Programming Communication with the User in Multiplatform Spreadsheet Applications

Jerzy Sikora[1], Jacek Sroka[2], and Jerzy Tyszkiewicz[2](✉)

[1] Institute of Archaeology, University of Lodz, Łódź, Poland
jerzy.sikora@uni.lodz.pl

[2] Institute of Informatics, University of Warsaw, Warsaw, Poland
{sroka,jty}@mimuw.edu.pl

Abstract. It is quite common that the same person uses many different devices, depending on the situation: smartphones and tablets in the field, laptops in the office, switching between operating systems and Web-based applications. A spreadsheet user in this situation needs a *multiplatform spreadsheet*, one which will work equally well on all types of devices. The alternative of having many spreadsheets and copying data between them is clearly inferior, because it is a well-known source of errors.

The topic we want to address in the present paper is programming the interaction with the user in a multiplatform spreadsheet, using only the core spreadsheet functionalities, which are implemented in the majority of spreadsheet systems.

We report here on our experiences with creating the user interface of a multiplatform spreadsheet application for archaeologists working in the field.

1 Introduction

1.1 Early History

The initial challenge came from one of the authors of the present paper (J. Si.), who needed a mobile application for Android, capable of storing stratigraphic data collected during excavations, consisting of textual descriptions of archaeological contexts, and chronological *earlier-than* and *later-than* relations between them. The data was intended to be transferred for further analysis to a standard, Windows-based application (running under Wine on a Linux machine). So from the very beginning the application was intended to be used in an environment with at least two (or perhaps even three) operating systems involved. This is nothing particular. Nowadays users routinely switch between devices, operating systems, and between online and offline mode of work.

We decided to implement the archaeological application, later named *Strati5*, as a multi-platform spreadsheet. The application was described from the user's point of view in [10]. We also published a short, nontechnical paper [9], advocating the general idea of rapid development of mobile multiplatform applications in the form of spreadsheets with Strati5 as a working example.

P. Milazzo et al. (Eds.): STAF 2016, LNCS 9946, pp. 356–371, 2016.
DOI: 10.1007/978-3-319-50230-4_27

1.2 Why a Spreadsheet?

In the paper [11] the other two of us (J. Sr. and J. Ty., with other co-authors) made a claim, that spreadsheet formulas in fact constitute a platform-independent programming language, even though there is no common formal standard in this respect. Applications written in this language run on virtual machines, which are spreadsheet management systems, like *Microsoft Excel* (available for Windows desktop and phone, Mac, Android and iOS), *Apache OpenOffice* and *LibreOffice calc* (available for Windows, Mac and Linux), *WPS Office* spreadsheet (available for Windows desktop, Linux, iOS and Android), and many other.[1]

When we had a chance to verify our belief in practice, we did so, treating the archaeological application as a test, if spreadsheet technology can indeed serve for multi-platform programming.

Next, using spreadsheet as an application saved us a lot of implementation work. Programming the user interface is typically one of the most laborious parts of each project. In our case, a vast majority of the user interface is always provided directly by the spreadsheet system used. It is responsible for all functionalities related to data navigation, typing, editing, undo, redo, file opening and saving, etc. It also adapts the application to different screen sizes and resolutions, mouse or touch as a pointing device, etc. Finally, it seems almost impossible to find another technology which would make the very same code run on Android, iOS, Windows, Linux and MacOS.

Having one application for all systems, we did not have to implement any protocols to facilitate data transfer between different applications. Copying data between spreadsheets is generally considered to be error-prone, so sharing the same spreadsheet between devices and systems prevents many potential errors and risks.

Last but not least, spreadsheet technology is very conservative and backward compatibility has always been a major concern. Therefore, we expect our application to remain fully functional for many years without any need of modifications, and even to get ported to new operating systems, should they appear on the market—most likely without a single keystroke on our part.

1.3 Development History

After testing a few systems, WPS Office for Android was chosen as the optimal one to start with and within a few days the first working version of Strati5 for the tablet was available.

Tests took place during regular archaeological excavations in Ostrowite (Pomerania, Poland) led by J. Si. A number of improvements resulted in a tool, running on Android tablet (for collecting data in the field) and on Linux laptop

[1] A *Spreadsheet management system* (or spreadsheet system) is a software used to create, manage and execute individual *spreadsheets*. This distinction resembles the relation between a *database management system* and individual *databases*.

at the base camp (for exporting the data to an external application). The whole spreadsheet was routinely used and transferred between the devices, causing no problems during many months of excavations. Strati5 proved to be reasonably comfortable and intuitive in everyday operation.

Then we decided to offer the application to other users, making it as independent of the operating system as possible. Achieving this goal required a number of changes and appeared to be an interesting programming experience.

In the present paper we discuss the technical issues of programming the interaction with the user in a spreadsheet intended to be transferred between many platforms. We hope that this knowledge and developed know-how will be useful in other cases. Requests for help in porting existing spreadsheets into mobile environments already show up at MrExcel.com, a very active spreadsheet-related forum, as witnessed by recent posts [1,3,7,8]. We expect that this demand will grow. There have been a few papers which deal with the usability of mobile spreadsheets [2,4,5], but they all discuss spreadsheet management systems rather than individual spreadsheets, so we are probably pioneers in this respect.

