Abstract: A graphical user interface (GUI) for interacting with a user, the GUI comprising: a grid region comprising a plurality of cells, the cells being arranged to receive content and to allow dependency relationships between cells to be created; a code region arranged to receive programming code; data entry means arranged to allow the user to enter content into the cells of the grid region or programming code to the code region; program code means arranged to analyze the plurality of cells within the grid region, generate programming code representative of the grid region and integrate the user-entered programming code contained within the code region and the programming code representative of the grid region into a resultant program; and output means arranged to output the resultant program wherein the program code means comprises code generation means arranged to: (i) traverse the plurality of cells in the grid region and identify content within the plurality of cells; (ii) determine dependency relationships between cells that contain content; and (iii) generate programming code representative of the dependency relationships and the content of the cells.
Published:
— without international search report and to be republished upon receipt of that report
Improvements relating to Graphical User Interfaces

Field of the Invention

The present invention relates to improvements to graphical user interfaces (GUI) and in particular a GUI for use in integrating programming functions with spreadsheet style applications.

Background to the Invention

The present invention relates to the use of electronic spreadsheets.

Spreadsheets are widely used for the analysis of numerical data, for such purposes as financial models and "what-if" projections. They are also frequently used for data collection, acting in this instance as a template to build a form (for the data collector) and as a program allowing the form to be filled in (for the data provider).

A spreadsheet normally consists of one or more grids of cells, each grid being called a worksheet; the spreadsheet as a whole can thus be referred to as a workbook. Each cell can contain a literal value or a formula; the formula is specified in a language similar to a programming language, and specifies how a value for that cell should be calculated, which is frequently through arithmetic manipulation of the values in other cells. For example, a series of twelve cells might contain values relating to a company's sales over the months of a year, and a final cell might contain a formula expressing the sum of those twelve cells. A cell may be referenced in such a formula in one of several ways, the most popular being to use a alphabetical column descriptor followed by a numeric row descriptor; for example, "A1" would refer to the top-left cell, and "AC23" would refer to the cell in the 29th column, 23rd row.

When a value or formula in a cell is changed within a spreadsheet, it recalculates all of the formulae that depend on that cell's calculated value. This rapid feedback and ease of use makes the spreadsheet a valuable tool for ad-hoc data analysis.

A typical worksheet that a user might create within a spreadsheet would have a regular table-like structure, with the cells in the first row containing column headers, the cells in the first column containing row headers, and the remaining cells containing data relevant
to both their row and their column. Rows and columns will frequently have summary cells at the end of the data, showing the total of all of the other cells.

Figure 1a shows an example of how a spreadsheet table may be specified and Figure 1b shows how the table may be displayed.

It should be noted that the formulae in cells can refer to the values calculated using other formulae - for example, the formula in cell G6 in the above example (see Figure 1a) sums up a set of sums from the cells above it. As such therefore spreadsheets use spatial relationships to define program relationships. Spatial relationships and dependencies between items are inherently easier for the average user to understand than sequential programming which usually requires typing code text line by line and it is this feature of a spreadsheet that means that many users find it easier to perform calculations in a spreadsheet than by writing the equivalent sequential program code.

Formulae referring to formulae which in turn refer to other formulae allow a spreadsheet to describe a complex calculation, making it able to perform some of the same functions as a traditional programming language - with considerably less training required for the user.

Spreadsheets have a number of intrinsic problems which stem from their development history and the difficulties inherent in changing the core user interface and coding language. The form of user interaction with a spreadsheet has not changed significantly in more than twenty years and the language used for their program code has not advanced with the developments in the computer field.

The various problems with existing spreadsheet implementations are discussed below.

Firstly, it is noted that traditional spreadsheets do not support the same level of functionality as sequential programming code products; existing commercial products frequently provide separate programming facilities that allow limited interaction with the spreadsheet model, but there is invariably a conceptual gap between the two parts of the system. In other words, existing products tend to be constructed as a programming language and a spreadsheet, with a limited connection between them.

Spreadsheets cannot easily be reused in larger-scale computer programs; frequently a spreadsheet to perform a calculation must be rewritten from scratch in a traditional programming language if it is to be reused.
Increasingly, spreadsheets are being used to hold large amounts of data, a task to which they are ill-suited; best practice would be to put the data into a database. There are two possibilities when using spreadsheets to handle large amounts of data. Either the data resides in an application other than the spreadsheet application and the spreadsheet requires read access to that data, or the data logically belongs within the spreadsheet application but needs to be edited by multiple users simultaneously.

It is also noted that combining data from multiple sources can be very difficult in current spreadsheet applications. For example: a sales director might want up-to-date information on the sales made by a number of salespeople. With a traditional spreadsheet, a template for each salesperson might be created and then emailed to each of them. Completed spreadsheets would then be received individually and would need to be aggregated into a summary document. Such a manual aggregation process is cumbersome and error-prone.

Frequently a worksheet is specified as the results of combining the numbers from other worksheets; for example, the numbers in a worksheet containing account balances might be multiplied by those in equivalent positions in a worksheet containing percentage interest rates, to provide an identically-structured worksheet containing interest amounts. In a traditional spreadsheet program, specifying a worksheet like this requires a formula to be near-replicated across the entire resulting sheet, and although spreadsheets provide functionality to make this replication easier, errors can be introduced into the resulting grids of formulae easily - for example, a new row added to the end of the first worksheet and to the second worksheet will not cause a new row to be added to the end of the third.

Some known spreadsheet systems have made attempts to address some of the above problems. For example, Calc4Web, a product of Savvysoft in New York, converts spreadsheets into C++ code which enables one-off conversions of complex spreadsheet models into computer programs but does not provide seamless integration between the two.

It is an object of the present invention therefore to address or substantially mitigate the above mentioned problems.

**Statements of Invention**

According to a first aspect of the present invention there is provided a graphical user interface (GUI) for interacting with a user, the GUI comprising: a grid region comprising a
plurality of cells, the cells being arranged to receive content and to allow dependency relationships between cells to be created; a code region arranged to receive programming code; data entry means arranged to allow the user to either enter content to the cells of the grid region or programming code to the code region; program code means arranged to analyze the plurality of cells within the grid region, generate programming code representative of the grid region and to integrate the user-entered programming code contained within the code region and the programming code representative of the grid region into a resultant program; and output means arranged to output the resultant program wherein the program code means comprises code generation means arranged to: (i) traverse the plurality of cells in the grid region and identify content within the plurality of cells; (ii) determine dependency relationships between cells that contain content; and (iii) generate programming code representative of the dependency relationships and the content of the cells.

