Spreadsheet structure inspection using low level access and visualisation

Daniel Ballinger, Robert Biddle, James Noble

School Mathematical and Computer Sciences
Victoria University of Wellington
PO Box 600, Wellington 6001, New Zealand

{db, robert, kjx}@mcs.vuw.ac.nz

Abstract

Spreadsheets are an extremely common form of end-user programming used for many applications from student marks to accounting for global multinationals. Ways of studying the structure of a spreadsheet itself is normally constrained to the tools provided in the spreadsheet software. We wanted to explore ways to use new visualisations for spreadsheets, and this paper documents our approach.

Keywords: spreadsheets, visualisation, end user programming

1 Introduction

When first interacting with a spreadsheet, a user can have to process a daunting amount of information in terms of layout and hidden inter-cell dependencies created by formulas. This problem of the organisation of code has been observed by users for some time and was commented on by Bonnie Nardi [9] where she mentions:

> *It is difficult to get a global sense of the structure of an individual formula that may have dependencies spread out all over the spreadsheet table. Users have to track down individual cell dependencies one by one, tacking back and fourth all over the spreadsheet.*

In this paper we present a solution to this problem through the creation of a set of images that represent the contents at a more abstract level than possible with the spreadsheet software. The user would start with very general information and then progress towards the actual details present in the spreadsheet, all the time learning about the layout and dependency structures without being exposed to actual values and other lower level structures in the spreadsheet. To create these abstract diagrams access to the internals of the spreadsheet, at the level accessible by the user, is required. To avoid the limitations present in the spreadsheet software this information will be extracted to a more versatile programming environment. Java was chosen in the case of this project, as the extraction process will eventually form an integral part of a larger Java based toolkit that will be used to perform analysis of spreadsheets and it is important that it integrate tightly.

2 Accessing the internal structure of a spreadsheet

Excel files are formatted and stored using one of several versions of the BIFF (Binary Interchange File Format) depending on the version of Excel used to save the file. There has already been significant work on documenting this file format (outside Microsoft), including that done by Daniel Rentz as part of the OpenOffice project [12].

2.1 What needs to be extracted

The artefacts required from each Excel file are derived from the low-level structure of the spreadsheets. The basic unit of interest is the cell, the atomic building block of the spreadsheet model. Each occupied cell will have a value that is of some type, such as text or a number. Cells that are to perform a calculation, where the result is assigned to the value, will also contain a formula. The formula extracted should be in exactly the same notation presented to the user when interacting directly with Excel. That is, not the stored Reverse-Polish-Notation (RPN) format or missing any important elements such as absolute reference identifiers. This requirement stems form the need to see the formula as the user would, so the full set of dependencies can be extracted using the toolkit's formula parser.

It is undesirable to build a full extraction utility to read the basic file format from scratch, as that is a complex project in itself.

Due to the limited budget for this research project the API needed to be freeware or open source. Several commercial packages for reading Excel do exist [15].

When looking for a suitable method of converting BIFF encoded files to a representation using Java object structures several technologies and processes were investigated. Each of these brought certain implications, benefits and limitations. All techniques were assessed on the ability to extract the required information, minimal user interaction, platform independence and fault tolerance. Platform independence is important so the extraction process can migrate comfortably with the rest of the Java based toolkit. Fault tolerance is important because some extraction methods encounter difficulty with certain Excel structures and it is important that they tolerate them to allow the toolkit to continue functioning.

The two third party libraries that follow provide the toolkits current extraction abilities.

2.1.1 IBM alphaworks Java Bean

IBM alphaworks have produced the ExcelAccessor Java bean suite [5] to access and modify the contents of worksheets using a Windows DLL. This option was trailed first as it had easy configuration options over the other available methods. The bean usually works well, but the use of native code and requirement of having Excel installed limits its overall portability. Also, it is prone to irregular crashes during larger processing jobs involving multiple spreadsheets. When it does crash it often leaves an Excel process open on Windows 9x with subsequent runs having problems due to the still running process.

In its favour is the ability to correctly extract the exact components of the spreadsheet file that the user sees.

2.1.2 JExcelAPI

An alternative extraction process found uses JExcelApi by Andy Khan [8]. Being pure Java it integrates well with the rest of the toolkit and doesn't require native code. Some element types produce problems with the extraction methods currently available, such as array functions, intersections, absolute references that become relative in formulas, and files using a BIFF from early versions of Excel and the more recent XP version.

2.1.3 Future opportunities

The capability to extract values and string formulas in the user observed format from all occupied cells of the current range of freeware Java Excel readers is improving constantly as most are still actively in development. An example is the Apache POI project [10] where formula support is still being added.

