1	1					
5				if N = 2	1					
6				if N > 2	13					

Table 3.2.3.2(b) the result for Fibonacci number when time = 5

One can use the *show time* command to look at the result at a specific time point. If there are subsheets in that timesheet, the user can look at the subsheet pointed by that cell in that timesheet at that time point. The calculation can even be a nested loop. For example, timesheet A has a subsheet which is defined as a starting timesheet, the user can opens the subsheet at time point A, then uses the *show time* command to look at different timesheets within time slot A.

3.3 3D Structured Spreadsheet

3DSS combines subsheets and timesheets together, so that one can organize information into a complex dynamic hierarchical way.

The mainsheet and its subsheets construct the 2D main frame of a 3DSS application. Following timesheets of a starting timesheets construct the third dimension. If a sheet varies with time, all its subsheets also vary with time. If a sheet varies with time and a subsheet of it also varies with time, the subsheet varies with time at its each supersheet time point. It opens another third dimension at its each supersheet time point. So, 3DSS is multiple dimensional. See figure 3.3.1.

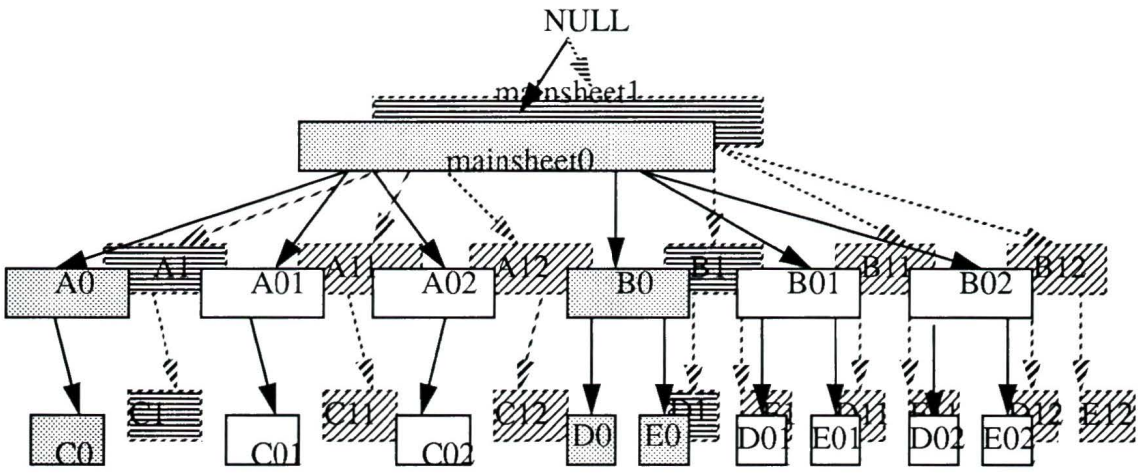

Figure 3.3.1. the structure of a 3DSS application

In the above picture, the mainsheet, subsheets A0, B0, C0, D0 and E0 construct the main frame of the application. They are interactively created and defined by the user. The mainsheet has two subsheets, A0 and B0. A0 has one subsheet C0. B0 has two subsheets D0 and E0. The mainsheet, A0 and B0 are defined as starting timesheets. Other sheets (without dot shadow) are all following sheets which are generated by the system. They have the same parent and same definitions as their starting timesheets, but their results each correspond to their specific time. The mainsheet varies with time, so all its subsheets vary with time (from front to back with horizonal slash). Within any mainsheet specific time point, the mainsheet's two subsheets also vary with time (from left to right). When time = 0 for the mainsheet, A0 and B0 vary with time (A01, A02, B01, B02). When time = 1 for the mainsheet, A1 and B1, which are starting time sheets, vary with time(A11,A12, B11, B12).

All the references in subsheets and time subsheets follow the scoping rules. For example, in the above picture, when time = 1, if one cell in timesheet D1 references a cell name in its supersheet, the system will first search B1, then search the mainsheet1. There is another

reference relationship. The relationship among timesheets. The cells in the following timesheets can reference a cell in its previous timesheets using the **pre** operator. So, in summary, in 3DSS, the reference relationships among the sheets are from bottom to top and from right to left.

The number of timesheet does not have to be fixed. An end time formula in a starting time subsheet can reference a cell's value in its supersheet, which in turn can be changed by the user or can reference other cells. So 3DSS can organize information into a complex dynamic hierarchy. Dynamic means the number of the time subsheets pointed by a cell can be dynamically changed. In effect, the tree structure of a 3DSS program can be changed.

The following gives an example using subsheets and time subsheets. This example extends the item prices given in 3.2.2. All the fruit prices will increase 5% every month. All the meat prices will increase 6% every month. We create two groups of time subsheets, one for the fruit, and the other is for the meat. These two groups of time subsheets share a common named cell **month**, which is in the mainsheet. We use it to control how many months we want to calculate in each group of timesheets, that is to control the number of timesheets in each group. Figure 3.3.2(a) gives the mainsheet of the application. Select cell **Fruit**, press the right button to open the subsheet. The opened time subsheet is the last timesheet in the timesheet group, which is the fruit prices of month Dec. See figure 3.3.2(b). One can use the *show time* command to look at other timesheet in that group. Figure 3.3.2(c) gives the fruit timesheet at time 11 and at time 5. If one changes **month** cell from 12 to 6., the number of the two groups of timesheets will be 6 instead of 12.

	0	1	2	3	4	5	6	7	8	9
0										
1										
2				ITEM	PRICES		month	12		
3				1.	Fruit					
4				2.	Meat					

Table 3.3.2(a) the mainsheet for item price

	0	1	2	3	4	5	6	7	8	9
0										
1										
2				month	2					
3										
4				orange	banana	apple				
5			retail	0.63	0.73	0.84				
6			wholesale	0.57	0.66	0.76				
7										

Table 3.3.2(b) The fruit prices in Feb. (time = 1)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2				month	5					
3										
4				orange	banana	apple				
5			retail	0.77	0.89	1.02				
6			wholesale	0.69	0.80	0.92				
7										

Table 3.3.2(c) the fruit prices in June (time = 5)

3DSS combines subsheets and timesheets together, so it has all the advantages of subsheets and timesheets. It is expected to provide a powerful tool to let the user organize information into a flexible structure as one like, and provide more powerful calculation capabilities. Further more, it allows one to define relatively complex procedures and functions through programming-by-example.

3.4 User-defined Procedure and Function

Spreadsheet programming is end user programming. The matrix format leads to a natural representation of many problems. Program design and program testing are combined together: the user's operation is interactive, values are permanently displayed, any changes to the cells are constantly up-dated.

However, a conventional spreadsheet program does not constitute a complete programming environment, mainly because it does not incorporate user-defined functions and procedures for abstraction and structuring larger programs[8]. The user can use Macros to define functions or procedures, but the Macro programming paradigm is different from that of spreadsheet, and it is a step back to conventional programming. However, defining functions and procedures is as easy as developing a spreadsheet program in 3DSS through programming-by-example using subsheets and timesheets.