As a by-product of our technical developments, in Sect. 3.2 we describe a simple design pattern, which can be used to control location of cyclic references in all types of Excel: desktop, mobile and online.

2 The Overall Structure of the Interface – Strati5

In this section we describe the spatial organization of the interface and spreadsheet functionalities necessary to implement it. Our working example is Strati5.

2.1 Fundamental Requirements

The requirement was to develop a mobile application, capable of storing stratigraphic data, consisting of archaeological contexts with textual descriptions, a set of chronological *earlier-than* and *later-than* relations between the contexts, which are edges of a directed acyclic graph of interest to archaeologists, called Harris matrix. Later we added groups, i.e., named sets of contexts, to the data model. We identified the following main requirements:

A. Entry and storage of the data, with estimated growth rate of at most 20 records a day and average of 200 records per month.
B. Warn about/prevent duplicated context id.
C. Warn about/prevent cycles in the relations between contexts.
D. Warn about/prevent using undefined contexts in the relations.
E. Warn about/prevent duplicate group id.
F. Warn about/prevent assigning contexts to undefined groups.
G. Operating on a mobile devices and laptops, with strong preference toward network-independent operation.
H. Automatic or semi-automatic data export in a format accepted by *Stratify* [6], a popular free desktop application for maintaining and processing stratigraphic data.

I. Low resource consumption, to enable smooth operation on a tablet and prevent draining the battery, assuming load from item A above.

As it can be recognized, most of the above items were related to data validation. While analyzing the methods to implement them, we came up with the conclusion, that **it was impossible to support all popular spreadsheet systems**.

Instead, we decided to work toward a more realistic goal of **supporting sufficiently many spreadsheet systems to offer our tool on all major operating systems, and to cover a few most important spreadsheet systems**: Microsoft Excel for Windows desktop and Mac, LibreOffice and Apache OpenOffice for Windows, Linux and Mac, and Google sheets.

2.2 Structure of the Interface

After some consideration, we decided that the key services we needed from the spreadsheet systems were:

– Reporting emergence of cyclic references (required for reporting cyclic relations between contexts);
– Support for array formulas;
– Support for "freeze panes" (i.e., making the top row(s) and/or the leftmost column(s) always visible on the screen);
– Support for data validation by a list of allowed values;
– Support for conditional formatting.

They were needed for a very schematic idea of the interface, shown on Fig. 1. It was based on the *freeze panes* function applied to a few rows and columns.

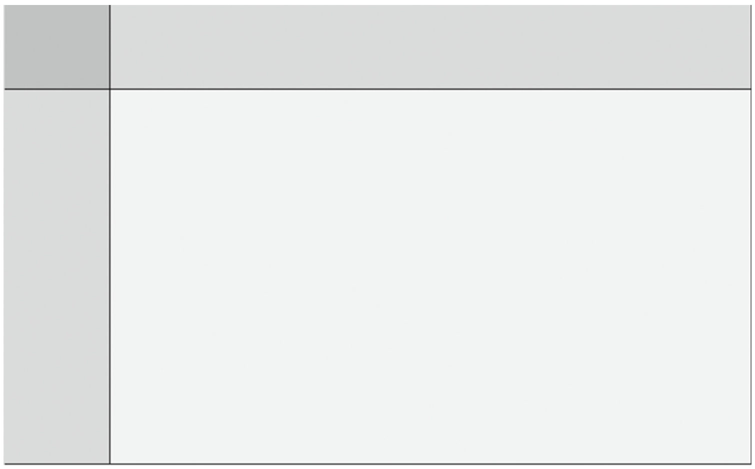


Fig. 1. Schematic structure of the user interface. The gray top row(s) and leftmost column(s) are frozen panes.

Referring to the colors on the figure, the roles of the particular areas are as follows:

– The dark gray area in the top left corner is always present on the screen. It can display messages about spreadsheet's global state, by formulas producing texts of messages and conditional formatting indicating their presence.
– The medium gray area on the top edge consists of cells which are visible whenever any fragment of their column is visible. They can be used to display column-related messages.
– The medium gray area on the left edge consists of cells which are visible whenever any fragment of their row is visible. They can be used to display row-related messages.
– The light gray area are individual cells and the functions used in them are most likely data validations and perhaps conditional formatting.

Eventually, the following operating systems and spreadsheet systems passed tests of compliance with the above requirements:[2]

Windows Excel, LibreOffice, OpenOffice, WPS Office
MacOS X Excel, LibreOffice, OpenOffice, WPS Office
Linux LibreOffice, OpenOffice, WPS Office
Android WPS Office
iOS WPS Office
Windows mobile MS Office
All systems Google sheets.