The present invention provides a solution to the above mentioned problems by providing a graphical user interface in which a user can either enter content via a spreadsheet style interface, the "grid region", or programming code into a "code region".

The data entry means may be arranged to allow content within the grid region to be added and also to allow spatial relationships (dependency relationships) between cells to be defined or changed. Programming code entered via the data entry means within the code region is taken to mean sequential program commands.

The GUI is arranged to be able to analyse the content entered into the grid region and convert the spatial relationships (the dependency relationships) between cells within the grid region into sequential code. Additionally the program code means is arranged to output a resultant program of sequential code commands that comprises both the grid entered content and the user-entered programming code.

The GUI in accordance with the first aspect of the present invention therefore provides an interface which bridges the gap between the programming domain and the spreadsheet domain. This enables users of a computer system who have limited programming abilities to take advantage of the extra functionality that the programming domain provides without having to be able to encode the entire contents of a spreadsheet in a programming language.

The GUI also enables the resultant program to be output for use either within the context of the GUI or exported to an external application.
It is noted that the code generation means is essentially identifying cells which are independent (i.e. their content is not derived from or in any way dependent on other cells) and cells which are dependent (i.e. their content in some way is dependent on another cell). For example, for a spreadsheet having cells A1 to A5 which contain constants and a cell A6 which is the sum of cells A1:A5, the A6 cell can be viewed as dependent on cells A1:A5.

By identifying the dependency of cells within the grid region the code generation means can effectively place the cells in a sequential order, i.e. the order they would be executed by an equivalent expression in a programming language.

Cells in the grid region may be arranged such that spatial relationships are arranged between cells and preferably the code generation means is arranged to determine all spatial relationships defined by the grid region such that programming code comprising sequential programming commands can be generated that corresponds to the identified spatial relationships.

Conveniently, the output means may be arranged to export the resultant program, e.g. for use by an external application or for transmission to a remote user. This feature conveniently allows either an external application to utilise the program or a user to review and/or amend the resultant programming code.

Preferably, the program code means is arranged to execute the result program and the output means is arranged to output the results of the executed program for display in the code and grid regions of the GUI. This therefore allows results of updates to the grid or code regions to be calculated and then displayed back to the user via the GUI.

Conveniently the GUI may comprise an output region which may be used to display user print statements to trace execution. This is an extremely useful tool for a user who can enter code into the code region to output the intermediate results of the programming code for analysis. It is also noted that the output region may be used to display notification messages to a user (e.g. that an execution of the resultant program has commenced or that the recalculation process has completed).

Conveniently, a user may insert programming code into the code region that, when executed, imports data from an external data source. Such an external source may comprise a database and the programming code entered by the user may comprise code to generate a structured query language (SQL) query for the external data source.
Conveniently, data that is imported from an external data source may be inserted into cells within the grid region of the GUI.

Preferably, whenever data is imported into the GUI the program code means is arranged to recalculate the resultant program.

The binding of the code and grid regions in accordance with an embodiment of the present invention provides a mechanism for easily importing data into the grid region.

The GUI may be arranged to be in communication with a communications network and, in such instances, may preferably further comprise network listener means arranged to listen for notification messages from external data sources that are also connected to the communications network, the notification messages indicating that the content of the data source has been updated. By including network listener means into the GUI updates to the content contained therein may be automated.

The GUI may be arranged such that a visual distinction is made to a user between cells that contain content and cells that are empty. The visual appearance of the GUI may be controlled by code entered into the code region of the GUI.

Cells within the GUI may define an active region (where cells that either contain or display content are located) and an inactive region and the GUI may be arranged such that the boundary between the active and inactive regions is indicated by a change in cell colour.

The GUI may conveniently be arranged to define a rectangular region of the grid region such that all cells either containing or displaying content are located within the boundary of the rectangular region. The GUI may than be arranged to indicate the extent of the rectangular region by changing the cell colour across the region boundary. This is a very useful feature as, for grid regions that are larger than the display capacity of a display device, it can indicate that there are active cells outside of the user’s view.

Conveniently the GUI further comprises control means which is arranged to control the display of content on the GUI.

Conveniently, the control means comprises one or more data hooks, each of which indicates that cells within the grid region incorporate content from an external data source. Preferably, the GUI may be in communication with a communications network and may comprise network listener means as described above. For such a GUI (comprising data hooks and a network listener), whenever a notification message is
received at the network listener means, the control means may be arranged to determine whether the GUI requires updating by comparing the contents of the notification message to the one or more data hooks. In other words, a data hook may say "this GUI comprises data from database 1". Upon receipt of a message from database 1 indicating a change in content of the database, the control means may be arranged to determine whether the change in data affects the GUI.

It is noted that the control means may determine which cells of the grid region require updating or alternatively, if the GUI comprises a number of pages (cf multiple worksheets in a single spreadsheet), the control means may be arranged to determine which pages require updating.

The combination of the network listener means and data hooks allow a system to be created in which the limitations associated with shared spreadsheets can be overcome or substantially mitigated. A remote user may therefore update a shared spreadsheet by saving their data to a database table, the database being equipped with a module for sending a notification message whenever new content is added or existing content is changed. The content stored within the database table may then be imported automatically into the GUI according to the present invention thereby avoiding any manual aggregation as in prior art systems.

Conveniently, the GUI may comprise a plurality of pages, each page comprising its own grid region and the plurality of pages sharing the code region.

Conveniently, the program code means is arranged to traverse the cells of the grid region in order to identify cells that have relationships with other cells and to initially place the identified cells into an unprocessed list of cells. The program code means may then subsequently be arranged to determine all the dependency relationships of the cells in the unprocessed list and to reorder the cells into a processed list such that cells which are dependent on other cells for their content appear later in the list (later in time). Once the processed list has been created sequential programming code may be generated that corresponds to the now sequential list of cells in the processed list.

According to a second aspect of the present invention there is provided a method of interacting with a graphical user interface comprising: providing a grid region comprising a plurality of cells, the cells being arranged to receive content and to allow dependency relationships between cells to be created; providing a code region arranged to receive programming code; entering content to the cells of the grid region or entering
programming code to the code region; analyzing the plurality of cells within the grid region and generating programming code representative of the grid region; integrating the user-entered programming code contained within the code region and the programming code representative of the grid region into a resultant program; and outputting the resultant program wherein analyzing and generating step comprises: (i) traversing the plurality of cells in the grid region and identifying content within the plurality of cells; (ii) determining dependency relationships between cells that contain content; and (iii) generating programming code representative of the dependency relationships and the content of the cells.