In 3DSS, a subsheet and all its descendants (including timesheets) can be defined as a function or procedure. So a function or procedure is a set of sheets which have the same structure described in the previous section. The user defines the input parameters whose values will be provided when it is called in a formula. All the parameters are transmitted by value like the VALUE transmission in PASCAL.

In 3DSS, a function can return a number after its execution through a **return** statement in its root sheet[1]. A 3DSS procedure is different from conventional language procedure. Because of the local feature of the spreadsheet programming, it have no side effects. A procedure in 3DSS is an abstract unit that displays independent information.

The basic idea of programming-by-example is to first tackle an example problem in a trial and fashion, and in case of success, identify the relevant parameters of the problem[8]. The user starts by entering some constants as input data in a supersheet, then defines the function or procedure starting from a subsheet. The constants in the supersheet can be regarded as intermediate test data. The user can modify these data in the supersheet, modify and test the prototype function or procedure in the subsheets and so on. Eventually, the user will get the desired results. The user can define a procedure or a function by assigning a name, and defining input parameters for the function and procedure using the *define function/procedure* command. If an input data is defined in the function or procedure's parameter, the user need to provide this data when it is called later. If an input data is not defined in the parameters, the system will try to find the referenced data in the sheet or its supersheets in which the function or procedure is called.

The following gives an example how to define a function for getting absolute values in 3DSS. The function has one input data, the data for the absolute value, and one output data which is the result. In a supersheet, for example in the mainsheet, in the cell R[0]C[0], enter value -2, give the cell name X. Then select any cell which is empty, say R[1]C[1], define it as a cell containing a subsheet called **abs**. Then at the subsheet, select any cell, say R[2]C[1], define its formula as **if X > 0 then X else -X.** Now the result 2 is displayed

1. A function or procedure is a set of sheets which has the same structure as a spreadsheet program. The top sheet is the root sheet for the function or procedure. There is only one return statement in a sheet. If there is a return statement in a sheet other than the root sheet, the return statement is the same as the return statement in a subsheet.

at the cell R[2]C[1] in subsheet **abs**. Below the R[2]C[1] in subsheet **abs**, enter another formula **return up**, the result 2 will be displayed in the cell R[3]C[1] in subsheet **abs** and in the cell R[1]C[1] in the mainsheet. You can change the value of X in the supersheet to test other cases. When you believe the program is correct, select the *define function/procedure* command, enter the input parameter X, then press OK, a new function is defined. Later you can use the function just like a built-in function.

The following gives an example to calculate square root by Newton's method using the user-defined function **abs**. See table 3.4.2(a) and 3.4.2(b). The **Difference** shows absolute value of the difference of the current result and the previous iterative result. The cell uses the user-defined function to calculate the absolute different value.

	0	1	2	3	4	5	6	7	8	9
0										
1				ROOT	BY	NEWTON	METHOD			
2				guess	1.00					
3				x	100.00					
4				tolerance	0.01					
5				difference	17.52					
6				root	10.84					
7										

Table 3.4.1(a) Newton's method for square root using the user-defined function abs, time = 4

.....	2	3	4	5	6
3			1.00		
4			100.00		
5			0.01		
6			abs(x - root * root)		
7			g[a]		
8					

a. g is the given name for cell R[3]C[4]. root is the given name for cell R[7]C[4]

.....	2	3	4	5	6
3					
4					
5					
6			abs(x - root * root)		
7			if abs(x - pre* pre) < t then stop else 0.5 * (pre + x/pre)[a]		
8					

a. x is the given name for cell R[4]C[4], t is the given name for cell R[5]C[4]

Table 3.4.1(b) the initial and time formulas of Newton's method for square root using the user-defined function abs

There is no restriction on the definition of a function or procedure. As long as a problem can be solved using the facilities provided in 3DSS (subsheet, timesheet), it can be defined as a function or a procedure.

Furthermore, in 3DSS the ability to provide user-defined functions or procedures gives more powerful facilities for information organization. If a cell calls a function or a procedure, one can select that cell and open the subsheets to display the contents of the functions or procedures (if there are more than one function/procedure calls in the cell, the system will display all the contents of the function/procedure calls.). This is very useful

for an application that has many similar parts which only input data is different. We can use abstraction technique to define a function or procedure. We use the function or procedure with different input data, and look at the function or procedure contents just like opening subsheets.

CHAPTER 4

IMPLEMENTATION

4.1 Overview

A conventional spreadsheet is a big, complex software package which provides a friendly user interface, many commands and built-in functions. It also supports many kinds of data types: integer, decimal, money, date etc. The goal of our implementation was to only illustrate the basic ideas behind 3DSS. So we only implemented a simple user interface, text and number data types and some commands to demonstrate the power of 3DSS -- subsheets, timesheets and user-defined functions and procedures.

In our implementation, there are four function parts: the user interface, the spreadsheet parser, the evaluation engine and the data manager. Figure 4.1.1 shows the relationships among these parts. The user interface is responsible for creating, destroying, displaying windows (one sheet corresponds to one window), accepting user inputs and commands, updating cell results in windows and displaying status and error information. According to the user input, it will communicate with appropriate modules: the parser, the data manager, or the evaluation engine. This module is mainly composed of window's call back functions[1]. The spreadsheet parser translates formulas (initial formula and time formula)

1. 3DSS is implemented under X-Windows Xview. A callback function is a kind of function or procedure that is called by the system, not by the program[14].

into internal parse trees for later evaluation. The data manager is responsible for creating, initializing, deleting, modifying and maintaining all the data structures. The evaluation engine evaluates formulas (initial formula and time formula) and also does the recalculation. In this chapter we will emphasize the internal implementation related to the special features of 3DSS. The user interface and the parser are described in appendix A and appendix E.

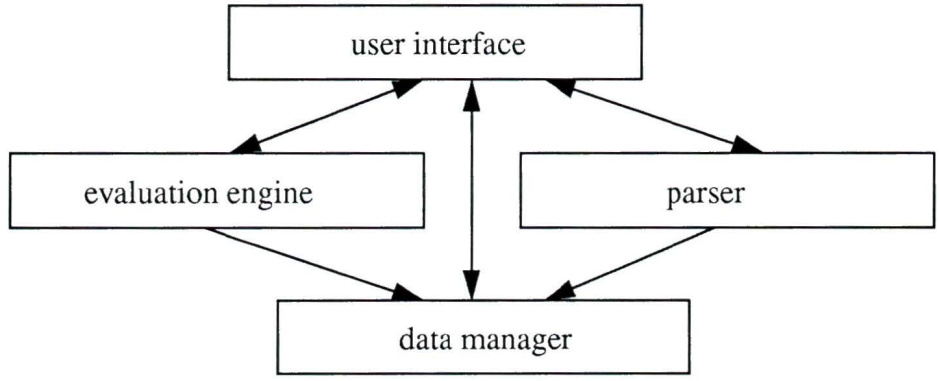

Figure 4.1.1 the internal structure of the 3DSS implementation

4.2 the Data Manager

The data manager module is responsible for all the management of internal data structures for 3DSS.

4.2.1 SHEET and CELL Data Structures

In 3DSS the major data structures are SHEET and CELL.

SHEET contains all information related to a sheet. Figure 4.2.1.1 gives the description of a

SHEET data structure.