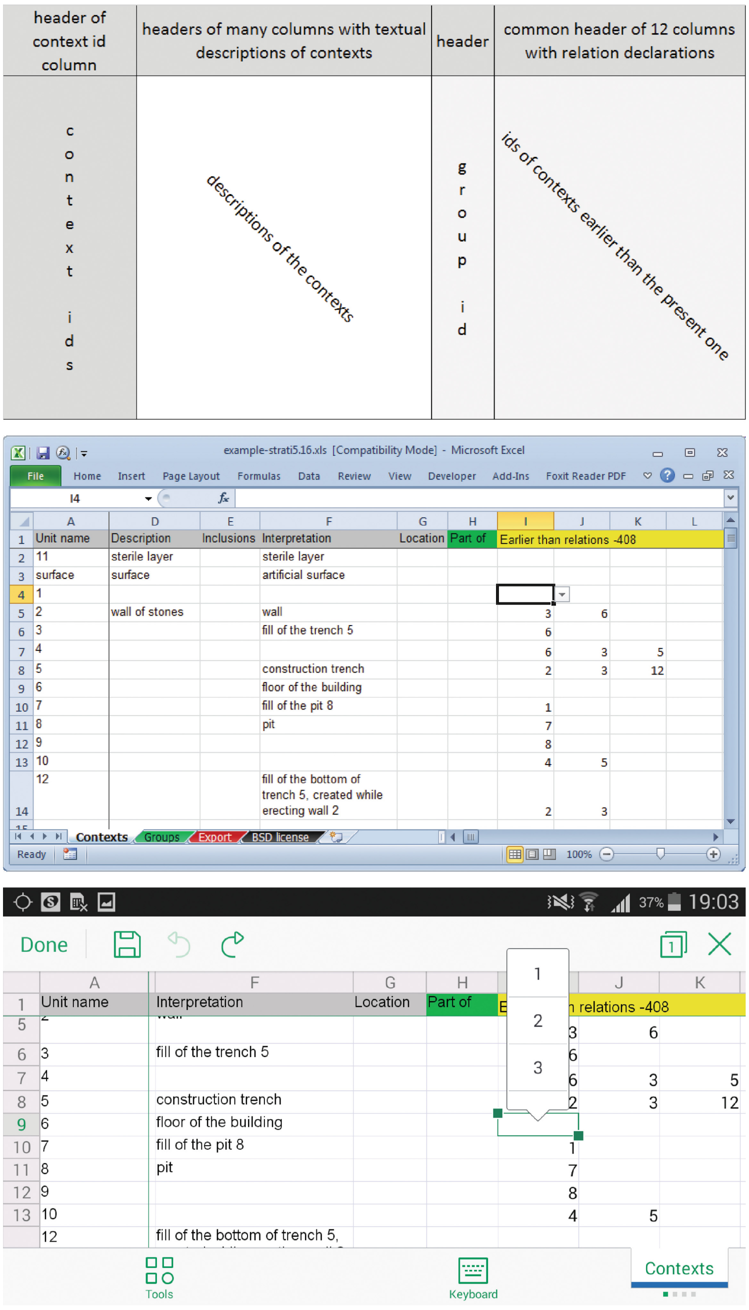
3 Interaction with the User

In this section we point out the main issues we encountered while designing and programming the way Strati5 interacts with the user.

First of all, we separated the spreadsheet into a number of worksheets. The bulk of user interaction is performed on worksheet intended for data entry,[3] whose structure is presented in Fig. 2 below and follows the schema from Fig. 1. The headers of rows do not produce any messages, but contain the id of the archaeological context being described in the row. Column headers are used to display column-related messages. The *descriptions of the contexts* area contains no validations, as it is intended for textual descriptions. The corresponding column headers are merely names of columns.

The requirements presented above determine, that the dominating activity was to either block entries violating data integrity, or at least to warn the user about them. The formulas of Strati5 we present below come from a variant with 200 contexts, at most 500 contexts and relations together, at most 12 relations per context and at most 20 groups.

[2] Some other combinations might be fully functional, too.
[3] In fact, the remaining worksheets are almost never used in the field, and relatively seldom at the base camp.


header of context id column
headers of many columns with textual descriptions of contexts
header
common header of 12 columns with relation declarations
context ids
descriptions of the contexts
group id
ids of contexts earlier than the present one
example-strati5.16.xls [Compatibility Mode] - Microsoft Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Add-Ins
Foxit Reader PDF
I4
Contexts
Groups
Export
BSD license
Ready
100%
37%
19:03
Done
1
2
3
Tools
Keyboard
Contexts

	A	D	E	F	G	H	I	J	K	L
1	Unit name	Description	Inclusions	Interpretation	Location	Part of	Earlier than relations -408			
2	11	sterile layer		sterile layer						
3	surface	surface		artificial surface						
4	1									
5	2	wall of stones		wall			3	6		
6	3			fill of the trench 5			6			
7	4						6	3	5	
8	5			construction trench			2	3	12	
9	6			floor of the building						
10	7			fill of the pit 8			1			
11	8			pit			7			
12	9						8			
13	10						4	5		
14	12			fill of the bottom of trench 5, created while erecting wall 2			2	3		

	A	F	G	H		J	K
1	Unit name	Interpretation	Location	Part of	E	relations -408	
5	2	wall			3	6	
6	3	fill of the trench 5			6		
7	4				6	3	5
8	5	construction trench			2	3	12
9	6	floor of the building					
10	7	fill of the pit 8			1		
11	8	pit			7		
12	9				8		
13	10				4	5	
	12	fill of the bottom of trench 5,					

Fig. 2. The structure of the data entry worksheet of Strati5 (top). The colors follow Fig. 1. There are no global state messages. The cells in the white areas undergo no validation. Below Strati5 in Microsoft Excel under Windows 7 desktop (middle) and in WPS Office under Android on a smartphone (bottom).