3 Aspects of Spreadsheet structure

The spreadsheet paradigm exhibits two main characteristics: the spatial arrangement of cells on a table and the logical relationships between them created by formulas. These characteristics are not entirely disjoint, with the spatial relationship between cells often having a strong correspondence to the logical dependencies between them. A third, temporal, characteristic is exhibited by a subset of spreadsheets that are long-lived and contain a large number of cells with complex dependencies. Markus Clermont et al. observe that due to the nature of these long lived sheets they can go through similar evolutionary steps as conventional software [3].

Studies have investigated how spreadsheets assist usability.

The benefit of the spreadsheet paradigm is that it allows users to represent their understanding of the problem space on a two-dimensional table. This visual tabular layout is augmented via invisible formulas [6]. While entering raw data and formulas into the spreadsheet, users are shielded from the low-level details of traditional programming [1]. Application domain experts are able to express themselves in a reasonably transparent programming environment in a similar fashion to how they would with pen, paper, and a calculator. In doing this, the developer is naturally attempting to organise the arrangement of a sheet in a way related to its semantics [3].

4 Software support for visualisation with examples

The following figures are examples of images produced by the toolkit. For the purposes of demonstrating how the diagrams are derived from a single Excel workbook the source spreadsheet has been specially constructed using 3 worksheets of hypothetical data and dependencies.

4.1 Real-estate utilisation

When first encountering an unfamiliar workbook it can be useful to get a general orientating view of the layout of all the worksheets.

By counting the number of occupied cells that occur at each spreadsheet coordinate in each worksheet a *real-estate* diagram is created. The main purpose of this diagram is to understand basic positioning information about the workbook. Using this diagram it is possible to quickly get an impression about which sections of the spreadsheet are occupied and to what extent these sections are utilised. This diagram also makes finding sections of data that the designer has purposely tried to obscure easier to find. These obscured sections will often be dragged to a far corner of the sheet and provide functionality that does not need to be seen when using the core of the sheet, such as lookup tables.

In the following diagrams, the cells in the spreadsheet are arranged on the grid lines in their corresponding spatial positions. This produces a diagram with a layout very similar to that presented in Excel. The advantage of this is that relationships understood in one view could more easily be carried over to the view presented in Excel.

Two alternative methods are available to display this information.

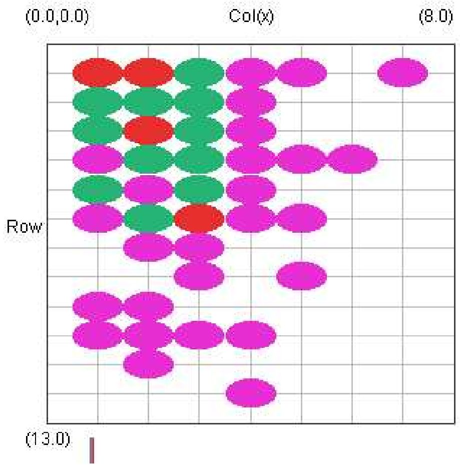


Figure 1. Real-estate utilisation diagram in 2D

The first approach makes use of Java's standard graphics environment to draw grid-based 2D data as in Figure 1.

In this diagram the number of cells that occur at each grid coordinate are counted and displayed using a heat map colour. Grid coordinates where a large number of cells occur will be coloured towards the red end of the colour spectrum will those with lower counts will be coloured towards the violet/blue end. Each row and column in this figure has a 1 to 1 relationship with the rows and columns in the spreadsheet respectively.

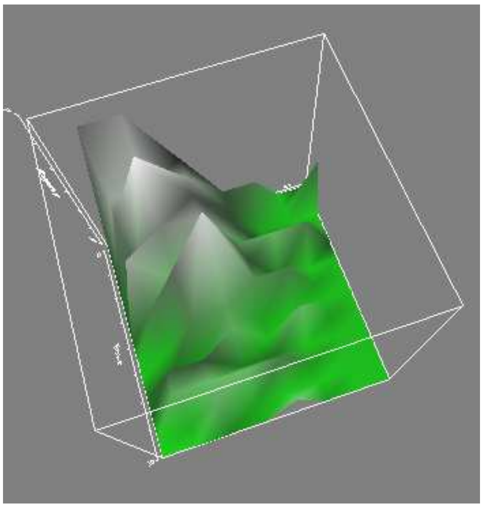


Figure 2. Real-estate utilisation diagram in 3D

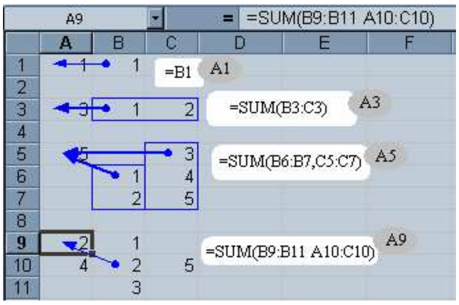
The second real-estate diagram utilises a 3D surface map approach created using Java3D [7] and VisAD [4], a visualisation tool for numerical data. This has the benefit of being interactive by allowing the user to rotate and zoom the image. An example of this type of diagram is Figure 2. In this figure the left axis contains the rows and the upper (obscured) axis the columns. The altitude represents the occupancy level and is coloured to create a terrain type appearance.

