FRM: An Intelligent Assistant for Financial Resource Management[1]

Andrew Gelman, Susan Altman, Matt Pallakoff, Ketan Doshi,
Catherine Manago, Thomas C. Rindfleisch and Bruce G. Buchanan
Knowledge Systems Laboratory
Stanford University

Abstract

FRM is an experimental, knowledge-based system that assists in the judgmental aspects of budget planning and financial resource management. Problem solving in this domain requires many kinds of knowledge from many sources. We represent domain knowledge uniformly as constraints and view resource management and planning problems as constraint satisfaction and resolution tasks. We sketch here the financial resources management problem, our approach, and early results, concentrating on constraint representation and management issues in the system.

1 Introduction

Preparing and managing budgets are knowledge-based activities that require substantial expertise to do well. These are constraint satisfaction tasks, in the abstract, where the constraints are symbolic as well as numeric, and are judgmental as well as definitional. They are large tasks in which the organization of knowledge is critical to their success.

The FRM system[2] is a prototype working program that attempts to integrate many of the tasks an intelligent financial assistant should perform beyond the bookkeeping that a spreadsheet program does with numerical relations. It is an object-oriented system in which hierarchical organization among constraints, as well as among budget items and budgets themselves, is an important design principle. We use the same mechanisms to represent a hierarchy of perspectives under which to view the same financial information in different ways. Because of the nature of budgeting tasks, it is important also to represent temporal segments of budgets implicitly as sub-budgets and reason with them just as arbitrary collections of line items can be considered as sub-budgets. A uniform interface is provided by a form-filling system that is itself driven by constraints on how to present information under a perspective.

While constraints, perspectives, and hierarchies are the central themes of our work to date, we also include in FRM, and briefly report on, a replanning system that adjusts finished budgets in light of new information and an explanation system that presents audit trails or explanations under specified perspectives. FRM also includes a distributed database utility in its design, but not in its current implementation. Figure 1 shows the major components of the FRM system that are described in subsequent sections.

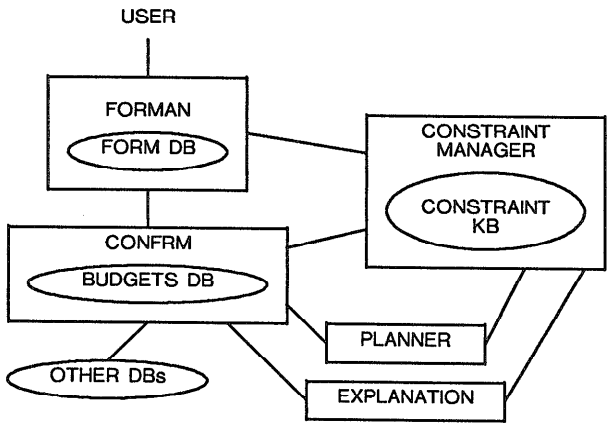

Figure 1: The major components of the FRM system.

2 Methods

2.1 User Interface: FORMAN

Form filling is a natural metaphor for the budgeting assistant, and a job that most managers will gladly turn over to an assistant. FORMAN is the FRM interface through which users create,

[1]This work was funded in part by: DARPA under contract N00039-86-C-0033; Boeing Computer Services under contract W266875; a gift from Price Waterhouse Technology Centre; Lockheed Missiles and Space Company under gift 1-72-L031; NASA under cooperative agreement NCC2-274; and NIH under grant RR-00785.

[2]FRM runs on Xerox 1186 machines and is implemented in the CLASS/HYPERCLASS object-oriented programming system [SmithR 86, Schoen 83]. CLASS and HYPERCLASS are trademarks of Schlumberger Technology Corporation and were formerly known as STROBE and IMPULSE respectively.