Requirements D and F can be very efficiently programmed using data validation of type "List", where the user has to choose the data element from a list declared in the data validation form. This form of data validation is widely supported by spreadsheet systems. On a majority of systems, the spreadsheet displays a drop-down list of permitted values, from which the user can choose the right one. It is particularly convenient on mobile devices, where typing is much less comfortable and produces more typos than on machines with hardware keyboards. Such lists are used in the *ids of contexts earlier than the present one* and *group id* areas. The drop-down list can be potentially quite long, but we did not decide to do anything about that.

Concerning requirements B and E, the only method to block a new data entry in spreadsheets is to use data validation tools. Initially we intended to use them in the *context ids* area. It is possible and not too difficult to write a custom formula in the Excel data validation form, which will permit entering only a new context id in a cell. However, data validation in Apache OpenOffice and LibreOffice does not permit custom formulas. Worse still, upon reading an Excel-created file with such a data validation, it is corrupted and produces a faulty data validation, which permits duplicates, but blocks completely some entries. Consequently, we submitted bug reports and had to choose another method to satisfy our need.

Being unable to block duplicates, we decided to indicate them instead, leaving corrections to the user. There are two tools, which can be used for this purpose: conditional formatting and formulas. The former can be used to mark duplicates. However, Google sheets does not support marking duplicates, so the only way was to enter a custom formula there. After the unpleasant experience with formula-based data validation, we decided to use the *header of context id column* cell to insert a single array formula, verifying if there were any duplicates on the list below. It was written to yield a warning message if there are duplicates, and the usual column header otherwise:

```
A1 {=IF(MAX(COUNTIF(A2:A200,A2:A200))>1,"DUPLICATE","Unit name")}
```

A single simple conditional formatting rule applied to this cell turns it red when the warning is displayed, to increase its visibility. This form is sufficient, because in archaeological practice, context ids are created, stored and almost never changed. Consequently, if a new entry is a duplicate, it is clear that this entry must be changed, not the other. Very recently an update of LibreOffice introduced a faulty mechanism of array formula computation, which in our case produces a constant duplicate warning. Therefore we decided to replace the array formula by a formula-based conditional formatting in column `A`.

3.1 Reporting Emergence of Cyclic References

This was probably the source of the largest problem we had in the development of Strati5. Our implementation of breadth-first-search (BFS) graph traversal (presented for completeness in Appendix A) produces a cyclic reference between spreadsheet cells as a manifestation of a cycle in the "earlier than" relation created by the user. Therefore, we needed to notify the user of this event. Unfortunately, the way spreadsheet management systems react on cyclic references is very diverse. We have noted the following basic groups:

pop-up Displaying a pop-up window with an appropriate message. This form is exhibited by Microsoft Excel for Windows desktop and Mac OS, WPS Office for Windows and for Android. Additionally, the cells which are lying on the cycle, as well as those which depend on them, are not recomputed.

error Evaluating the cells which are lying on the cycle to an error value. This form is exhibited by OpenOffice, LibreOffice and Google sheets.

stop The cells on the cycle, as well as those which depend on them, are not recomputed. This form is exhibited by Microsoft Excel for Windows Mobile, Microsoft Excel Online, WPS Office for iOS.

Systems exhibiting **pop-up** message notify the user themselves.

For **error** group we used the top row to display the message. The common header of the columns where the user was supposed to enter the ids of later contexts were a single cell merged from several individual ones, whose text was constructed by a formula of the form `=IF(ISERROR(SUM('Cycle test'!I2:I500)),"Cycle detected!", "Earlier than")`. The column `I` on the worksheet `Cycle test` is the place where cyclic references, and consequently, error values, emerge (see Appendix A). `SUM` function evaluates to an error if there are any errors in the summation area, and to a number otherwise. The whole formula thus produces the message, whose visibility is increased by conditional formatting. This gave us a common solution for **pop-up** and **error** systems.

Supporting **stop** was very important for us, because all available spreadsheet management systems for iOS and Windows Phone were of this type.