SHEET

CELL: records all the information about each cell in the sheet.

current cell: records the current cell address in the sheet.

parent sheet: points to its parent sheet. This pointer is NULL in the mainsheet.

windows information: records all the window information that the sheet is in.

Figure 4.2.1.1 SHEET data structure

Each cell corresponds to a cell data structure. Figure 4.2.1.2 gives the description of a CELL data structure.

CELL

cell value: records the current value of the cell.

cell name: records the name of the cell. If there is no name in this cell, it is an empty string.

cell type: records the value type of the cell. It can be text, boolean or numerical.

cell initial formula: records the initial formula string for the cell. If there is no initial formula for this cell, it is an empty string.

cell time formula: records the time formula for this cell. If there is no time formula for this cell, it is an empty string.

cell initial formula parse node: records the internal parse tree for the initial formula.

cell time formula parse node: records the internal parse tree for the time formula.

Pointer to subsheet: if the cell points to a subsheet, this pointer points to a SHEET structure, otherwise, it is NULL.

initial reference chain: a link that records all the cells that reference this cell in their initial formula.

time reference chain: a link that records all the cells that reference this cell in their time formula.

initial calculation natural order: a number indicates the cell's initial formula calculation natural order in the sheet.

time calculation natural order: a number indicates the cell's time formula calculation order in the sheet.

Figure 4.2.1.2. CELL data structure

Figure 4.2.1.3 gives an example of the structure organization of a 3DSS program.

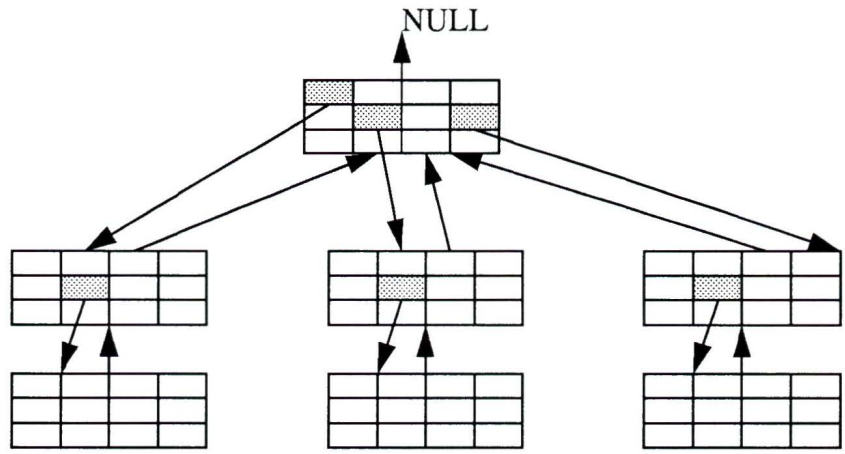

Figure 4.2.1.3. An example of the structure organization of 3DSS

In the above figure, every big square represents a sheet, every small square represents a cell. Shaded cells contain the subsheets. The scope of cell is realized automatically by the way of data structure organization. The evaluation engine searches one level after another following the parent pointers if a cell references the values in its supersheets. If there are the same names in a subsheet's supersheets, this guarantees that it uses the nearest local one. The user interface uses subsheet pointers to open, display and operate on subsheets.

A subsheet is created or connected, disconnected or deleted in response to the actions from the user. Some cell items are created, modified or deleted when the user performs operations in the cell.

From the above descriptions, one may notice that there is no sheet data structure to repre-

sent timesheets. In 3DSS, a timesheet is dynamically created and destroyed. If a sheet is defined as a starting timesheet that varies in "time", the starting sheet is used as a displaying sheet to display the final results. If the user wants to look at timesheets at different times, the system will create new sheets to display the results at those times, whose parent are the same as their starting timesheet[1]. The reason for creating and deleting timesheets dynamically instead of creating all the timesheets and keeping all the timesheets in the system is to save memory space, because the number of timesheets is not fixed. Sometimes it can be very large.

4.2.1.1 Reference Chain

One cell may be referenced by many other cells. Cells that reference a given cell constitute a **reference set.** In 3DSS we record the reference set statically instead of finding it dynamically during recalculation. We use a link to represent the set, which is called a cell's **reference chain**. In 3DSS, there are two kinds of formulas: initial formulas and time formulas. Each kind of formula corresponds to a reference chain in a cell. So for a cell, it has an initial formula reference chain and a time formula reference chain. After the parser has translated a cell's source formulas into internal parse trees, and the evaluation engine has successfully evaluated the values, the user interface will call the data manger to check the reference relationships. It will add this cell into the appropriate cell's reference chains if the cell has referenced other cells.

In 3DSS, reference chain provides data records to update only the related cells when one cell has been modified.

1. In section "time sheet engine", we will discuss timesheet creation and deleting in detail.

4.2.1.2 Calculation Natural Order

In 3DSS, there are two kinds of calculation natural order, one is for the initial formula, the other is for the time formula. Calculation natural order is represented as a series of number starting from zero. A cell with a smaller number is calculated before a cell with a larger one. Cells with the same numbers are independent of each other, and can be calculated in any order. For the example given in 2.1.2, the initial formula calculation natural order is:

R[1]C[1] = R[2]C[1] = R[3]C[1] = 0

R[5]C[3] = R[6]C[3] = R[6]C[4] = R[6]C[5] = 1

R[7]C[4] = 2

R[8]C[4] = 3

R[5]C[4] = 4

R[7]C[4] = 5

R[8]C[4] = 6

R[5]C[5] = 7

R[7]C[5] = 8

R[8]C[5] = 9

R[9]C[3] = 10

Each time the data manager adds a cell to the appropriate cell's appropriate reference chains, it calculates the calculation natural order number for it.

Calculation natural order provides a fast way to recalculate a whole sheet. In 3DSS, it's especially useful for the user-defined function and procedure calculations and the timesheet calculations.[1]

1. This will be discussed in section 4.4 "Evaluation Engine".

4.2.2 Data Structure for User-defined Function and Procedure

In general, there are three parts in a function or a procedure: input parameters, function or procedure body, and output. Only a function has output which is realized by the **return** statement. Procedure does not has any output. In 3DSS, a user-defined function or procedure body is represented internally as SHEET data structure. The data structure for functions and procedures in 3DSS is FUNCTION:

function/procedure name: points to a function/procedure name string.

input parameters: a linked list of the input parameters.

function/procedure body: SHEET data structure.

Figure 4.2.2.1 FUNCTION data structure

When the user defines a function or a procedure, the user defines a template. Any call to the defined function or procedure will generate an instance of the template like CLASS and INSTANCE object in C++. By default, the system creates an instance of the called function or procedure, does the calculation in all the sheets in the function or procedure, returns the value if necessary, then destroys the instance to save the memory. The system will create special set of sheets for an instance of the function or procedure call again when the user wants to look at the content of the instance.

The data manager maintains a template library, which can be accessed by the evaluation engine. In the current implementation, the library is just a linear link of FUNCTION data structure.