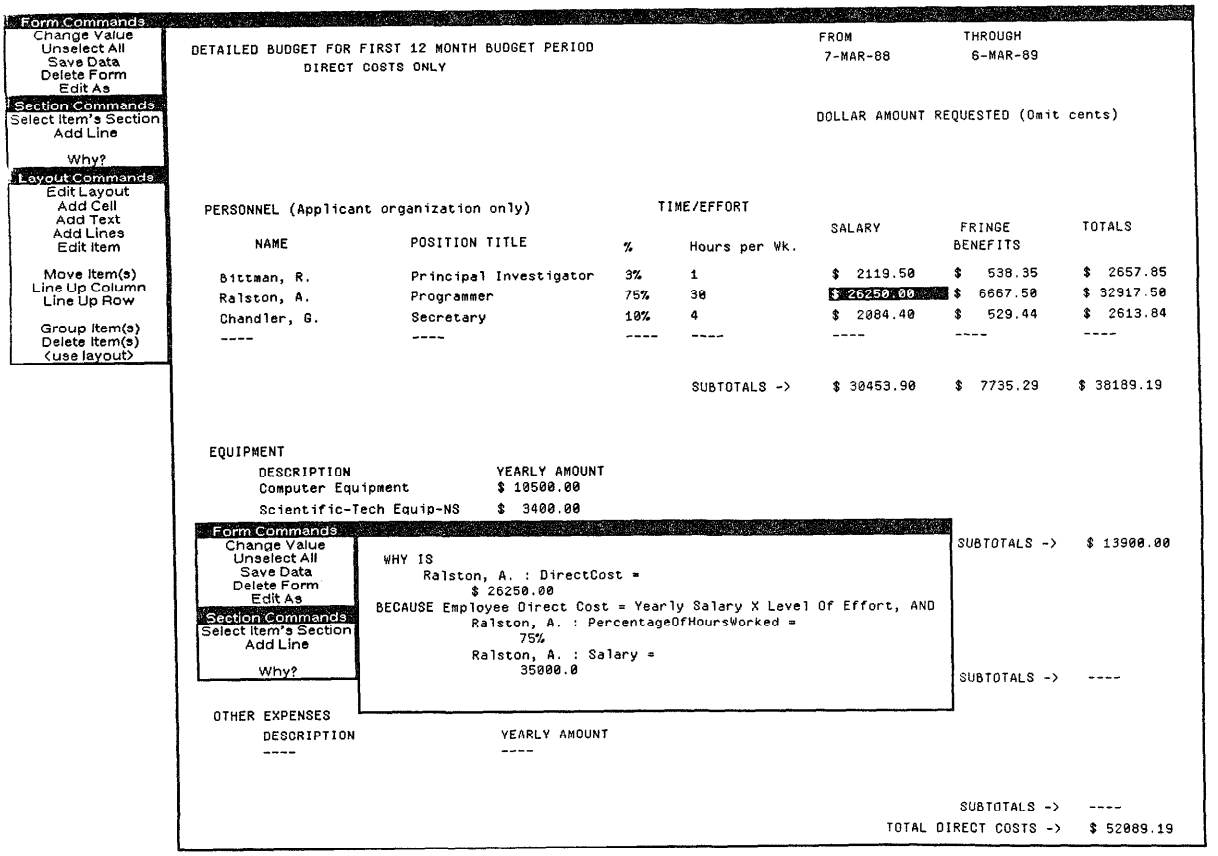

Figure 2: User's view of a form during an FRM session.

examine, and modify budgets. Users select items, with a mouse, on images of forms and invoke operations on the items by selecting commands from menus (see Figure 2). When a value on the form is changed, the system may change other values automatically or after consultation with the user as a result of applying domain knowledge. We have attempted to keep interactions simple and consistent by adopting menu-driven, object-oriented, and what-you-see-is-what-you-get (WYSIWYG) approaches to user interfaces.

A key design feature is the separation of data, stored in the CONFRM data managing module, from presentation information which is the domain of FORMAN.[3] One datum may appear on several different forms concurrently. Conversely, a single form may be used repeatedly to view different budgets.

A form is defined as a collection of text, active cells, and sub-forms, all represented internally as objects. Sub-forms are forms themselves and may be displayed and edited accordingly. As an example, the form in Figure 2 has a sub-form labeled "PERSONNEL".