We divided the header into two cells. The first of them contains the formula

```
=IF(ISERROR(SUM('Cycle test'!I2:I500)),"Cycle detected!",
"Earlier than relations"&SUM('Cycle test'!H2:H500)),
```

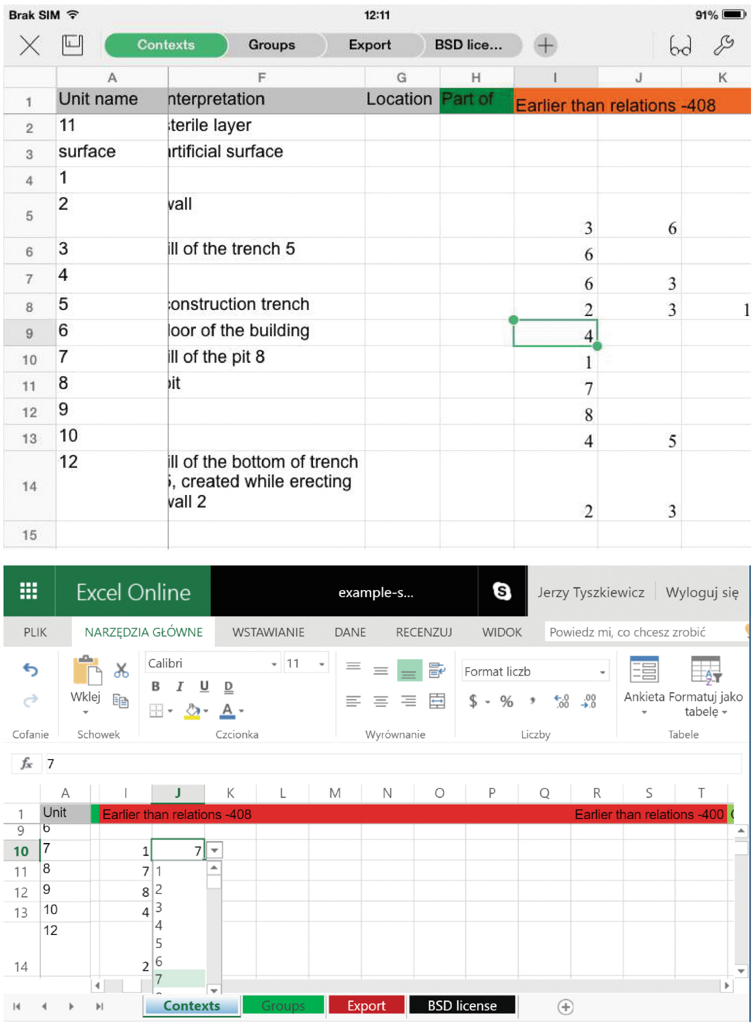
while the other cell contains

```
="Earlier than relations"&SUM('Cycle test'!H2:H500)).
```

Then conditional formatting is applied to both cells, turning them red if their values differ.

The cells in the range `'Cycle test'!H2:H500` contain numbers related to each tuple in the *earlier-than* relation such that any *single* change of the relations causes a change of their sum. `'Cycle test'!I2:I500` contains distances of the contexts from the sterile layer and is, as before, the place where cyclic references emerge.

In systems with **pop-up** and **error** responses the new formulas work very much as before. In systems with **stop** response, the two results are obviously equal if there are no cyclic references. However, if the user adds to, or modifies a tuple in the relation creating a cyclic reference, the system stops the evaluation of some cells in the range `'Cycle test'!I2:I500`. One of the header formulas depends on this range, the other does not. The latter is computed normally and its value changes. The former is not recomputed due to its dependence on unevaluated cells. Crucially, after the recomputation is finished, conditional



	A	F	G	H	I	J	K
1	Unit name	nterpretation	Location	Part of	Earlier than relations -408		
2	11	terile layer					
3	surface	rtificial surface					
4	1						
5	2	vall			3	6	
6	3	ill of the trench 5			6		
7	4				6	3	
8	5	onstruction trench			2	3	1
9	6	loor of the building			4		
10	7	ill of the pit 8			1		
11	8	it			7		
12	9				8		
13	10				4	5	
14	12	ill of the bottom of trench ;, created while erecting vall 2			2	3	
15							

fx 7

	A	I	J	K–Q	R	S	T
1	Unit	Earlier than relations -408			Earlier than relations -400		
9	6						
10	7	1	7				
11	8	7	1				
12	9	8	2				
13	10	4	3				
14	12	2	4 5 6 7				

Fig. 3. Strati5 in WPS Office under iOS (top) and in Excel Online (bottom), showing a warning about cyclic references. They are **stop** systems, so the indication is the red color of the header. The bottom screenshot shows the the second cell of the header, with a different value in it, triggering the conditional formatting.

formatting is applied to all cells, irrespectively of their evaluation status and the two cells become red and issue a visual warning to the user, although no text message is produced.

3.2 By-Product: A Cycle Indicator

A by-product of the above mechanism is our construction of a universal detector of cyclic references in Excel desktop and online.

Excel desktop reports the emergence of cyclic references by a pop-up window and indicates one, more-or-less randomly selected cycle, among all that are created. This gives access to one cycle at a time. If such references are not enabled, but already present somewhere in the workbook, there will be no warning or indication of the subsequently created ones. This is especially likely if formulas with `OFFSET` or `INDIRECT` are used and edited, but `INDEX` function used in the reference form can cause the same effect.

Excel online does not report emergence of cyclic references in any way and have no error checking tool.

The design pattern described below allows the user to set up cycle warnings for any number of cells. All of them are activated simultaneously and visually indicate ones which lie on cycles or depend on such cells. The form presented here works for cells which do not evaluate to error values, but this can be overcome with minor modifications.

As a matter of example, let us assume that the range `A1:A10` are the cells suspect of becoming elements of cycles we want to monitor.

