References

[1] R. Ross. *Design of Personal Computer Software*, pages 282–300. IEEE Press, New York, 1985.

Box A: Spreadsheets

Spreadsheet programs have gained enormous popularity as flexible, extensible tools for financial accounting [1]. A spreadsheet can be viewed as a shared data pool represented by a matrix of values, with toolies represented by equations associated with positions in that matrix. When data in one of the matrix entries changes, the run-time system automatically reevaluates all equations that depend on that entry, updating the appropriate entries. Suppose an equation defines the rightmost value in a row as the sum of the row's other values. If a user changes one of those values the spreadsheet will automatically reevaluate the sum. This in turn may trigger the reevaluation of other equations, such as one to add all of the values in the rightmost column. A spreadsheet's toolie invocation mechanism thus depends on dataflow analysis to determine which matrix entries affect which equations. (Early spreadsheets simply recalculated the entire matrix.) Most spreadsheets have simple mechanisms that do not handle, or even identify, circular relationships among the toolies. Luckily, in the domains most generally addressed by spreadsheets, such circularities rarely arise.

A spreadsheet system is not a general-purpose tool environment, but a specialized application generator. Consequently, the range of toolies that a spreadsheet implementor can describe is constrained. Nonetheless, spreadsheets exhibit the architectural hallmark of tool abstraction: a shared pool of data together with event-driven control of function. Moreover, they effectively handle functional evolution: circularities aside, equations can be added to the system independently of other equations in the system; while complex dependencies may exist between these equations, the *system* rather than the programmer manages those interactions.