4.2 Formulas at a Sheet Level

Once the user understands which parts of the spreadsheet are occupied the formulas that occupy each sheet can be examined.

4.2.1 Referencing cells

Each dependency created by logical references between cells in formulas can be converted to a mathematical vector. The vector between cells has an origin set at the referencing cell and a magnitude that ends the vector to the referenced cell.



A9 = =SUM(B9:B11 A10:C10)

	A	B	C	D	E	F
1	1	1	=B1 A1			
2						
3	3	1	2	=SUM(B3:C3) A3		
4						
5	5		3	=SUM(B6:B7,C5:C7) A5		
6		1	4			
7		2	5			
8						
9	2	1		=SUM(B9:B11 A10:C10) A9		
10	4	2	5			
11		3				

Figure 3. Sheet 1 with a precedents trace

Figure 3 shows the first sheet in the example workbook with precedent trace arrows added by Excel's built in auditing tools. We have added boxes by hand containing the formula present in the space immediately to the right of each cell. This particular sheet shows the 4 main techniques that a user can embed in a formula to reference other cells. Notice how Excel's trace is somewhat deceptive in the case of an intersection, in this case looking more like an individual cell reference.

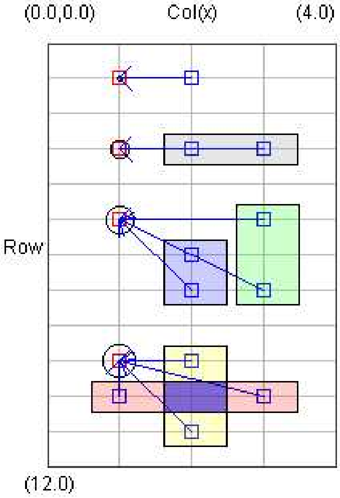
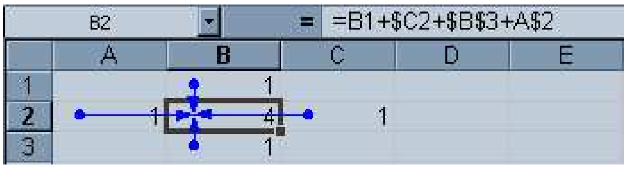


Figure 4. Formula Extracted from Sheet 1

In Figure 4, created by the toolkit, a precedent cell dependency is represented by a blue arrow in a similar fashion to that present in Excel. Each cell containing a formula has a circle that represents the length of the formula as a string and hence the intersection has a larger circle than the union. Any ranged references are shown using a shaded box. A single range is shaded light grey while a union has the left and right sub-ranges coloured blue and green respectively. Intersections use yellow and red boxes for the left and right sub-ranges. The actual resulting intersection is shaded dark blue.

4.2.2 Absolute and relative references



B2 = =B1+$C2+$B$3+A$2

	A	B	C	D	E
1		1			
2	1	4		1	
3		1			

Figure 5. Sheet 2 with precedents trace

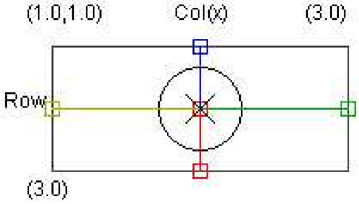
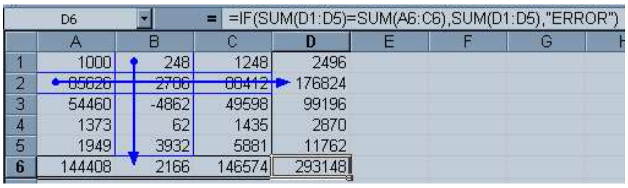


Figure 6. Formula Extracted from Sheet 2

Figure 5 and 6 demonstrate the colouring of relative and absolute cell references. The reference from B1, coloured blue, is a standard relative reference. B3 has a red absolute reference. A2 and C2 are both partially absolute references, in that only one axis is fixed with the $ symbol, and are coloured yellow and green respectively.

The distinctive colouring of these different styles of referencing should help in identifying patterns on large more complex sheets.