We set up a single reference cell, let it be

$\boxed{\texttt{C1}}$`=NOW()`

The monitors are installed by formulas (we assume that the formula is entered into the top cell of the range and copied down, with automatic modifications introduced by the spreadsheet)

$\boxed{\texttt{B1:B10}}$`=IF(A1=A1,$C$1)`

and conditional formatting rules applied to cells in `B1:B10`, so that they change the formatting of cell `Bi` iff its value is **smaller** than the value of the reference `C1`.

Let us assume the user makes an edit, which might cause some of the cells to become members of cycles themselves or to depend on cells which are now on cycles. Then the following events happen:

– Some of the cells in `A1:A10` are attempted to be recomputed. Ones which are now on cycles are not recomputed and the recomputation of their dependents, including those in `B1:B10`, is also blocked[4].
– The formula in `C1` is volatile, hence it is recomputed and its value increases.
– Formulas in `B1:B10` depend on cell `C1` whose value has changed, hence their recomputation is attempted. It succeeds for `Bi` iff `Ai` is not on a cycle and does not depend on a cycle, otherwise it is blocked and the old timestamp is retained.

[4] Excel for Android recomputes all cells which are not elements of cycles, even if they depend on cells which are elements of cycles. Therefore our solution does not work in Excel for Android.

– Conditional formatting is applied to all cells in `B1:B10`. Those which have been recomputed are equal to `C1` and are not formatted; those which have not been recomputed contain a timestamp older than the present value of `C1` and are therefore formatted.

A small variation allows a single monitor to be applied to several cells, e.g.,

```
B1 =IF(COUNT(A1:A10)=COUNT(A1:A10),$C$1)
```

collectively monitors the whole range `A1:A10`.

This design pattern incurs only low computational overhead and does not require modifying the monitored spreadsheet computation. In particular, it can be easily removed, when it is no longer needed.

3.3 Data Export

Requirement H was to provide data export from Strati5 to Stratify [6]. This tool can read csv files of a specific structure, defined by the number of columns, their data types and headers. The solution we implemented was to produce such a csv file by concatenating values of certain cells, in a worksheet intended for data export. The user is supposed to either copy and paste its content into a text editor, or save it directly from the spreadsheet system as a csv file. Subsequently Stratify can import such a file.

3.4 "Soft" Methods to Reduce Resource Consumption

We used two "soft" tricks to reduce the resource consumption of Strati5, both related to the interaction with the user. This was done to satisfy requirement I.

The first one is that the user, while describing a context, is allowed to specify only contexts which are later than the present one, while relations in both directions make perfect sense in archaeology, and Stratify permits them to be specified. Specifying that a context c is earlier than the presently edited context d can still be done: by going to the row with context c and entering there d as a later context. This way we avoided expensive sorting by spreadsheet formulas to group the tuples of the *earlier-than* relation by the first coordinate, which is crucial for our implementation of BFS.

The next trick is that we introduced two predefined contexts: the chronologically oldest context "sterile layer" and the top context "surface", which are crucially *not processed* in the acyclicity test (but are processed in the duplicate-freeness test). The typical structure of many archaeological sites causes these two contexts to be present in a very large fraction of the *earlier-than* tuples, while they have no impact on the cyclicity of the relation. By eliminating them, we get a significant reduction of the number of tuples Strati5 must process.

We expect that in many other contexts, domain-specific knowledge about data to be processed can help devising analogous layout solutions to reduce the computational cost of the spreadsheet application.

4 Scalability Problem

This is a problem we did not find any good solution for. Spreadsheets come always with certain fixed number of rows and columns of formulas, and thus are capable of processing a predefined maximal number of data items. Too small a spreadsheet is therefore bad, too large one slows down the applications and drains the battery—the opposite of requirement I. The designer has two basic methods to overcome this problem: either to produce a couple of spreadsheets of different sizes and let the user transfer the data between them when necessary (which is error prone), or to assume that the user will modify the spreadsheet, adding or removing rows of formulas. Preparing the spreadsheet for the latter action is not trivial and requires a good deal of additional design work. It also interferes with the good practice of hiding or locking those portions of the spreadsheet which are not supposed to be edited by the end user.

5 Availability

Strati5 is available from http://bit.ly/Strati5, and is an open-source software with a BSD license. Let us note here that, as far as we know, there is no technology to close the code of a multiplatform spreadsheet application, so multiplatform spreadsheets are open source by necessity.

6 Standardization Issues

We were asked by the anonymous reviewers to discuss the issue of standardization between spreadsheet management systems.

The following list indicates the main difficulties encountered while developing our multiplatform spreadsheet:

a. Limited mutual compatibility of spreadsheet systems.
b. Highly insufficient documentation.
c. Technical and legal problems with using SDK tools for mobile systems.

It is clear that the more functionalities are compatible between spreadsheet systems, the easier is to program multiplatform spreadsheets. Issues a and b are two sides of the same coin. Definitely, they are real: during the whole development process of Strati5 we had problems with mutual compatibility between spreadsheet management systems and with their (lack of) documentation. Therefore we had to rely on experiments choosing solutions in many cases, and we could not use SDK tools for that.