There are many advantages using the SHEET structure to represent function or procedure bodies. Firstly, from the point of view of the spreadsheet language, the sheet is the major unit of a spreadsheet program. Functions or procedures are an abstract mechanism for

abstraction and repetition, which is a special piece of spreadsheet program. This makes user-defined functions or procedures in the same spreadsheet paradigm, just as block structure language's block, function and procedure can contain the same statements as those of the main program. Secondly, user-defined functions or procedures provide another way to use abstract units to apply to different situations. The system can easily uses the subsheet mechanism to open a function or procedure if the user wants to view the contents of an instance of a function or procedure.

4.3 the Evaluation Engine

The evaluation engine is responsible for evaluating cell values. After the parser translates the source formulas into internal parse trees, the user interface calls the evaluation engine to evaluate the cells and do the appropriate recalculation.

4.3.1 Formula Execution

The Formula-Execution procedure evaluates a formula's parse tree, and returns the result. It is nothing but a big case statement. The Formula-Execution procedure evaluates a parse tree from left to right, bottom to top. It recursively calls itself to go from a higher level to a lower level. After it gets all the necessary values for a node, it calculates the node's value according to the node type, then returns the node value to one up level. This procedure continues till it finishes the calculation of the highest level, then returns the final result. See appendix D for further information.

4.3.2 Time Sheet Execution

If a sheet does not have a time formula, the system can only calculate the modified cell's

value and update its dependent cell values along its initial formula chain[1]. However, if a sheet has at least one time formula, the system can not just simply calculate the modified cell and update its dependent cells, for the cells in the sheet that contain the time formulas will vary with "time".

In 3DSS, evaluation of a starting timesheet is finished by Time-Sheet-Execution procedure. It fully uses the initial formula calculation natural order and time formula calculation natural order. Every time, the user modifies a cell's formula in a starting time sheet, the system will call the Time-Sheet-Execution procedure. It recalculates all the cell's initial formulas in the given sheet according to the initial formula calculation natural order, then calculates all the cell's time formulas in the given sheet according to the time formula calculation natural order. The following gives the algorithm of the Time-Sheet-Execution procedure (SHEET * Sheet).

1. sorts the initial formula calculation natural order numbers in an increasing order. (only lists the cells that contain an initial formula).
2. if the list is not empty, calls Formula-Execution procedure to get all the initial formulas' value in the order of the sorted list, else exit.
3. records these results into the given sheet.
4. sorts the time formula calculation natural order numbers in increasing order (only lists the cells that contain time formulas)
5. if the list is not empty, creates a new SHEET structure as a new next timesheet, copies the previous timesheet into the new timesheet. Calls Formula-Execution procedure to get all the time formula's values in the order of the shorted list, records the results into the new sheet, else exit.
6. If in this iterative step, the end time formula goes to a **stop** branch, or reaches

1. We will discuss recalculation in the next section.

the number of a time sheet that the user wants to look at[1], or reaches the maximum iteration number, it copies the latest sheet into the starting sheet, frees all the new created sheet. Otherwise repeats step 5.

The most important dimensional operator for timesheet is **pre.** During the calculation of time formula, timesheets are connected through the parent pointer, that is, the later timesheet points to its previous sheet through the parent pointer. The starting timesheet's parent pointer points to the real parent sheet that is shared by all the timesheets. The **Pre** operation goes to previous timesheet. Recursively using **pre,** like **pre pre...** finds the value in the several timesheet ahead along the parent pointer.

4.3.3 Recalculation

When a cell is modified, spreadsheets will recalculate the necessary cells so that the computation is always up-to-date ("what if...?). In chapter 2, I have described three recalculation methods in conventional spreadsheet. In 3DSS, this recalculation is more complex because of the introduction of subsheet and timesheet. There are two ways of recalculation in 3DSS: the recalculation of cells in a starting timesheet and the recalculation of cells in a sheet which is not a starting timesheet. During the recalculation, before encountering a cell in a starting timesheet, it uses the third method to only up-date the dependent cells along the initial formula reference chain using Formula-Execution procedure. Once there is recalculation involving a cell in a starting timesheet, it recalculates the whole starting timesheet and all its descendants using Time-Sheet-Execution procedure.

---

1. The algorithm stops in two condition: 1. It stops when it reaches the number inputted by the user to look at a specific timesheet. 2. It stops when the end time formula goes to **stop** branch.

4.3.4 Function Execution

In 3DSS a function or procedure body is represented as a SHEET data structure, so the evaluation of a function or procedure call is just like the evaluation of a sheet and all its subsheets using the Time-Sheet-Execution procedure. The following gives the algorithm of the Function-Execution(EXPR * parser-node) procedure.

1. finds the function/procedure name in the user-defined library (template), if not found, exits.
2. calls the Formula-Execution procedure to evaluate the input parameters, and records the values into the input parameter's link[1].
3. creates a new FUNCTION structure, copies the found function/procedure in the user-defined library to the new FUNCTION structure (instance), copies the input parameter's values from the input parameter's link into the new FUNCTION structure.
4. calls the Time-Sheet-Execution to evaluate all the sheets.
5. deletes the new FUNCTION structure.

When the function execution engine calls Time-Sheet-Execution procedure, it tells it that the evaluation is a function/procedure body evaluation. There are two differences between a function evaluation/procedure and general sheet evaluation: one is the calculation of cells that reference the input parameters, the other is for the return statement in a function definition. If a cell in a function body references a name that is defined in the input parameters, Formula-Execution procedure gets the value from the actual parameters in the new created FUNCTION structure, instead of trying to search in the current sheet or its supersheets. If it can not find the referenced name, it will try to search in the current sheet or in the supersheets. For the return statement in a function definition, the Formula-Execution procedure returns the value immediately without telling the data manager to up-date its

1. See appendix C

parent sheet's cell.

The implementation of the 3DSS includes 14 files: 12 C source code files and two include files. Altogether is about 10000 line C source code.

CHAPTER 5

CONCLUSIONS AND FURTHER WORK

5.1 Conclusions

The 3DSS is a new kind of spreadsheet using program organization concepts similar to those of block-structured languages. A spreadsheet program can contain many sheets: one mainsheet and optionally many subsheets. A cell in a sheet can contain formulas (initial formula and time formula), and a subsheet or a group of timesheets. Each cell can be given a name, and the reference relationships among the cells of parent sheets and subsheets follow the block-structured scoping mechanism. Any sheet can be defined using time formula as a starting timesheet which varies with "time" (3D) and constitute a group of time frames -- timesheets. The system combines subsheet and timesheets together to allow to organize sheets in a 3DSS program into a complex dynamic hierarchical structure. In addition, one can define functions and procedures using subsheets and timesheets through programming-by-example in the same spreadsheet programming paradigm.

The 3DSS design is based on ideas of conventional spreadsheets, and it has all the core features that a traditional spreadsheet has. The hierarchical structure of 3DSS helps the user to organize information into a real top-down hierarchy. The rules of name scope allow one to use the same name in different sheets for different purpose without fear of name clashes. It gives the spreadsheet programmer a chance to organize a spreadsheet

application's user interface. The 3D feature of 3DSS helps the user to organizes information that varies with "time", and it also provides a utility to do iteration and nested iteration in the 3D point of view. 3DSS combines subsheet and timesheet together so that one can organize information into a complex dynamic hierarchy. Defining functions and procedures is as easy as developing a spreadsheet program through programming-by-example using subsheets in the same spreadsheet programming paradigm. It provides more powerful ability to organize blocks of information having similar structure.