According to a third aspect of the present invention there is provided a distributed computer system comprising: a computer terminal arranged to store a document, the document comprising at least one page comprising a GUI according to the first aspect of the present invention, the GUI comprising network listener means in communication with a communications network; a database management system comprising a database table and a network server module, the network server module being in communication with the communications network wherein the database management system is arranged to receive content from a user for storage in the database table and is arranged upon receipt of content from the user to instruct the network server module to send a notification message across the communications network indicating that the content of the database table has been updated and the GUI is arranged to receive the notification message via the network listener means and to determine whether any content associated with the GUI should be updated with the content in the database table.

The third aspect of the present invention relates to a distributed computer system in which a GUI in accordance with the first aspect of the invention is provided. By further providing network listener means and a database management system comprising a network server module that can output notification messages whenever the database is updated, the distributed system provides an arrangement in which users can effectively share a spreadsheet.

It is noted that the GUI may comprise multiple documents and any or all of the documents may be shared across the communications network.
It will be appreciated that preferred and/or optional features of the first aspect of the invention may be provided in the second and third aspects of the invention also, either alone or in appropriate combinations.

5 The present invention also extends to a computer program when embodied on a record medium/read-only memory/electrical carrier signal or stored in a computer memory, the computer program comprising program instructions for causing a computer to perform the process of the method of the second aspect of the present invention.

10 **Brief Description of the drawings**

In order that the invention may be more readily understood, reference will now be made, by way of example, to the accompanying drawings in which:

15 Figures 1a and 1b show a typical known spreadsheet arrangement;

   Figure 2 shows a screenshot of a graphical user interface (GUI) in accordance with an embodiment of the present invention;

   Figure 3 shows a schematic architecture of a computer terminal comprising an application in accordance with an embodiment of the present invention;

20 Figure 4a shows a general schematic of the various components that comprise an application in accordance with an embodiment of the present invention;

   Figure 4b shows the composition of two components of Figure 4a in greater detail;

   Figure 5 shows a further screenshot depicting the addition of a database to a GUI in accordance with an embodiment of the present invention;

25 Figure 6 shows an example of the generation of resultant program code in accordance with an embodiment of the present invention;

   Figure 7 depicts the overall process of generating program code from the contents of the grid region of the GUI;

   Figure 8 shows how the dependency relationships between cells in the grid region may be determined;
Figure 9 shows how program code may be generated once the dependency relationships have been determined;

Figure 10 shows a general system architecture of a computer comprising an application in accordance with an embodiment of the present invention which is in communication with a remote server computer comprising a database management system;

Figure 11 shows an alternative view to Figure 10 in which the internal structure of the application has been expanded;

Figure 12 shows an external application which is in communication with a database management system which in turn is in communication with an application in accordance with an embodiment of the present invention;

Figure 13 is a flow chart of the process of updating a document held in the GUI from an external application;

Figure 14 shows a known representation of a spreadsheet comprising a plurality of cells;

Figure 15 illustrates how cells within the grid region of the GUI in accordance with an embodiment of the present invention can be colour coded.

**Detailed description**

In the Figures and associated description below it is noted that like numerals are used to denote like features.

Figure 2 shows a screenshot of a graphical user interface (GUI) 2 in accordance with an embodiment of the present invention in which a user is exposed to the computer program underlying the spreadsheet alongside a traditional spreadsheet style grid.

As discussed above and as described in greater detail below the GUI in accordance with embodiments of the present invention converts the spreadsheet's structure and formulae into a readable, easily understood computer program which is executed whenever the spreadsheet is recalculated.

By tightly binding the spreadsheet grid and the code view together the present invention provides a number of advantages over known spreadsheet systems. For example,
functions defined via program code in the program code section of the GUI may be used in a spreadsheet grid cell or in other user code.

Returning back to Figure 2 it is noted that a user can input data and formulae through the traditional spreadsheet view, the grid region 4, or can enter code within a coding region 6 of the GUI by entering additional user defined functions and code.

The GUI according to an embodiment of the present invention is arranged to generate code from the formulae and data entered onto the grid and interleave this with code entered via the coding region of the graphical user interface. The composite code generated in this way can be updated in real time as the spreadsheet is modified and allows the user to modify and extend the program in a structured manner.

It is also noted that the graphical user interface of Figure 2 also comprises an output region 8 which allows the user to follow execution of their code for example by adding user print statements to trace execution, by displaying intermediate calculation results or by examining stack traces showing the details of any error's location.

Figure 3 is a basic schematic of a computer system 18 comprising a graphical user interface 2 in accordance with an embodiment of the present invention. It can be seen that the computer system comprises: a display device 20 upon which the graphical user interface 2 is displayed; input means 22 (in this example a keyboard is shown but other input means, e.g. mouse, may be used) for inputting code into the coding region 6 and formulae or data into the grid region 4 of the graphical user interface 2; a data store 24 for storing codes and formulae; and, an application 26 in accordance with an embodiment of the present invention arranged to integrate the coding and grid regions and to receive and process user interactions via the GUI and to control the display of data in the GUI. It is noted that the application and GUI are regarded as corresponding aspects of the present invention.

The application 26 in accordance with an embodiment of the present invention comprises code generation means 28 arranged to traverse the cells of the spreadsheet in the grid region (via traversal means 30) and to derive the dependency relationships between any occupied cells (using dependency means 32). Code fragments may then be generated, via the code generator 34, from these identified dependency relationships to create a "generated program" which may then be inserted into a resultant program that also
comprises code entered via the coding region of the graphical user interface. The resultant program is created by the Program Result Generator 36 which may also be arranged to execute the resultant program. The resultant program may be executed each time a user modifies the data within the GUI (either via the spreadsheet view 4 or the code view 6) and output for display within the graphical user interface or output to an external database or application.

As shown in Figure 3, the application 26 further comprises network listener means 38 that can receive notification message from external data sources that may signal that data in the application requires updating. A control means 40 is also provided which controls the display signal output via the output means to the control the display of the GUI on the display device 20.

It is also noted that the resultant program may be exported.