FORMAN has three main components: a form editor, a form data base, and a table that links items in the form data base to locations in the budget data base. The form editor is built on the HYPERCLASS object editor and is responsible for creating and maintaining the graphic images of forms.[4] Form structures are stored and classified hierarchically in the form database and can be specialized, copied and edited to create new form layouts. These layouts become views of budget data when form's cells and sub-forms are linked to locations in the CONFRM database. A table object maintains these links. Each table entry points both to a CONFRM location and to all FORMAN objects that display the location's value. The table provides a means for FORMAN to instruct the database to change a value and for the database to tell FORMAN when a value needs to be redisplayed.

[3]Ciccarelli's work[Ciccarelli 84] also emphasizes separating presentation information from data.

[4]HYPERCLASS editors are hierarchies of CLASS objects that describe components of an editor (e.g., a window, command menus, main and sub-editors), along with message receivers and associated functions that perform the essential editing tasks.

Early use of the system indicates that with flexibly defined forms and intuitive user interactions, FORMAN provides users of FRM with a powerful tool for creating views and using them to manipulate data. Further developments would increase FORMAN's utility. These include a database browser for linking forms to data, and improvements to the human interface of the copy and linking mechanisms.

2.2 Constraint Representation and Management

Spreadsheets operate with numerical constraints on the values of cells in a matrix. FRM extends the concept of constraints to include not only relations among numerical values, but also relations among names, titles, and other symbolic values. FRM encodes in constraints its knowledge of how to fill out or revise a form, and how to make substantive changes to budgets [Gelman 87]. The system recognizes that some constraints are strong and must be satisfied without exception, while others reflect weak preferences, with many judgmental considerations in between.

The language of constraints must be expressive enough to capture the following kinds of knowledge:

- *Definitions* -- the total cost of a budget is the sum of the costs of its sections;
- *Rules & Policies* -- a Principal Investigator must devote at least x% of his/her time to a project;
- *Promises & Commitments* -- if you support my student this quarter, I will support yours next quarter;
- *Judgments & Preferences* -- agency A is unlikely to support more than x% time for clerical support;
- *Planning Heuristics* -- try to support student researchers full time during the summer, giving preference to PhD candidates over MS candidates;
- *Rebudgeting Strategies* -- when reducing a budget's total cost, cut non-essential items before essential items.

The constraint whose syntax is illustrated in Figure 3 is a symbolic, preferential one that cannot be represented by a spreadsheet formula. When more than two part-time secretaries provide support in a budget, it may be desirable to create a view that combines the clerical components into a single "super-secretary" item. This constraint will detect such a situation and modify the structure of the current budget view, while retaining a detailed underlying representation for use when the extra detail is appropriate. Super-items are described in Section 2.3.

Still other kinds of constraints check on relationships between parts of a budget. For example, experience may show that telephone or supplies should be budgeted at a constant dollar amount times the number of full-time-equivalent employees. Such a constraint has a conditional corrective action. If no telephone expenses are yet budgeted, it creates a telephone budget item with the indicated cost. If telephone costs are present but have a value inconsistent with the constraint, it updates the cost accordingly.

```
CONSTRAINT: SuperSecretary

Arguments = ($Budget $Secretary $AllSecretaries)
IF-Clause = (Type? $Secretary SECRETARY)
THEN-Clause = (Less (Length $AllSecretaries) 3)
CorrectiveActions = (CreateSuperItem $Budget $AllSecretaries)
BindClause-1 = (BIND $Secretary (confrm PersonnelItems))
BindClause-2 = (BIND $Budget (FindRoot $Secretary))
BindClause-3 = (BIND $AllSecretaries
                    (FindItems $Budget SECRETARY))
Strength = 4
Priority = 300
ImposedBy = Agency A
Source = Bittman
Author = Ralston
LastEdited = 1/01/88
```

Figure 3: Syntax for a typical symbolic constraint.

