COMP 304B – Object-Oriented Software Design
Assignment 2 – Spreadsheet UML Design

Due date: Thursday March 13, 2003 before 23:55

Practical information

- Team size == 2 (pair design) !
- Each team submits only *one full solution*. Use the `index.html` template provided on the assignments page. Use **exactly** this format to specify names and IDs of the team members. The other team member *must* submit a single `index.html` file containing only the coordinates of both team members. This will allow us to put in grades for both team members in WebCT. Beware: after the submission deadline there is no way of adding the other team member's `index.html` file and thus no way of entering a grade !
- Your submission must be in the form of a simple HTML file (`index.html`) with explicit references to *all submitted files* as well as inline inclusion of images. See the general assignments page for an `index.html` template.
- The submission medium is WebCT.

Goal

In this assignment you will use UML *Class Diagrams* and *Collaboration/Sequence Diagrams* to describe the design of a Spreadsheet application. At this point, the spreadsheet application does not yet have a graphical user interface component.

Your assignment solution should contain:

1. A UML Class Diagram for your design. As the diagram may grow too large, you may split it into different parts. This splitting may break a link (*e.g.,* association) in the diagram. In this case, the class on the "remote" side of the link must be *repeated* (possibly without attributes).
2. A UML Object-Interaction Diagram describing a small but meaningful use case.
3. A brief textual discussion of your design.

Upload *all* files to WebCT and provide links to them from your `index.html` file.

Design

In Figure 1, the design from the first assignment is given in the form of a UML Class Diagram. A larger version can be found here.

Class Diagram

Your design will start from a main application `Spreadsheet`. For the time being, there is no graphical user interface, and spreadsheet only contains the `name` of a spreadsheet and the `data`. `data` is an instance of `SpreadsheetData`. For the time being, a `Spreadsheet` contains one instance of `SpreadsheetData`. Note how in the future, the design might be extended to allow for multiple `sheets`, each an instance of `SpreadsheetData`.

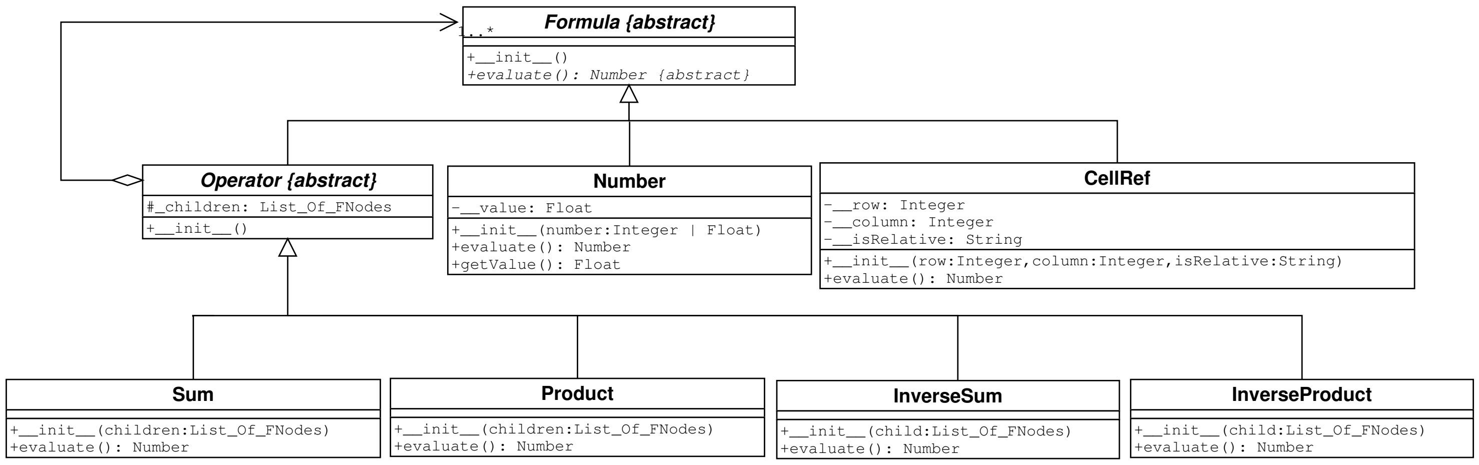

Figure 1: The spreadsheet formula design

`SpreadsheetData` holds the actual content of the spreadsheet. This content is distributed over a number of `SpreadsheetCell` cells. As can be seen in last year's first assignment, it makes sense to let `SpreadsheetData` hold a dictionary of `SpreadsheetCell` references indexed by `(row, column)` keys as opposed to a dense (array) representation. This, as (1) a spreadsheet can be quite sparse and (2) we wish to have row and column values unbounded. For printing purposes, `SpreadsheetData` should keep track of `min_row, min_col, max_row, max_col`. The `get_cell` and `set_cell` methods allow access to `SpreadsheetCell` objects at some `(row, column)` in the `SpreadsheetData`. The signature of these methods must reflect this. `SpreadsheetData` supports `evaluate` and `check_dependencies` methods. `evaluate` will evaluate all cells (calculate their value from the formula). `check_dependencies` will sort the cells in an order appropriate for efficient calculation. It will also check for cyclic dependencies. It is used internally (only) by `evaluate`.

`SpreadsheetCell` objects know their own `row` and `column`. They have a `value` which is never set, but rather derived by `evaluate`-ing the `formula`. `SpreadsheetCell` contains exactly one `Formula` referred to as `formula`. Appropriate `set` and `get` methods are present to access the `formula` reference (which is initially `None`). Explicit setting of `formula` (as opposed to obtaining it through parsing a formula string) is required when formulas are copied from one cell to another. Usually, `formula` will be obtained by means of the method `parse_formula` from the `formula_string`. Note how subsequently asking for the string representation (aka as unparsing) of `formula` should yield a string equivalent to the original `formula_string` (a sanity requirement). The `get_depend` method will collect dependencies in the `depends_on` list.

Note how it with the design of assignment 1 it was impossible to implement the `evaluate` method for `CellRef` as there was no way to access other cells' formulas. This must be fixed now.

Note also how *all* classes in the whole design (including the part copied from assignment 1) must implement the `__str__` method to allow for textual output.

Object-Interaction Diagram

Draw an Object-Interaction Diagram (Collaboration Diagram *or* Sequence Diagram) for a typical use case where formula strings are entered in a few cells, parsed, evaluated, ... Doing this will give you insight into what exactly will have to happen in each of the methods. Describe the sequence of operations in text in addition to in the diagram. Not too much detail is required. You may wish to use pseudo-code annotations.

References

- Dia, the gtk+ based diagram creation program was used to draw the above UML class diagram. It is available from http://www.lysator.liu.se/ alla/dia/
  Dia is available in the SOCS labs on the FreeBSD machines. You may use any other tool however.
- `design.dia` is the Dia source of the above design. You may use it as a starting point.