An example of the form of the application 26 and also a spreadsheet in accordance with an embodiment of the present invention is shown in Figures 4a and 4b.

Figure 4a is a general schematic of the various components that comprise the application 26 in accordance with an embodiment of the present invention. The application is arranged to create a "Central Object" 42 on the computer system on which it resides. This Central Object 42 acts as a core for the application and comprises a list of Main Form objects 44, each of which has an associated Document object 46. The Documents 46 each represent a spreadsheet, of the kind described above and the Main Form object 44 with which each Document 46 is associated comprises programming code (the control means 40 from Figure 3) to control the graphical user interface 2 with the Document. The Central Object also creates an object 38 (the network listener) whose purpose is to listen to external data sources (which are in communication with the application via the communications network 47) for data updates that may affect the data held in the Document 46 (the network listening means from Figure 3).

In terms of the screenshot shown in Figure 2, the spreadsheet as a whole would be represented by a Document object 46, and an associated Main Form 44 would handle the display of that Document on the computer's display 20, and the user's interaction with the Document.

Turning to Figure 4b, each Document 46 comprises:
A list of Worksheet objects 48, which is referred to as the Model list 50. This represents the user's specification of the spreadsheet, as defined by editing formulae, setting formatting details, and the like.

Three fragments of programming code in a programming language, referred to as the pre-constants user code 52; pre-formula user code 54 and post formula user code 56. For this embodiment, these represent the modifications the user has made to the resultant program; the pre-constants user code 52 relates to, for example, definitions and any other system code that the user has entered that he wishes to user later, the pre-formula user code 54 relates to, for example, code that utilises the constants that have been input above, and the post-formula code 56 relates to code that is executed after the generated program. It is noted that the post-formula user code would be a good place to put code that could create a new worksheet for summary information and to aggregate the various types of data together. It is noted that during the generation of the resultant program the user supplied pre-constants, pre-formula and post-formula code will be interleaved with code generated by the application.

A list of code fragments 60, each flagged as editable or non-editable. This represents the generated program with the user's modifications, i.e. the various pieces of user code, integrated. This resultant program code is displayed to the user in the coding region of the GUI.

A Result Generator object 36, which is described below.

A Workbook object 62. This represents the results of executing the generated program (as defined above) and the user's modifications to it, as expressed in the user code. The list of Worksheet objects held in the Workbook object is also duplicated in the Document for convenience, and is here known as the Result list 63.

A list of "Data Hooks" 64, each "Data Hook" 66 being an identifier to an external data source that, when changed, affects the data held in the spreadsheet/coding region. Whenever a change in the external data source is detected this triggers the application in accordance with an embodiment of the present invention to recalculate the resultant program code. This recalculation maintains synchronization between external data sources and any worksheets within the document that are displaying data derived from them; the details are described
below with reference to Figures 7 to 9. By way of example, an implementation of
the present invention in the context of worksheets relating to employee vacations
may comprise a data hook saying, in effect, "this Document requires recalculation
when there are changes to the Employees table on database A", and a further
data hook saying "this Document requires recalculation when there are changes
to the Employee vacation shared worksheet on database B".

Each Worksheet object 48, regardless of whether it is in the Result or a Model list,
contains a set of Cell objects 70, and (optionally) sets of Row 72 and Column 74 objects.
A Cell contains the information required to format a cell - for example the font and the
number of decimal places. Row objects can contain the height of the Row, Row-wide
formatting information, and similar properties. Column objects can contain the width of
the column, Column-wide formatting information, and similar properties.

The Cells contained inside Worksheets that are in the Document's Model list also have
the ability to store an expression, which may be a fixed value or a formula, which is
expressed in a language similar to a programming language.

A Worksheet object in the Document's Model list may also have a formula, but it cannot
have a fixed value. This allows a formula to be associated with a complete worksheet
and not just a cell.

The Cells contained inside Worksheets that are in the Document's Result list do not store
expressions, and simply store a value.

The Main Form object 44 is arranged to display the Graphical User Interface 2 described
above in relation to Figure 2 on the computer's display. Significant regions of the GUI
have already been discussed above but it is noted that the Main Form object is arranged
to provide the following components to a user via the GUI:

- The grid area 4 (also discussed above), which normally shows the values from
  the Cells in the Result list of Worksheets from the associated Document in a
  manner similar to that of a traditional spreadsheet program. When the user
  signals that he or she wishes to edit the Document (for example by double-
  clicking on a Cell), the view for that cell switches to showing in an editable text
  field the formula or value contained in the Cell from the Model worksheet list that
  corresponds to the Result-side Cell it was previously showing. If there is no such
  corresponding cell, it either creates one (if there is a Model-side worksheet to
  contain it) or does not let the user edit the Cell (if the current Worksheet has no
Model-side equivalent). If the user makes a change to a Cell, the edited value is written to the appropriate Cell in the Document's Model list, and then a recalculation, as defined below, is performed.

- The code region 6 (also discussed above), which displays the document's list of code fragments 60 as foldable "Chunks", each of which is editable or not depending on the code fragment.
- A number of command options typical of desktop computer programs, for example options to Open a new file, Save the current file, Cut, Copy, Paste etc.
- An option to export programming code. This option simply saves the contents of the code area to a file on a filesystem accessible from the computer.
- An option to add and remove Worksheets to/from the Model list. Doing this triggers a recalculation.
- An option to add Database Worksheets to the Model list, and an option to allow editing of the settings of these Database Worksheets (see Figure 5). These worksheets hold no cells, but instead contain the details required to make a standard Open Database Connectivity (ODBC) connection to the database in question and to query it using the database query language SQL. The user cannot change the cell values in these worksheets.
- An option to add Shared Worksheets to the Model list, and one to allow editing of the settings of these Shared Worksheets. These Worksheets hold no Cell objects directly, but instead contain the details required to make an ODBC connection to a specific database, and the name of a table within it that has a specific format. When the user applies a change to a Cell in a Shared Worksheet, the change is written to the associated database table, and a recalculation is not directly triggered. Adding a Shared Worksheet to a Document adds the associated database table to the list of Data Hooks associated with that Document; likewise removing the Shared Worksheet removes that Data Hook. The Data Hook's existence is what ultimately triggers the recalculation. For example, to use the employee vacation example from above, if one user edits the vacation detail shared worksheet, perhaps to add some vacation that they took, when they entered data it would be stored immediately into the database. The database would then send messages to all users' instances of the client program, which
would then all recalculate, at which point they would pick up the change. This is described in more detail below.