5.2 Further Work

The current implementation is only a demonstration version, which is a small model[1]. We only implemented part of current spreadsheet functions. More work could be done on the user interface design, on large model memory management and on providing more user commands and built-in functions.

Because of the introduction of subsheet and timesheet, the recalculation is more complex than that of a conventional spreadsheet. The current recalculation algorithm is not the most efficient one. Once it encounters a recalculation related to a starting timesheet, it will recalculate the whole starting timesheet and all its descendants. More information can be obtained through the search along the cell's initial formula reference chains and time formula reference chains. When the system updates the cells which are not in a starting timesheet during recalculating the timesheet's descendants, it only updates the dependent cells instead of recalculating the whole sheet. The current recalculation algorithm does not detect circular references. If the user enters a circular reference, the system will go into a dead loop. An algorithm needs to be developed to detect direct and indirect circular references.

1. The sheet size is fixed, 10 rows, 10 columns

In the current 3DSS implementation, we only consider to organize information in one program. Some applications may need two or more programs share data, for example, two or more programs share data through network, or when an application is too big to fit into memory. In conventional spreadsheets, the user can use links to share data among independent sheets in different files. The 3DSS organizes sheets into a complex dynamic hierarchy, which provides a good way for information encapsulation. How to provide mechanism to support data sharing among 3DSS programs need to be further studied.

In 3DSS a cell can contain a subsheet. An extension of 3DSS can allow a cell not only contain a subsheet, but also point to a graphics, a text file or even a program and so on, so that 3DSS becomes a powerful tool for organizing information and end-user programming.

Bibliography

[1]Thoms B. henderson, Douglas Ford Cobb, Gena Berg Cobb "Spreadsheet Software From VisiCalc to 1-2-3", Que Corporation, Indianapolis, 1983.

[2] Weichang Du "An Intentional 3-D Spreadsheet and Its Implementation" Department of Computer Science, University of Victoria, 1986.

[3] Lotus 1-2-3 for Windows, Quebec Development Group

[4] John E. Nicholls "The Structure and Design of Programming Languages", IBM(UK) Laboratories Ltd., 1975

[5] Andrew T.Williams "Lotus 1-2-3 from A to Z", 1985

[6] Edited by Brad A.Myers, "Languages for Developing User Interface", 1992

[7] Barry Shell "Running Hypercard with Hypertalk", Management Information Source, Inc. ,1988

[8] Michael Spenke, Christian Beilken "A Spreadsheet Interface for Logic Programming", Geerman National Research Center for Computer Science, 1989

[9] M.H Van Emden "Spreadsheet with Incremental Queries as a User Interface for Logic Programing", Department of Computer Science, University of Waterloo, 1986

[10] William A. Barrett, Rodney M.Bates, David A. Gustafson, John D.Couch "Compiler Construction, Theory and Practice", second edition, 1986

[11] O Jerry Parker, Gary L.Breneman "Spreadsheet Chemistry", Department of Chemistry and Biochemistry, Eastern Washington University, 1991

[12]Glenn I. Ouchi "Lotus in the Lab: Spreadsheet Applications for Scientists and Engineers", BREGO Research and Laboratory PC Users Group, 1988.

[13]Michael Griffin, Paul Mcfedries, Don Scellato and Debbie Wkowski "Excel 4 Super

[14] Dan Heller, "Xview Programming Manual", third edition, O'Reilly & Associates, Inc., 1991.

[15] Boug Bell, Mike Parr, "Spreadsheet: a Research Agenda", ACM SIG PLAN Notices, Sep, 1993

[16] William G. McArthur, J. Winston Crawley, "Structure Data with Turbo Pascal-- A Practical Introduction to Abstract Data Types" Shippensburg University Prentice Hall, Englewood Cliffs, New Jersey 07632, 1992.

APPENDIX A

3D Structured Spreadsheet User Interface

Overview

The 3DSS display has two parts: control panel and cell area. The control panel contains a tool box, a user-defined name list, a user-defined function/procedure name list and a status information line. The cell area is organized into 10 rows and 10 columns. The control panel provides different commands that the user can use to operate the spreadsheet. The user-defined name list lists all the given names in a sheet. The user-defined function/procedure name list gives all the user-defined function/procedure names in the system. The status information line gives the current system status.

There are three kinds of sheets in 3DSS, the mainsheet/subsheet, timesheet (not including starting timesheet) and the function/procedure sheet. In the mainsheet and any subsheet, the user can perform interactive programming. The user can only look at timesheets or function/procedure sheets, he/she can not enter or modify the content of the cells.

The following figures show the scheme of the mainsheet, subsheet, timesheet and function/procedure sheet:

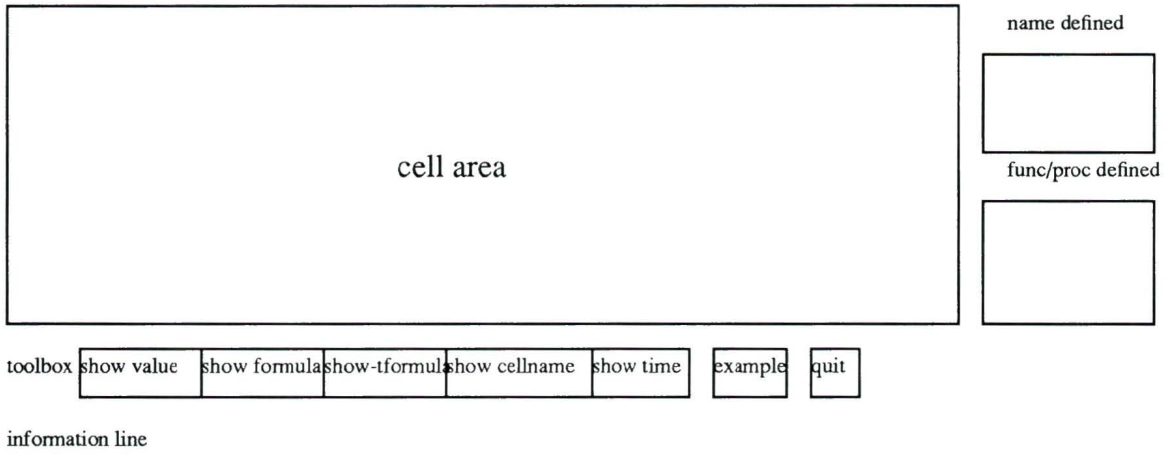

Figure 1. the user interface for the mainsheet

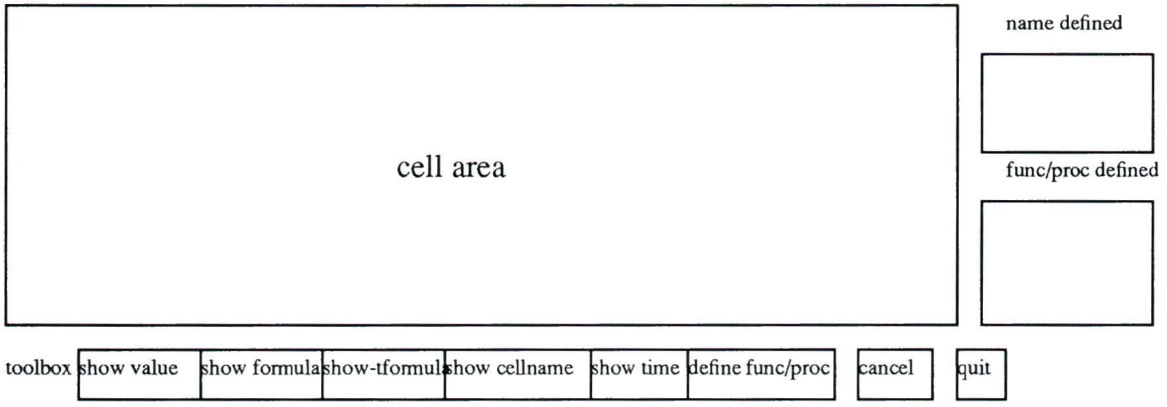

Figure 2. The user interface for a subsheet

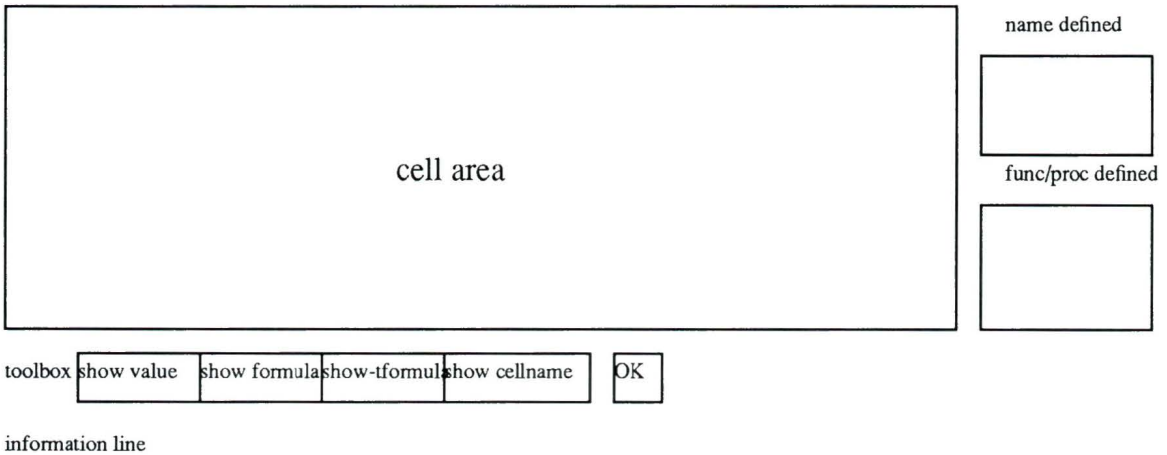

Figure 3. The user interface for a timesheet

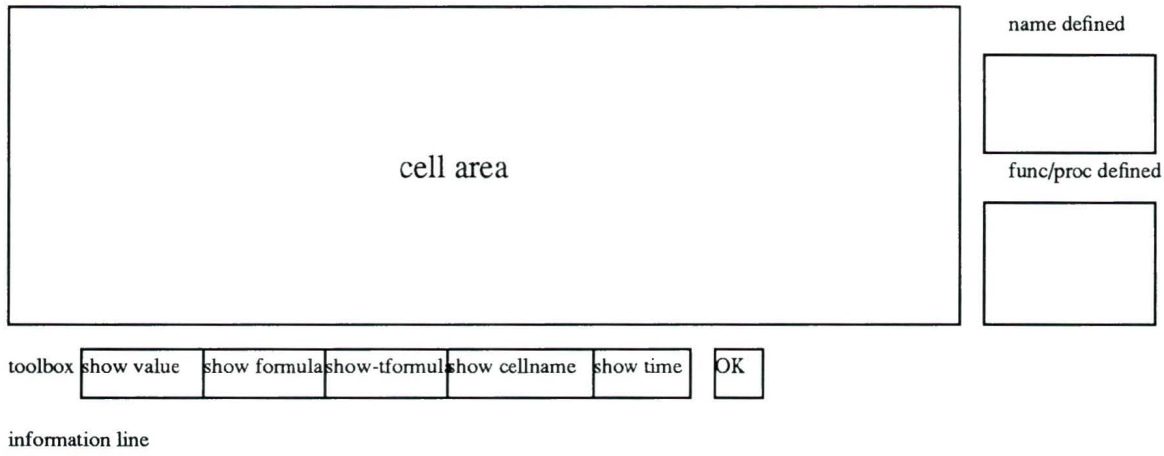

Figure 4. The user interface for a function/procedure sheet

Tool Box

Different kinds of sheets have different tool boxes. The items in the tool box for the main-sheet are *show value, show formula, show-tformula, show cellname, show time, example* and *quit*. The items in the tool box for a subsheet are *show value, show formula, show-tformula, show cellname, show time, define func/proc, cancel* and *OK*. The items in the tool box for a timesheet are *show value, show formula, show-tformula, show cellname* and *OK*. The items in the tool box for a function/procedure sheet are *show value, show formula, show-tformula, show cellname, show time* and *OK*.

show value: displays cell's values in the cell area. In this mode, the user can perform interactive programming, for example, enter data, define a subsheet, look at, or edit the contents of a cell.

show formula: displays the cell's formulas in the cell area.

show-tformula: displays the cell's time formulas in the cell area.

show cellname: displays the cell's given names in the cell area.

show time: when the user selects this tool, the system will pop up a dialogue box, asking which timesheet the user wants to look at. The user enters a number, press *OK* button, then the system will display the timesheet the user wants.

define func/proc: when the user selects this item, the system will pop up a dialogue box, asking the user to enter the input parameters. The user enters the input parameters, presses the *OK* button, then the system will create the new user-defined function or procedure.

example: when the user selects this item, the system will pop up an example list provided by the system.

quit: exit 3DSS.

cancel: discards all the operations performed this time.

OK: accepts the operations performed this time.

User-defined Name List and Function/Procedure List

The user-defined name list gives all the names given to the cells in a sheet. The user can select one of the items and look at more information about the name. For example, which cell has the name, the cell's contents. Each sheet has a different user-defined name list which lists the names defined in this sheet.The user-defined function/procedure list gives the list of all the user-defined functions/procedures. The list is the same for all the sheets. The user can select an item in the list to look at the detailed information about the function/procedure. For example, input parameters, or a delete button to delete one of the user-defined functions/procedures in the list.

Cell Area

At any time, there is a current sheet, into which the user can enter data. By default the mainsheet is the current sheet. The user can use the middle button to change current sheet

from one sheet to another. At any time, there is a current cell in the current sheet. The user can use the left button to move the current cell from one cell to another cell in the current sheet. The second time the user presses on the current cell, a dialogue box pops up. It displays all the detailed information about the cell. The user can enter, or edit the contents of the cell in the dialogue box.

The dialogue box includes:

. dialogue title: gives the current cell's absolute address and its type.