A lot of work was necessary to find a workaround (not a solution!) of the problem that cyclic references, a fundamental property of a spreadsheet, are reported in so many different ways. Even spreadsheet systems coming from the same vendor differ in this respect: Microsoft Excel for Windows (pop-up) differs from its Online and Windows mobile versions (stop), WPS Office for Android and Windows (pop-up) differ from WPS Office for iOS (stop).

Cross-vendor problems concern other fundamental issues: LibreOffice and OpenOffice do not permit formula-based data validations. Even the syntax and behavior of formulas tends to differ from vendor to vendor like, e.g., error encoding and handling between various Excel variants and LibreOffice on the one hand and OpenOffice on the other hand.

The methods to create references to external workbooks are not transferable between spreadsheet systems, in particular if one wants to use `INDIRECT` or some other programmable, foolproof mechanism.

Indeed, as commented during the workshop, the relations between different spreadsheet systems and their vendors seem to resemble those between Web browsers and their vendors during the browser wars, with similar consequences for the users and programmers.

A significant problem is related to the performance of the whole systems and of their particular functions. Working with hardware of potentially low performance, the programmer must know how spreadsheet systems implement specific functions.[5] E.g., in desktop Excel `=MATCH(val,rng,0)` is of linear time complexity, while `=MATCH(val,rng,1)` is logarithmic, assuming that `rng` is sorted. It would be desired to have guarantees that these complexities carry over to other spreadsheet systems. In desktop Excel `=COUNTIFS(rng1,val1,rng2,val2)` is much faster than `=SUMPRODUCT((rng1=val1)*(rng2=val2))`. Again, one would like to know that the same relation holds in other systems. If many formulas are being evaluated, the ability of the spreadsheet systems under consideration to perform multi-threaded computations becomes an important factor, too.

We can also list two particular functionalities which would be very useful in programming multiplatform spreadsheets.

Widely adopted and extended `INFO` function to determine the identity of the spreadsheet system. That would help in programming spreadsheets in a clean way, with clearly indicated portions of the code to be executed on particular spreadsheet systems. The present functionality of `INFO` is insufficient: it returns the version number of the system, but not its identity. E.g., `=INFO("RELEASE")` returns the same value 11 in the latest WPS Office on Windows and on Android, even though both spreadsheet systems differ in behavior and functionality, and in quite different and much older Excel 2003. The situation is even worse with Excel, whose online and Android versions do not support this function at all.

Row-wide and column-wide conditional formatting – in particular, setting conditionally the width of columns and height of rows. This way one could set a column to be a (part of a) frozen pane and be always present on the screen, fill it with formulas computing warning messages, and set its width conditionally to be 0 if it does not contain any warnings. Effectively, the column would then play the role of a pop-up window produced without any scripts or macros.

Built-in array function sorting its input (like the one present in Google sheets) in $O(n \log n)$ time would be a great tool to reduce resource consumption by many

[5] We would like to thank one of the anonymous reviewers, who has pointed to us the importance of this topic.

algorithms implemented by spreadsheet formulas. The typical sorting algorithms built from formulas are quadratic. There is a spreadsheet sorting algorithm of $O(n \log^2 n)$ time complexity, but it requires $O(n \log n)$ formulas and is quite complex [11]. Reducing resource consumption is particularly important on mobile systems, due to their relatively low performance and dependence on battery.

Acknowledgments. We would like to thank the anonymous reviewers of our paper and all participants of SEMS 2016, whose comments influenced this post-proceedings paper.

The research project in Ostrowite, where J. Si. tested Strati5, was financed by the National Science Centre grant based on the decision 2015/19/B/HS3/02124. The research of J. Sr. was sponsored by National Science Centre grant based on the decision 2012/07/D/ST6/02492.

A BFS by Spreadsheet Formulas

Below we describe our implementation of the cyclicity test, which is based on BFS graph traversal of [11], for Strati5 with the size limits we have indicated. The way it is programmed is important for the generation of messages about cyclicity of the relation.

Initial data is located in the worksheet `Contexts`. The range with contexts is `Contexts!A2:S200`, with two contexts predefined: the top and the bottom layer. The range `Contexts!I4:T200` contains the relations: in row i (i.e., the range `Contexts!I`i : `T`i) contains the list of contexts which are later than the context in cell `Contexts!A`i. The formulas below are located in the worksheet `Cycle test`, which is hidden by default, because it is not intended to be edited by the end user.

Below we indicate ranges and formulas. Each time, if the range consists of more than one cell, we assume that the formula is entered into the top cell of the range and copied down, with automatic modifications introduced by the spreadsheet.

The formulas below ignore rows 1, 2 and 3 of the tab `Contexts`. The first of them contains the headers, the other two contain two predefined contexts: the sterile layer and the present surface, which are the bottom and top contexts in the *earlier-than* relation. We do not process them.