- An option to set a formula on a worksheet.
- An option to instruct the grid 4 to colour each cell white if it is within its worksheet's bounds, and grey if it is not, allowing the user to see what part of a worksheet the data takes up (see also Figures 14 and 15).

As described above, the application 26 is capable of analyzing the data entered via the spreadsheet region 4 of the GUI 2 and any code entered via the code area 6 in order to generate program code, the resultant program. The generation of code fragments from the grid view is handled by the code generation means 28. The composition and execution of the resultant program is handled by the Result Generator 36.

There are six main sections of program code which are executed by GUI/application in the following order for every recalculation of a workbook. Three are system defined and are not editable and the other three (also thought of as "set-up", "tweak" & "polish") are user input and maintained.

1. **Import statements & Worksheet creation** - The code here loads appropriate background data, libraries and databases (where created through the spreadsheet user interface) and creates a workbook and worksheets for results to go into. This code is not editable by the user.

2. **Pre-constants user code** - The first section in the code where the user can enter their own code. It allows users to "set-up" their workbook as entering programming code in this section enables data from external sources to be uploaded and unique, personalised functions to be incorporated. External libraries can be loaded here from central repositories or internet sites. Bespoke links can also be established to external databases instead of using the drop-down menus to create more standard database connections. Code executed here cannot reference data or formulae entered into the grid as they have not yet been defined in the program code.

3. **Constants and formatting** - This section of code is generated by the system from cells containing just data constants (such as text or numbers but not formulae) and from all the formatting defined at cell, row or column level in each worksheet of the workbook. As data is held completely separately from the formulae, this aids locking
the formulae and not the data making it easy to produce a secure data input file. This code is not editable by the user.

4. **Pre-formulae user code** - Entering programming code in this section enables users to "tweak" their calculations as it allows access to constants, but values to be defined by formulae have not yet been calculated. User code in this section can reference the constants entered into the worksheet grid, but values to be defined by formulae in the grid have not yet been calculated in each recalculation and therefore cannot be used. However functions or calculations defined in the pre-constants user code can be used.

5. **Formula code** - This section of code is generated by the code generation means from the formulae entered into the worksheet grid. The formulae in the grid are converted where necessary to a suitable programming syntax but there is always a one-to-one relationship between formulae typed into grid cells and formulae in this section of the program code. This code is not editable by the user.

6. **Post-formula user code** - Entering programming code in this section enables users to "polish" the result by highlighting, isolating and/or manipulating outputs for re-use elsewhere. User code in this section can use any data or results on the grid or use any functions already defined. However it cannot be used by any formulae on the grid and therefore is best used for final changes to the presentation of data on the grid or for exporting data to external files or databases etc.

Figure 6 illustrates a very simple example of the generation of the resultant program code by the Results Generator 36 (also referred to as the Program Result Generator above or the "program code means" in the claims) in accordance with an embodiment of the present invention and also how data and formulae input via the grid region 4 of the GUI 2 are used in the different sections of the program code and how results are displayed back on the GUI.

In this example a title has been defined in cell A1, numbers have been typed into two cells (B2, B3) in the spreadsheet region 4 of the GUI 2 and a formula has been typed into cell B4. Box 80 indicates the data that has been input into the grid region 4 of the GUI 2.

Box 82 is a section of the actual GUI showing a two column, four row table 84 with the above data and formula in place.
Box 86 indicates the process steps in generating the program code, i.e. the code that is generated from the traversal of the spreadsheet and from the user's modifications to the code entered as entered via the code region of the GUI.

In Step 88 (corresponding to point 1 above "Import statements and Worksheet creation"), system defined code is created. No user code is added at this time.

In Step 90, the program code means takes any pre-formula code entered by the user. It is noted that this step effectively allows users to set up the system to access external data sources. It is also noted that any user defined functions or constants that may have been entered into the code region will be available for use by formulae that are entered into the grid region of the GUI.

In Step 92, text and numbers entered as constants in the grid region of the GUI are automatically converted into program code.

In Step 94, the program code means takes any pre-formula code that may have been entered by the user. It is noted that user defined functions or constants in the pre-formulae user code section are available for use on the grid (as denoted by line 96).

In Step 98, formulae entered in the grid region 4 of the GUI 2 are automatically converted by the code generation means 28 into program code. In this step the application 26 traverses the grid region 4 determining cell dependencies and generating code fragments 60 as described in more detail with respect to Figures 7 to 9 below.

In Step 100, the program code means 36 takes any post-formula user code that has been entered, e.g. user code to output the results to a destination other than the GUI. The application is also arranged to display the results of Steps 88, 90, 92, 94, 98 and 100 on the GUI 2.

As noted above, code and functions added as user code can be used by formulae entered in the grid 4. Similarly formulae or constants in the grid 4 can be referenced by user code. Additional libraries and functions can be added through the code region 6 of the GUI 2 which can be used both within the code region itself and from the cell formulae in the grid region. This tightly binds the cell formulae and the code region together into a
single coherent program. Defining a function in the user code allows that function to be used in a grid cell or in other user code.

An example function to add on Value Added Tax (17.5% in the UK) is shown below. This simple function - `withVAT()` - takes an amount passed to it (which could be a cell reference) and returns the amount after adding VAT.

```python
def withVAT(amount):
    return amount * 1.175
```

This function can either be used in a cell:

```
IfCellA1 = 100 then typing "=withVAT(A1)" into CellA2 would show 117.5
```

Or directly in user code:

```
vatTotal = withVAT(workbook["Sheet1"][A2].Value)
```

The application 26 (i.e. the Result Generator 36) executes the generated program (with the user's modifications and extensions) each time the user changes it, either via the spreadsheet view or directly in the code view, and then takes the results of this execution and displays them back as values in the spreadsheet or through interactions with external databases and services.

The application also allows the user to export the customised program - that is, both what was generated from their work in the grid, along with any code they have written - as code that can be executed as part of a traditional computer program. There is always a one-to-one relationship between the code displayed and the code executed to populate the spreadsheet.

Figures 7, 8 and 9 show the process of generating program code from the contents of the spreadsheet, i.e. the processes involved in Step 98 of Figure 6 above. Figure 7 details the overall process and Figures 8 and 9 relate to respectively determining the dependency relationships between cells in the grid region and generating code for the cells once the dependency relationships have been determined.
Turning to Step 102 of Figure 7, the code generation means first creates a list of all cells or worksheets in the grid region 4 that are associated with formulae. These items are hereinafter referred to as "dependency items" and the list is referred to as the "unprocessed list".