. *cell value*: gives the current value of the cell.

. *formula:* for initial formula.

. *symbol:* for text and subsheet name.

. *cell name:* for the cell name.

. *time formula:* for the time formula.

. *subsheet* button: defines a new subsheet.

. *clear* button: clears the contents of the cell.

.*cancel* button: cancels the operations performed this time.

. *OK* button: accepts the operations performed this time.

A subsheet is defined through the cell dialogue box. If the user wants to define a cell to point to a subsheet, he/she simply gives a name for the subsheet in the symbol field, and presses the *subsheet* button. A new sheet will appear on the screen, which becomes the current sheet. The user can continue to do programming in the subsheet.

If a cell points to a subsheet, the cell is outlined in blue. If a cell's formula (initial formula) calls a user-defined function/procedure, the cell is outlined in green. The user can use the left button to select a cell outlined in blue or green, then press the right button to open the subsheet or look at the contents of the called function or procedure[1]. The user can click on

the *CANCEL* or *OK* buttons in the subsheet or function/procedure sheet to close the subsheet or function/procedure sheet.

Sheet Title

Every sheet has a name. The mainsheet's name is MAIN. The subsheet title is composed of all its supersheets plus its own name. For example: one cell in the mainsheet contains a sheet A, and a cell in A contains a subsheet B, so the sheet B's title name is **main.A.B**. The timesheet title is its starting timesheet title plus the iteration number, for example: **main.A.B iteration:2**. A function/procedure sheet title is the sheet title that the function/procedure is in plus function/procedure name, for example: **main.A.B function/procedure call abs**.

Error Information

All the information (including instruction information and error information) is displayed through pop up dialogue boxes. All these dialogue boxes, choices are given to the user can select when an error occurs. For example, when the parser meets a parse error, it will display a pop up dialogue box to tell the user what grammar error or syntax error it has encountered, then the system returns to the state before the error occurs.

---

1. If there are more than one function calls in the initial formula, the system will display all the instances of the function calls.

APPENDIX B

3D Structured Spreadsheet Grammar

KEY WORD

LEFT RIGHT UP DOWN HERE PRE IF THEN ELSE RETURN STOP

LITERAL

R C @ + - * / > < & | = . ( )

NAME

any length of character string which starts at character except R,C

CELL ADDRESS

CELL ADDRESS ::= NAME | NAME "." NAME | NAME "." "[" EXPR "]" "C"

"[" EXPR"]"| "R" "[" EXPR"]""C""["EXPR"]"| "@" NAME

| "@" "R" "[" EXPR"]" "C""["EXPR"]"

DIMENSIONAL OPERATORS

DIMENSINALOP ::= "LEFT"|"RIGHT"|"UP"|"DOWN"|"HERE"|"PRE"

NUMBER

NUMBER ::= {DIGIT}["."{DIGIT}]

EXPRESSION

CALL ::= NAME "(" [EXPR] ")"

EXPR1 ::= NAME|NUMBER|CALL|"(" EXPR")"|DIMENSIONALOP[DIMENSION

LOP]

EXPR2 ::= ["-"]EXPR1

EXPR3 ::= EXPR2 { "*" |"/"EXPR2}

EXPR4 ::= EXPR3 { “+” |”-” EXPR3}

EXPR5 ::= EXPR4 {“>” |”<“ |”=” EXPR4}

EXPR6 ::= EXPR5 { “&” EXPR5}

EXPR7 ::= EXPR6 { “|” EXPR6}

EXPR ::= EXPR7

IF STATEMENT

IFSTSMT ::= “IF” EXPR “THEN” EXPR “ELSE” EXPR

RETURN STATEMENT

RETURNSTMT ::= “RETURN” EXPR

APPENDIX C

Major Data Structure Definitions of 3D Structured Spreadsheet

CELL Structure

```
typedef struct
{
int type; /* cell's value type: text, number boolean */
double constant; /* value for number and boolean */
char cellvalue[40]; /* cell's value string */
char cellname[20]; /* cell's name */
char symbol[40]; /* for text */
char formula[80]; /* initial formula string */
char iteration-formula[80]; /* time formula string */
int x; /* cell's actual x position */
int y; /* cell's actual y position */
long expr; /* initial formula parse tree */
long iexpr; /* time formula parse tree */
long subsheet; /* subsheet pointer */
long relatcells; /* initial formula reference chain */
long iterationrelate; /* time formula reference chain */
int sequence; /* initial formula calculation natural order */
int isequence;/* time formula calculation natural order */
```

```
} CELL;
```

Function/Procedure PARAMETER Structure

```
typedef struct
{
long name; /* points to the input parameter name string */
double value; /* the value for the above name string */
int type; /* type of the input value */
long nextname; /* points to the next input parameter */
} PARAMNAME;
```

Cell logical Address Structure

```
typedef struct
{
int r; /* row address */
int c; /* column address */
} CELLADDRESS;
```

SHEET Structure

```
typedef struct
{
CELL cells[10][10]; /* all the cells in the sheet. One sheet has only 100 cells in the
current implementation */
long parent; /* points to the sheet's parent sheet */
char title[100]; /* tile for the sheet */
```

```
Window winid; /* window id for the sheet */
Display * dpy; /* display id for the sheet */
GC gc; /* GC for the sheet */
Frame frame; /* window frame for the window that contains the sheet. The above four
 items are specific for the X-Window system */
CELLADDRESS curcell; /* current cell address in the sheet */
CELLADDRESS subsheetcell; /* cell's address that points to the sheet */
int tool; /* current tool in the sheet */
} SHEET;
```

Reference Chain Structure

```
typedef struct
{
CELL * cell; /* points to a reference cell */
long nextcell; /* points to the next reference cell */
int i, j; /* the cell's address */
SHEET * cell-in-sheet; /* the sheet the cell is in */
} CELLLINK;
```

Parse Node Structure

```
typedef struct
{
int type; /* the parse node type */
char op; /* the parse node operator */
long leftexpr; /* points to left parse node */
```

```
long rightexpr; /* points to right parse node */
long elseexpr; /* points to else parse node for the if statement */
} EXPR;
```

FUNCTION Structure

```
typedef struct
{
PARAMNAME * parameters; /* input parameters for function /procedure */
char name[20]; /* function/procedure name */
SHEET * body; /* points to function/procedure body */
} FUNCTION;
```

APPENDIX D

Formula-Execution Procedure

```
void Formula-Execution (input: EXPR * parser-node, SHEET * current-sheet, out
put: RESULT * result)[1]
switch(parser-node->type)
{
case BOOLEAN: /* data type can be text, number and boolean */
result->value = the boolean value. (either 1 or 0)
break
case NUMBER:
result->value = the number value.
break
case NAME: /* use name to reference a cell */
searches in current-sheet or the super sheet of the current sheet,
if found and the cell has value
result->value = the found cell's value
else
error message
break
case ABSOLUTE:
case RELATIVE:
case ABSOLUTE-RELATIVE:
searches in the current-sheet
if the cell has value
result->value = the found cell's value
else
error message
break
```

