On the Division of Responsibility for Refreshing a Cell

Technical Memorandum #90-3
Ward Cunningham
April 23, 1990

This report traces the computation of a single value to appear in a single cell of a report. Were the report implemented as a spreadsheet, the formula associated with the cell would perform the computation, possibly aided by additional macros and worksheets. In fact, the calculation is spread over nearly a dozen Smalltalk objects that cooperate to perform the calculation in an organized and efficient way. Each object's contribution is explained. Further, we provide justification for this fragmentation by citing for each object related objects that can be substituted into the framework to produce related computations.

The computation in question, yield to maturity of a GIC contract, has proven to be one of the most involved in our development and testing. Programmed in Excel the calculation takes tens of seconds to complete. Smalltalk duplicates the calculation in a fraction of a second while employing more objects and protocols than any other instrument. What do those objects do? Their responsibilities fall into these four general categories.

Class	**Responsibility**
CellPane ReportBrowser	User interface
PositionReport AveragedColumn FormattedRows	Report specification
DatedPosition GI Ccalculator	Calculation optimization
CompoundBalance YieldDaemon CashFlow	Calculation implementation

This table was generated by stopping the computation and tracing through the suspended message sends. The same traceback appears below. Each entry includes the specific message received and a discussion of how it is handled. Our trace begins with the display of the report window. Five columns have already appeared. The sixth column, Yield, has reached a GIC on line two.

`CellPane refreshCell: 6 @ 2`

A CellPane is a kind of pane that fills its part of a window with tabular data. We pick up the calculation with it refreshing column 6, row 2.

`ReportBrowser cellAt: 6 @ 2`

The pane queries its model, a browser, for the string contents of the cell. This browser can adjust the columns of a user-defined report. Calculation, however, is delegated to the report.

`PositionReport cellAt: 6 @ 2`

Reports retrieve rows (in this case positions) and view them through the columns that they store.

`AveragedColumn formattedAt: 2`

A column can retrieve, cache and format a quantity from a row. This variant also reports a (possibly weighted) average for totals and subtotals. But, first the column gives the rows a chance to intervene in the retrieval.

`FormattedRows column: aColumn at: 2`

A FormattedRows is a kind of Collection that can permute row indices when required by row sorting or subtotaling. Retrieval continues with the permuted index (unchanged in this example).

`AveragedColumn valueAt: 2`

The column, having failed to find a cached value for row 2, sends its stored selector (#yield) to the row.

`DatedPosition yield`

Now calculation begins in earnest. The row, a DatedPosition, creates a calculator which it will cache to speed subsequent queries.

`GicCalculator yield`

Bond calculators can answer most queries by plugging instrument related quantities (dates, rates, etc.) into formulas. The GIC calculator, however, must actually run out all future cash flows from stored schedules and future-dated transactions. This comes down to finding the daily balance from here to maturity.

CompoundBalance yieldOn: $5,666,199.37

Each GIC interest calculation method has its own class in the heirarchy of balance calculators. This one makes payments based on the "compound" calculation method. A Balance can answer simple questions regarding interest or principal directly. More complicated calculations are delegated to daemons that run in sync with the balance.

YieldDaemon yield

The YieldDaemon, in particular, captures data from balance events (deposits, payments, etc.) and stuffs it (along with the initial balance) into a generalized CashFlow.

CashFlow rateOfReturn

A CashFlow is optimized for computing its own present value, the inner loop of the rate of return calculation. This CashFlow and the YieldDaemon that created it are discarded once the result is safely cached in the GICCalculator.

The calculation unwinds. The value passed back will be manipulated as it goes, first converted to a string of characters and then finally to a pattern of bits on the display screen. Again, each object is prepared to do its part.

Once an object has completed a task it stands ready to perform another. Often objects will cache the results they produce so that they can be delivered immediately when needed again. Objects caching obsolete values may be replaced by fresh recruits, ready to repeat the calculation. This works only as long as all data within an object age at the same rate. This has an obvious impact on the division of responsibilities as can be observed in the following table. Lifetimes range from indefinitely long to as short as the calculation. Others match the lifetime of a window or only last between window updates.

Object	Responsibility	Lifetime
CellPane	Fill screen space	Window
ReportBrowser	Examine Reports	Window
PositionReport	Retrieve & describe Positions	Indefinite
AveragedColumn	Format, aggregate & cache values	Indefinite
FormattedRows	Permute rows	Update
DatedPosition	Represent a pos-ition on a date	Update

GICCalculator	Compute statistics	Update
CompoundBalance	Compute payments	Update
YieldDaemon	Select & organize	Calculation data
CashFlow	Compute PV & ROR	Calculation

Lifetime variations motivate the separation of large objects into smaller ones. Other sources of variation have contributed to the present decomposition of the yield calculation. Whenever variation on a theme is present, Smalltalk programmers like to capture that variation into a hierarchy of related objects with common behavior factored to the top. We illustrate this influence by listing objects related to, but not involved in, the yield calculation we traced.

Object	**Related Objects**	**Variation**
CellPane	TextPane FieldPane	Display
ReportBrowser	TransactionBrowser	Interaction GraphBrowser
PositionReport	ActivityReport PerformanceReport	Query
AveragedColumn	TotaledColumn ProportionedColumn	Aggregation
FormattedRows	Array OrderedCollection	Indexing
DatedPosition	DatedCashposition DeferredCashPosition	Position
GICCalculator	CouponCalculator DiscountCalculator	Formulas
CompoundBalance	SimpleBalance EffectiveAnnualBalance	Interest
YieldDaemon	ForcastDaemon MaturityDaemon	Purpose
CashFlow	WeightedCashFlow	Formulas

Is this the best, or even the last, decomposition of this problem? Not likely. As the application grows and evolves these objects will take on additional responsibilities which may or may not fit well in this framework. Designs vary in their ability to accommodate change. A design with slack can advance in many directions, one with none cannot change without regression.

The amount of slack in our program has varied throughout development. The introduction of new kinds of objects (i.e. frameworks) takes time but creates slack. Coding against a framework (by using excessive navigation or case analysis, for example) consumes slack. We have often allowed slack to run dangerously low, waiting for a specific project to motivate reorganization. In the following table we group objects by their respective framework, identify the motivating project behind the framework, and list them in chronological order.

Object	**Motivating Project**	**Date**
CellPane ReportBrowser	Standard framework	n.a.
PositionReport AveragedColumn DatedPosition	User-defined reports	6/89
CashFlow	Performance report	8/89
GICCalculator	Standard formulas	9/89
FormattedRows	Subtotals	1/90
YieldDaemon CompoundBalance	GICs	4/90

Would different project priorities have produced a different program? My guess is no. The creation of slack for expansion is only the third of three forces we have seen applied to the decomposition. The other two, lifetime and inheritance, overwhelm the third. This holds true, of course, only so long as project priorities allow for the creation of slack through regular reorganization.

We have traced the computation of the value of a report cell containing a GIC's yield. Of the ten objects involved, one half had interface responsibilities, the other half, computational responsibilities. We analyzed the decomposition from three perspectives, lifetime, inheritance and chronology. We speculate that chronology has little influence on the final product so long as sufficient "investment in slack" is maintained throughout the schedule.

Scanned and reformatted March 1, 1999
Ward Cunningham