4.2.3 A more realistic Summation example



D6 = =IF(SUM(D1:D5)=SUM(A6:C6),SUM(D1:D5),"ERROR")

	A	B	C	D	E	F	G	H
1	1000	248	1248	2496				
2	85626	2786	88412	176824				
3	54460	-4862	49598	99196				
4	1373	62	1435	2870				
5	1949	3932	5881	11762				
6	144408	2166	146574	293148				

Figure 7. Sheet 4 with precedents trace

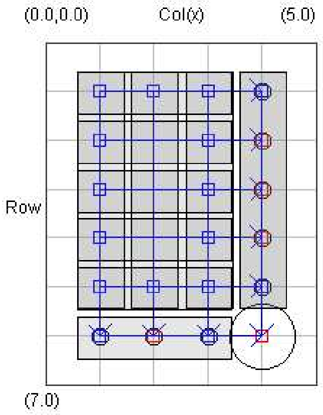


Figure 8. Formula Extracted from Sheet 4

Figure 7 and 10 demonstrate a more realistic style of worksheet where a series of columns are summed and then crosschecked with the sum of the rows. This style of image is similar to the static global view in [6].

4.3 Related Work

The Holy Grail in spreadsheet research at the moment appears to be the admirable goal of detecting and correcting errors [1, 2, 3, 6, 9, 11].

Margaret Burnett is continuing the development of a visual testing methodology to complement the visual programming environment that spreadsheets use [2]. The motivation for this comes from the high degree of errors that are often present in spreadsheets.

Takeo Igarashi presented a series of techniques to address the hidden dataflow graphs and superficial tabular layouts of spreadsheets in [6]. They were designed to improve the users understanding of the dataflow structure by enabling them to visually interact with the obscured structures. He mentions that there is a need to integrate these diagrams into a more realistic spreadsheet program. We have addressed this by allowing the diagrams to be generated free from the application.

Jorma Sajaniemi has presented a theoretical model of spreadsheets along with a description of various spreadsheet auditing mechanisms employing the model [13]. Our toolkit could be used to support such mechanisms.

5 Conclusions

This paper details the creation of a tool to first extract artefacts from spreadsheets and then convert this basic information into visualisations. Its purpose is to augment that provided already by Excel and make it easier to then create new diagrams.

The extraction methods and basic visualisations presented in this paper are encompassed in a larger suit of tools intended to be used for corpus based visualisations.

References

[1] YIRSAW AYALEW, MARKUS CLERMONT, ROLAND T. MITTERMEIR. (2000): Detecting Errors in Spreadsheets.

[2] M. BURNETT, A. SHERETOV, B. REN, G. ROTHERMEL. (2002): Testing Homogeneous Spreadsheet Grids with the 'What You See Is What You Test' Methodology, IEEE Trans. Software Engineering.

[3] MARKUS CLERMONT, CHRISTIAN HANIN, ROLAND MITTERMEIR. (2002): A Spreadsheet Auditing Tool Evaluated in an Industrial Context. http://www.sysmod.com/eusprig02.htm

[4] BILL HIBBARD. VisAD, Java component library, Space Science and Engineering Center - University of Wisconsin - Madison. http://www.ssec.wisc.edu/~billh/visad.html

[5] IBM alphaWorks (1999): ExcelAccessor bean suite. http://www.alphaworks.ibm.com/ab.nsf/bean/ExcelAccessor

[6] TAKEO IGARASHI, JOCK MACKINLAY, BAY-WEI CHANG, POLLE ZELLWEGER. (1998): Fluid Visualization of Spreadsheet Structures, IEEE. http://www.parc.xerox.com/pollez/papers/fluid-spread-vl98.pdf

[7] Java 3D™ API, Sun Microsystems, Inc. http://java.sun.com/products/java-media/3D/

[8] ANDY KHAN. JExcelAPI - A Java API to read Excel 97 spreadsheets. http://www.andykhan.com/excelread/index.html

[9] BONNIE A. NARDI. (1993): A Small Matter of Programming: Perspectives on End User Computing, MIT Press.

[10] ANDREW C. OLIVER, NICOLA KEN BAROZZI. (2002): HSSF (Horrible Spreadsheet Format) part of POI, Apache Software Foundation. http://jakarta.apache.org/poi/hssf/index.html

[11] RAYMOND D. PANKO. (1998): What We Know About Spreadsheet Errors, Journal of End User Computing.

[12] DANIEL RENTZ. (2002): OpenOffice.org's Documentation of the Microsoft® Excel File Format, OpenOffice.org. http://sc.openoffice.org/

[13] JORMA SAJANIEMI. (1998): Modeling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization, Journal of Visual Languages and Computing.

[14] TONY SINTES. (2001): It's Excel-lent - Read MS Excel files with Java, JavaWorld. http://www.javaworld.com/javaworld/javaqa/2001-06/04-qa-0629-excel.html

[15] ACTUATE (2002): Formula One e.Spreadsheet Engine. http://www.actuate.com/f1/products/espreadengine.asp