1. EXPR is defined data structure for parse node, RESULT is defined data structure for cell's result in 3D structure spreadsheet, see Appendix C.

```
case SHEET NAME. NAME:
case SHEET NAME. ABSOLUTE:
case SHEET NAME. RELATIVE:
case SHEET NAME. ABSOLUTE RELATIVE:
first searches the sheet name along current-sheet's parent pointer, then
searches the referenced cell,
if found and the cell has value
result->value = the found cell's value
else
error message
break
case PARENT.NAME:
case PARENT.ABSOLUTE:
case PARENT.REALTIVE:
case PARENT.ABSOLUTE RELATIVE: /* for @ keyword */
get parent sheet from current-sheet
searches the cell
if found and the cell has value
result->value = the found cell's value
else
error message
break
case LEFT:
case RIGHT:
case UP:
case DOWN:
case HERE:
RESULT * temp
if there is no combination call of the dimensional operators
result->value = the appropriate cell's value in current-sheet
else
Formula-Execution(parser-node->leftnode,current-sheet,result)
break
case PRE:
if there is no combination call of the dimensional operators
result->value = the same cell's value in current-sheet's parent sheet
else
Formula-Execution(parser-node->leftnode,current-sheet->parent,
 result)
break
```

```
case BINARY:
RESULT * left, * right
Formula-Execution(parser-node->left, current-sheet, left)
Formula-Execution(parser-node->right,currentsheet,right)
result->value = left->value   binary operator   right->value
break
case UNARY
RESULT * left
Formula-Execution(parser-node->left,current-sheet,left)
result =  unary operator     left->value
break
case IF:
RESULT * left
Formula-Execution(parser-node->left, current-sheet, left)
if left->value
Formula-Execution(parser-node->right,current-sheet, result)
else
if parser-node->else
Formula-Execution(parser-node->else, current-sheet, result)
break
case RETURN:
Formula-Execution(parser-node->left, current-sheet, result)
tells data manager to modifies the cell's value that calls this function
or the cell that points to the current sheet
break
case FUNCTION:
case PROCEDURE:
Function-Execution(parser-node, current-sheet, result)
break /* if the call is procedure, there is no value in the result */
}
```

APPENDIX E

3D Structured Spreadsheet Parser

The 3DSS parser parses the user input source formulas into internal parse trees.

The parser uses left-to-right top-down recursive-descent method[10]. The parer gets the input token from left to right, generates a parse tree for each formula from bottom-up. The 3DSS grammar is a kind of extended LL[1] grammar. Each rule of the grammar corresponds to a function or a procedure. The function or procedure can be recursively called according to the grammar rules. For detail description of 3DSS formula grammar, see appendix B.

In chapter 3 section 3.5 describes that the formula can be **if** statement, **return** statement and expression. The expression can be a combination of arithmetical expression, logical expression, dimensional operators and function and procedure call. The main entry of the parser determines three branches: if statement, return statements and expression. If the coming formula is an if statement, the parser will call if statement parse function. If the coming formula is a return statement, the interpreter will call return parse function, otherwise the parser will call expression parse procedure. All these three parts use a lexer to get next correct token from the input string. Figure D.1 shows the control of the parser.

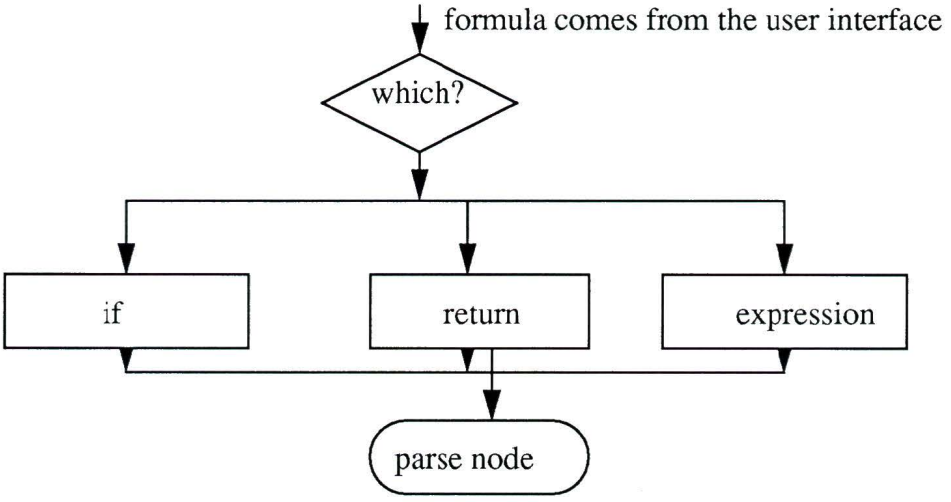

Figure D.1 the main control of 3DSS parser

Parse expression is the basic function of the parser. There are seven expression parse functions, each is responsible for one grammar rule which is organized from top to bottom level. Each function except the last one EXPRE1[1] uses its lower level expression parse function to get appropriate expression node to form a higher level expression node. If statement parse function and return statement parse function use expression parse function to get expression nodes

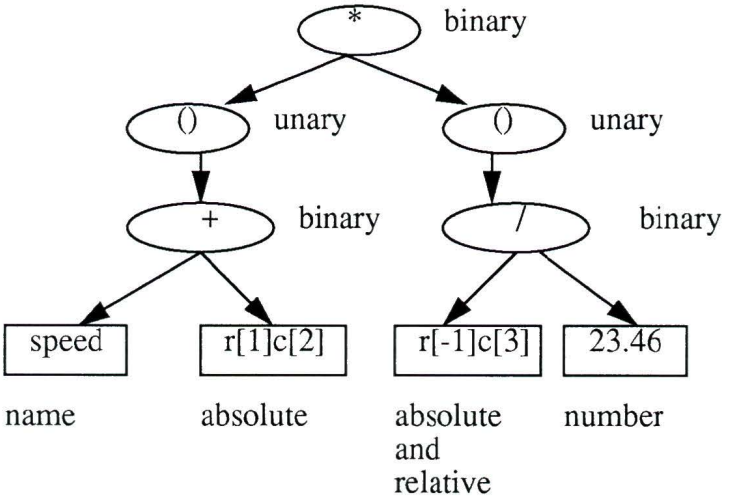

Figure D.2. the parse tree for expression (speed + r[1]c[2]) * (r[-1]c[3] / 23.46)

1. see appendix B

There are many kinds of parse nodes, such as name node, absolute node, relative node, combination of absolute and relative node, binary node, unary node, call node, if statement node and return statement node. A parse tree is composed of these nodes. Figure D.2 shows an example of a parse tree generated by the parser.

VITA

Surname: Wu Given Names: Qian

Place of Birth: Shengyang, China Date of Birth: Dec. 31, 1965

Educational Institutions Attended:

University of Victoria	1992 to 1994
Changsha Institute of Technology	1983 to 1987

Degree Awarded:

B.Sc	Changsha Institute of Technology	1987

Honors and Awards:

Publications:

PARTAIL COPYRIGHT LICENSE

Title of Thesis/Dissertation: 3D Structured Spreadsheet

Author: [redacted]

(Signature)

Qian Wu

(Name in Block Letters)

April 6, 1994

(Date)