The code generation means 28 additionally creates an initially empty list that will hold dependency items in the order that they will appear in the final derived program code. This second list is referred to as the "processed list".

Finally, a third list is created to contain the "recursion chain" which is used to detect cycles in the graph. This list is also initially empty. It is noted that in computer programming terminology a "graph" is a set of nodes with edges connecting the nodes. In the context of the present invention, a graph is a set of cells with dependencies connecting them.

In Step 104, the code generation means visits each dependency item in order to determine the turn in which these items should appear in the generated code and to place the items into the "processed list". This process is discussed in more detail below with reference to Figure 8.

At the end of Step 104, the "processed list" has been populated and the application has a list of dependency items in the order that they should appear in the generated code with each cell/worksheet from the grid region that has a formula appearing after any cells/worksheets on which that formula depends. It is also noted that the application has derived a list of Cycle objects, i.e. dependencies that cannot be resolved. For example, consider the case where cell A1 has been set equal to cell A2 (A1=A2) but cell A2 has also been set equal to cell A1 (A2=A1). The corresponds to a cycle A1→ A2→ A1 which the code generation means will be unable to resolve.

In Step 106, for each cycle identified in the dependency identification step 104, the code generation means is arranged to produce code that sets the values stored on all of the Cells in the cycle to zero and to associate an error object with each of them and with each Worksheet in the cycle. The code generated in Step 106 is then executed and added to a list of code fragments that form part of the generated code. It is noted that the cycles identified in Step 104 above correspond to user added content that contains
errors. Step 106 is therefore a mechanism for flagging up errors to the user for further attention and correction.

In the above example of cycle A1→A2→A1, if cell A3 refers to cell A2 (i.e. cell A3 is a cell which is dependent upon a cell in a cycle) then Step 106 provides a value for cell A2 which is then available for use in cell A3 during Step 108.

In Step 108, the code generation means is arranged to generate an expression in the chosen programming language that is equivalent to the dependency item's formula and to use this to generate code that, when executed will evaluate the expression and place the results in the dependency item's value field. This process is described below in Figure 9 in more detail. The code generated in Step 108 is also added to the list of code fragments that comprise the generated code.

Figure 8 shows the steps involved in identifying the dependency relationships in Step 104 above. In Step 104 the code generation means is arranged to visit each dependency item identified in Step 102 and to call a function, hereinafter referred to as the "visit function", on each item to identify its dependency relationships.

In Step 110, the visit function is arranged to visit a dependency item contained in the unprocessed list.

In Step 112, the visit function checks if the dependency item is already in the recursion chain created in Step 102. If the item is already in the chain then, in Step 114, it adds the chain to a list of cycles and then ends (Step 116).

If however the item is not already in the chain then, in Step 118, the visit function checks if the current dependency item is in the processed list. In other words it checks whether it has already been handled during the process of Step 104 (for example the current dependency item may already have been encountered when processing an earlier dependency item in the unprocessed list). If the current dependency item has already been handled then the process ends at Step 116.

If, however, the current dependency item has not been handled before then the visit function moves to Step 120 in which any dependency items on which the current dependency item depends are identified.
In Step 122, the visit function checks whether there are any unprocessed dependency items in the list of dependency items identified in Step 120. If there are none then the visit function ends at Step 116. If however there are unprocessed items then a new temporary chain is created by appending the current dependency item to the existing chain and then recursively calling (Step 124) the visit function on each of the unprocessed items, passing in the temporary chain.

In Step 126, the visit function determines whether it has encountered a cycle. If it determines it has encountered a cycle (yes) then it moves to Step 116. If it determines it has not encountered a cycle (no), then the visit function returns to Step 122.

The skilled person will appreciate that the above steps relate to a topological sort of a dependency graph where the nodes are cells and the edges are dependencies.

Figure 9 illustrates the process of generating code for each dependency item in the processed list determined in Figure 8 above. At Step 130, the code generation means takes the next dependency item in the processed list and generates an expression in the programming language that is equivalent to the dependency item's formula. This expression is then used to generate code that, when executed, will evaluate the expression and place the results in a value field associated with the dependency item.

In Step 132, the code generation means executes the code generated in Step 130.

In Step 134, the code generation means checks whether the executed code generated an error. If no error was generated, the code generated in Step 130 is added, in Step 136, to a list of code fragments that will be used to construct the resultant program code.

If an error is detected in Step 134, then some safe “fallback” code is generated in Step 138 to place the value zero in a value field associated with the dependency item and to associate the error with the cell or worksheet so that it can be displayed to the user later. This fallback code is then added, in Step 140, to the list of code fragments.

Figures 10 and 11 are general schematics of a computer network incorporating an application and system in accordance with embodiments of the present invention. In Figure 10, a computer terminal 18 similar to that described above with reference to...
Figure 3 is shown. The computer terminal comprises a display device 20, input devices 22 and an application 26 in accordance with an embodiment of the present invention. An operating system 150 (e.g. MS Windows) provides the platform upon which the application runs. The computer system 18 also comprises a network connection 152 to a computer network 47 (e.g. LAN or wireless network or the Internet).

Figure 10 also shows a server computer 154 which is in communication with the computer terminal 18 via the computer network 47. The server computer comprises a database management system 156 (DBMS, e.g. Microsoft SQL Server) running on an operating system 158. The DBMS is in communication with a data store 160 and a server module 162 that talks to both the DBMS 156 and the operating system 158.

As noted above the provision of a code region 6 in the GUI 2 enables a user to insert program code that is arranged to import data from external data sources into the application. By this mechanism a user may construct a document via the GUI 2 that is updated with data from, for example, an external database. The document constructed via the GUI is enabled to receive data from such external sources via the use of a data hook or database listener.

Figure 11 shows an alternative view of the schematic of Figure 10 in which the internal structure of the application 26 has been expanded to show the various components of the system. It is seen that the central object 42 is now associated with a network listener means 38 (which corresponds to the network listener means described above) which is arranged to listen for notification messages indicating that an external data source (156, 160) has been updated.

Figures 12 and 13 detail how a document composed in accordance with an embodiment of the present invention is updated upon a change to an external data source.