First we count later contexts in each row, to know how many tuples it will produce.

```
A2:A200 =COUNTA(Contexts!I4:T4)
```

Now we compute the incremental sum of the tuples to be created, adding one dummy tuple for each context.

```
B2 =1
B3:B500 =B2+A2+1
```

This is the total number of all tuples:

```
C2 =SUM(B2:B200)
```

And this is the total number of all contexts:

```
D2 =COUNTA(Contexts!A4:A200)
```

Next we produce the number of the row in which the first coordinate of the tuple is located. The count of rows refers to the area starting in row 4 in tab `Contexts`, hence here we start with 1.

```
E2 =IF(ROW()>C$2+D$2,"",1)
E3:E500 =IF(ROW()>D$2+D$2,"",IF(F2>INDEX(A$2:A$200,E2),1+E2,E2))
```

The following is then the number of the column from which the second element of the tuple originates:

```
F2 =IF(E2="","",1)  F3:F500 =IF(E3="","",IF(E2=E3,1+F2,1))
```

Now we import the id of the context, which is the second element of the tuple.

```
G2:G500 =IF(E2="","",INDEX(Contexts!I$4:T$200,E2,F2))
```

At this moment, consecutive rows in columns `E` and `G` contain the tuples of the *earlier-than* relation we should process. In column `E` they are represented by row numbers, in column `G` by real ids. They are grouped: all tuples that share the same value of the first coordinate form a contiguous block.

The next formula searches column with ids of the contexts in sheet `Contexts` to find the position of the context which is the second coordinate in the present tuple:

```
H2:H500 =IF(OR(E2="",G2=0),-1,MATCH(G2,Contexts!A$4:A$200,0))
```

In case of nonexistent tuples (second coordinate `""`) or artificial ones (second coordinate `0`) we produce `-1` without performing the actual search, because `IF` is a lazy function, otherwise `MATCH` does the exact search (third parameter `0`) for the value of `G2` in the range `Contexts!A$4:A$200` and returns the position of the match.

The last formula is the key one. Rows with `-1` in column `H` get value `1` and are the beginning of the recursion. Otherwise $a =$ `INDEX(B$2:B$200,H2)` gives the row number of the beginning of the block of tuples with the first coordinate equal to `G2` (via the value in the previous column), and $b =$ `INDEX(A$2:A$200,H2)` the size of that block. `OFFSET` then creates a range, which starts a rows below and `0` columns to the right of `$I$1`, and spans b rows and 1 column (default value, omitted in the formula). Now `1+MAX` of that range does the recursion. It is well-founded if there are no cycles in the *earlier-than* relation, and results in a cyclic reference in case this relation contains a cycle.

```
I2:I500 =IF(H2=-1,1,1+MAX(OFFSET($I$1,INDEX(B$2:B$200,H2),0,
        INDEX(A$2:A$200,H2)+1)))
```

Therefore the correctness test is really the test if the above formulas produce a cyclic reference or not.

References

1. Bradgar: iPad 2 running Excel with VBA? post #13. http://www.mrexcel.com/forum/excel-questions/607337-ipad-2-running-excel-visual-basic-applications-2.html
2. Chintapalli, V.V., Tao, W., Meng, Z., Zhang, K., Kong, J., Ge, Y.: A comparative study of spreadsheet applications on mobile devices. Mobile Information Systems 2016 (2016). doi:10.1155/2016/9816152
3. CWBlack: apps that support Excel VBA. http://www.mrexcel.com/forum/general-excel-discussion-other-questions/830464-apps-support-excel-visual-basic-applications.html
4. Flood, D., Harrison, R., Iacob, C.: Lessons learned from evaluating the usability of mobile spreadsheet applications. In: Winckler, M., Forbrig, P., Bernhaupt, R. (eds.) HCSE 2012. LNCS, vol. 7623, pp. 315–322. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34347-6_23
5. Flood, D., Harrison, R., Iacob, C., Duce, D.: Evaluating mobile applications: a spreadsheet case study. Int. J. Mob. Hum. Comput. Interact. (IJMHCI) **4**(4), 37–65 (2012)
6. Herzog, I.: Group and conquer - a method for displaying large stratigraphic data sets. BAR Int. Ser. **1227**, 423–426 (2004). http://www.stratify.org
7. kgkev: VBA & Mobile devices. http://www.mrexcel.com/forum/general-excel-discussion-other-questions/930944-visual-basic-applications-mobile-devices.html
8. QCMan: IPad and desktop. http://www.mrexcel.com/forum/excel-questions/923376-ipad-desktop.html
9. Sikora, J., Sroka, J., Tyszkiewicz, J.: Spreadsheet as a multi-platform mobile application. In: 2015 2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 140–141. IEEE (2015). doi:10.1109/MobileSoft.2015.34
10. Sikora, J., Sroka, J., Tyszkiewicz, J.: Strati5 - open mobile software for Harris matrix. In: Campana, S., Scopigno, R., Carpentiero, G., Cirillo, M. (eds.) Proceedings of the 43rd Annual Conference on Computer Applications and Quantitative Methods in Archaeology, vol. 2, pp. 1005–1014. Archaeopress Publishing Ltd., CAA (2016)
11. Sroka, J., Panasiuk, A., Stencel, K., Tyszkiewicz, J.: Translating relational queries into spreadsheets. IEEE Trans. Knowl. Data Eng. **27**(8), 2291–2303 (2015). doi:10.1109/TKDE.2015.2397440