Figure 12 shows an external application 170 (such as a trade entry system that updates a bank's database. Basically the external application could be any program that can communicate with a DBMS) which is in communication with a database management system 156 residing on a server computer 154. The database management system comprises a number of database tables 172 of which one is shown in Figure 12. The server computer also comprises a server module 162 which is arranged to connect the server computer to a computer network 47.
An application 26 in accordance with an embodiment of the present invention is also shown and comprises a network listener module 38 and a plurality of Documents 174.

Whenever the external application 170 sends data to the DBMS 156, the relevant documents are updated in accordance with the procedure shown in the flow chart of Figure 13.

In Step 180, the external application 170 requests the DBMS 156 to store data in a database table 172.

In Step 182, the DBMS writes the data received in Step 180 to the database table 172 and, in Step 184, notifies the server module 162 which table has been updated. In Step 186 the server module sends a message across the computer network 47 specifying which table 172 has changed and which server and which database the table is stored on.

In Step 188, the network listener 38 in each application 26 on each computer terminal 18 that is connected to the computer network 47 receives the update message that was sent in Step 186.

In Step 190, the database listener 38 in each connected computer terminal iterates through all Documents 174 within the application and tells them to recalculate if they are interested in the specified server/database/table combination. Each Document 174 determines whether it is interested in the update by comparing the identification details of the updated database with their Data Hooks 66.

It is noted that the ability to notify Documents 174 (composed by the application 26 according to an embodiment of the present invention) that data in an external source has been updated enables data to be gathered from multiple sources and integrated into a single spreadsheet. In this instance multiple recipients of a spreadsheet could update data held in a database table of a database management system which in turn would trigger a notification message via the server program. A remotely located Document with the appropriate data hooks would then be capable of updating any data saved in the database table.
Figures 14 and 15 depict identical spreadsheets comprising a plurality of cells and illustrate the manner in which cells within the grid region can be colour coded to alert a user of the extent of data held within the spreadsheet.

In Figures 14 and 15 cells containing content have been filled in and it can be seen that there is a main block of cells 200 and a single cell 202 that is isolated from the main block of content-containing cells. Rectangle 204 denotes the portion of the spreadsheet that is visible on a user's computer terminal and it is noted that cell 202 falls outside of the visible area of the display device.

In Figure 14 (a known representation of a spreadsheet) there is the possibility that the user may be unaware of the existence of cell 202. This is especially so if the content of this cell has been added by another user or if the spreadsheet is of far greater extent than the version shown in the Figure.

Figure 15 shows the grid region 4 of the GUI 2 in accordance with an embodiment of the present invention. In this case the control means 40 is arranged to colour code the cells such that cells comprising content or displaying content have a white background. Additionally cells 206 and 202, representing the topmost left and bottommost right cells that contain/display content define the corner points of a rectangular region 208 in which the cell background colour is also set to white.

Cells located outside of the rectangular region 208 are colour coded such that their background colour is not white.

In embodiments of the present invention therefore, a user is visually alerted to the size and extent of the spreadsheet within the grid region 4 of the GUI 2 and the risks of overlooking cell content are greatly reduced.

It will be understood that the embodiments described above are given by way of example only and are not intended to limit the invention, the scope of which is defined in the appended claims. It will also be understood that the embodiments described may be used individually or in combination.
CLAIMS

1. A graphical user interface (GUI) for interacting with a user, the GUI comprising:
   a grid region comprising a plurality of cells, the cells being arranged to receive
   content and to allow dependency relationships between cells to be created;
   a code region arranged to receive programming code;
   data entry means arranged to allow the user to either enter content to the cells of
   the grid region or programming code to the code region;
   program code means arranged to analyze the plurality of cells within the grid
   region, generate programming code representative of the grid region and to integrate the
   user-entered programming code contained within the code region and the programming
   code representative of the grid region into a resultant program; and
   output means arranged to output the resultant program
   wherein the program code means comprises code generation means arranged to:
   (i) traverse the plurality of cells in the grid region and identify content within the
   plurality of cells;
   (ii) determine dependency relationships between cells that contain content; and
   (iii) generate programming code representative of the dependency relationships
   and the content of the cells.

2. A GUI as claimed in Claim 1, wherein a first set of cells within the grid region is
   arranged to define a spatial relationship to a second set of cells within the grid region and
   the code generation means is arranged to determine the spatial relationship between
   cells such that programming code comprising sequential programming commands can
   be generated that equates to the determined spatial relationships.

3. A GUI as claimed in Claim 1 or Claim 2, wherein the output means is arranged to
   export the resultant program.

4. A GUI as claimed in any preceding claim, wherein the program code means is
   arranged to execute the resultant program and the output means is arranged to output
   the results of the executed resultant program for display in the code and grid regions of
   the GUI.

5. A GUI as claimed in any preceding claim, further comprising an output region
   arranged to display user-coded print statements to trace execution.
6. A GUI as claimed in any preceding claim, wherein the code region comprises programming code that when executed is arranged to import data from an external data source.

7. A GUI as claimed in Claim 6, wherein the external data source is a database.

8. A GUI as claimed in Claim 6 or Claim 7, wherein the programming code is arranged to generate an SQL query for the external data source.

9. A GUI as claimed in any of Claims 6 to 8, wherein the imported data is inserted into cells within the grid region.

10. A GUI as claimed in any of Claims 6 to 9, wherein the program code means is arranged to recalculate the resultant program whenever imported data is received.

11. A GUI as claimed in any preceding claim, wherein the GUI is in communication with a communications network and further comprises network listener means arranged to listen for notification messages from external data sources connected to the communications network, each notification message indicating that the content of an external data source has been updated.

12. A GUI as claimed in any preceding claim, wherein a sub-set of the plurality of cells in the grid region comprise content and the GUI is arranged to visually distinguish between cells containing content and empty cells.

13. A GUI as claimed any preceding claim, wherein the grid region is arranged to comprise an active region and an inactive region such that cells that either display content or comprise content are located within the active region.

14. A GUI as claimed in Claim 13, wherein the boundary between the active and inactive regions is indicated by a change in cell colour.

15. A GUI as claimed in any of Claims 12 to 14, wherein the GUI is arranged to define a rectangular region of the grid region such that all cells comprising content or displaying content are located within the boundary of the rectangular region, the cells within the rectangular region being of a different colour to cells outside the rectangular region.

16. A GUI as claimed in any preceding claim, further comprising control means arranged to control the display of content on the GUI.
17. A GUI as claimed in Claim 16, wherein the control means comprises one or more
data hooks, each data hook indicating one or more cells within the grid region that
incorporate content from an external data source.

18. A GUI as claimed in Claim 16 or 17, wherein the GUI is in communication with a
communications network and further comprises network listener means arranged to
listen for notification messages from external data sources connected to the
communications network, each notification message indicating that the content of an
external data source has been updated.

19. A GUI as claimed in Claim 18, wherein upon receipt of a notification message from
an external data source at the network listener means the control means is arranged to
determine which cells in the grid region require updating by comparing the contents of
the notification message with the one or more data hooks.

20. A GUI as claimed in Claim 18, wherein the GUI is arranged to display a document
comprising a plurality of pages and wherein upon receipt of a notification message from
an external data source at the network listener means the control means is arranged to
determine which documents require updating by comparing the contents of the
notification message with the one or more data hooks.

21. A GUI as claimed in any preceding claim, wherein the GUI is arranged to display a
document comprising a plurality of pages, each page comprising a grid region and the
plurality of pages sharing the code region as defined in Claim 1.

22. A GUI as claimed in any preceding claim, wherein programming code entered by a
user in the code region is referenced in the grid region.

23. A GUI as claimed in any preceding claim, wherein the program code means is
arranged to traverse the plurality of cells in the grid region, identify cells that have
dependency relationships with other cells and add these cells to an unprocessed list.

24. A GUI as claimed in Claim 23, wherein the program code means is arranged to
determine the dependency relationships between cells in the unprocessed list and to
reorder the cells into a processed list such that cells which are dependent on other cells
for their content are arranged later in time.

25. A method of interacting with a graphical user interface comprising:
providing a grid region comprising a plurality of cells, the cells being arranged to
receive content and to allow dependency relationships between cells to be created;
providing a code region arranged to receive programming code;
entering content to the cells of the grid region or entering programming code to
the code region;
analyzing the plurality of cells within the grid region and generating programming
code representative of the grid region
integrating the user-entered programming code contained within the code region
and the programming code representative of the grid region into a resultant program; and
outputting the resultant program
wherein analyzing and generating step comprises:
(i) traversing the plurality of cells in the grid region and identifying content within
the plurality of cells;
(ii) determining dependency relationships between cells that contain content; and
(iii) generating programming code representative of the dependency relationships
and the content of the cells.

26. A distributed computer system comprising:
   a computer terminal arranged to store a document, the document comprising at
least one page comprising a GUI as claimed in any one of Claims 1 to 24, the GUI
comprising network listener means in communication with a communications network;
   a database management system comprising a database table and a network
server module, the network server module being in communication with the
communications network
   wherein the database management system is arranged to receive content from a
user for storage in the database table and is arranged upon receipt of content from the
user to instruct the network server module to send a notification message across the
communications network indicating that the content of the database table has been
updated and
   the GUI is arranged to receive the notification message via the network listener
means and to determine whether any content associated with the GUI should be updated
with the content in the database table.

27. A computer program comprising program instructions for causing a computer to
perform the process of Claim 25, embodied on a record medium.
28. A computer program comprising program instructions for causing a computer to perform the process of Claim 25, stored in a computer memory.

29. A computer program comprising program instructions for causing a computer to perform the process of Claim 25, embodied in a read-only memory.

30. A computer program comprising program instructions for causing a computer to perform the process of Claim 25, embodied on an electrical carrier signal.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
<td>Apr</td>
<td>May</td>
</tr>
<tr>
<td>2</td>
<td>Sales of paper</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>=SUM(B2:F2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Sales of card</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>=SUM(B3:F3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Sales of paperclips</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>=SUM(B4:F4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Sales of pens</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>=SUM(B5:F5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Total sales</td>
<td>=SUM(B2:B5)</td>
<td>=SUM(C2:C5)</td>
<td>=SUM(D2:D5)</td>
<td>=SUM(E2:E5)</td>
<td>=SUM(F2:F5)</td>
</tr>
</tbody>
</table>

**Figure 1a**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
<td>Apr</td>
<td>May</td>
</tr>
<tr>
<td>2</td>
<td>Sales of paper</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>Sales of card</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>Sales of paperclips</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>Sales of pens</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>6</td>
<td>Total sales</td>
<td>400</td>
<td>800</td>
<td>1200</td>
<td>800</td>
<td>1200</td>
</tr>
</tbody>
</table>

**Figure 1b**
<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sales of paper</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>Sales of card</td>
<td>120</td>
<td>250</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>Sales of newspaper</td>
<td>130</td>
<td>230</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>Sales of pass</td>
<td>130</td>
<td>230</td>
<td>300</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>Total sale</td>
<td>460</td>
<td>830</td>
<td>1200</td>
<td>810</td>
<td>1200</td>
</tr>
</tbody>
</table>

Figure 2
Figure 5
Create a list of cells and worksheets that have formulae (items) (the unprocessed items), an empty list of cells/worksheets that have been processed, and an empty "chain" list.

For each item, call the "visit" function defined below.

For each cycle, produce code that sets the values on the Cells in the cycle to zero and that associates an error with every Cell or worksheet in the cycle. Execute, and add to the list of fragments.

For each processed dependency item, generate and execute code - as defined below - and add the code to the list of fragments.

End

VISIT FUNCTION

Return, reporting cycle

Is cell/worksheet in chain list?

Are there unprocessed dependencies?

Did the recursive call trigger a cycle exception?

Generate list of dependencies

End

Recursively call the visit function for the next unprocessed cell/worksheet in the dependency list, passing in a chain that contains the current cell/worksheet.

Are cell/worksheet in processed list?

Are there unprocessed dependencies?
Start → Generate code for current worksheet/cell's expression → Execute code in provided context

Generate safe code

Execute safe code in provided context, add to list of code fragments → End

Did the execution trigger an exception?

Y: Add code to list of code fragments

N: Continue with the next step
Figure 11

APPLICATION

DOCUMENT

MAIN FORM

CENTRAL OBJECT

NETWORK LISTENER

NETWORK

SERVER COMPUTER

KEY:

Contains one of

Contains several of
APPLICATION REQUESTS DBMS TO STORE DATA

DBMS WRITES DATA

SERVER MODULE INFORMED

SERVER MODULE SENDS MESSAGE

NETWORK LISTENER RECEIVES MESSAGE

NETWORK LISTENER ITERATES THROUGH DOCUMENTS

Figure 13