All of the FRM constraints have a common structure. A constraint is an object, created or edited through a specialized editor. The editor guides the input of slot values to ensure they are valid, and checks for consistency with pre-existing constraints [Altman 88]. A constraint may have any number of *arguments*, which will be bound to values at execution time. An individual *clause* is an expression consisting of arguments, constants, and the constraint language operators. The *IF-clause* corresponds to the preconditions of the constraint and is a logical expression made up of zero or more clauses. The *THEN-clause* is a conjunction of clauses that describe a desired state. *Corrective actions* are statements specifying database modifications to be invoked upon detection of a violation.

Each argument has a *binding clause* that binds it to either a database location, the value stored at such a location, or to the result of a functional expression The language allows bindings to be expressed in terms of other arguments in the same constraint. Arguments are bound dynamically during constraint evaluation as their values are needed. All bindings are generated from the initial binding of the *enable argument* of the constraint. The enable argument is the one corresponding to the datum whose changing value triggered the constraint; it may be a different argument each time the constraint is activated.

Links between budget data and the constraints are created at the time a constraint is loaded. These links depend on binding the arguments to class objects in the database, and are used to enable the constraint when slot values are changed. Enabled constraints are added to a task agenda from whence they will be evaluated by the Con-

straint Manager/Scheduler. The scheduler decides which of the pending tasks has highest priority and executes it. The priority attribute of a constraint gives a default measure of the urgency of considering the constraint.

The evaluation process begins with the IF clauses of the constraint. The IF-Evaluator checks each of these clauses to see if the preconditions are met. If they are, the THEN-Evaluator is called to check for a violation of the desired relationship. If it is satisfied, no action is taken. Otherwise, corrective actions may be undertaken to force satisfaction. Possible actions include filling in or overwriting database values, creating or deleting budget items, calling the planner (see Section 2.5), or consulting the user about an unusual situation.

Our constraint language supports the specification of and reasoning about time intervals [Allen 84, Ladkin-A 86, Ladkin-B 86]. Temporal representation in constraints supports viewing time slices of budgets which are equivalent to sub-budgets along the temporal dimension. Constraints use appropriate rate computations that differentiate, for example, between yearly and monthly rates, and language operators implicitly handle variables whose values change over time. We provide a set of operators describing primitive temporal relations as well as higher level operators to manipulate intervals. Our extrapolation constraints provide a way to project a budget from one time interval to another using the time operators and methods that convert relative time intervals to absolute ones.

Constraint hierarchies allow users more control over the invocation of families of constraints. Constraints are indexed by several attributes, such as expert source or strength. The user can load and delete groups of constraints using these or user-defined indices and thus have the system use one expert's preferences or any other desired combination of constraints. Similarly, evaluation of some constraints may be deferred during hypothetical sessions or in early stages of budget preparation.

There may be times when a manager decides to violate constraints, or is forced to compromise because of conflicts between constraints. FRM currently provides a simple means to manage these situations. Each constraint has a *strength* attribute, which indicates the importance of satisfying its relationship. It provides a quantified measure of the *hardness* or *softness* of the constraint. We believe negotiation expertise [Lax 86] is relevant when considering conflicting constraints that have different criteria for importance, and are looking at ways of incorporating this knowledge into the FRM planner.

2.3 Perspectives and Recursive Sub-budgets

The design of CONFRM was guided by the need for a flexible and extensible representation that allows for multiple hierarchies. A budget is often part of a larger budget in an organizational framework, and conversely may itself represent the merger of smaller sub-budgets. The FRM system must be able to display budget information at an appropriate level of abstraction. Also, a budget may be organized quite differently for presentation to different agencies (e.g. NIH as opposed to NSF). In order to satisfy these needs we have implemented the concepts of *recursive sub-budgets* and *perspectives*.

Several object hierarchies exist in the CONFRM subsystem, the most central being the taxonomic Canonical Representation Hierarchy (CRH). CRH class objects contain definitions of all budget object attributes, including slots for costs, descriptions, codes, etc. Object types become increasingly specialized as one moves downward through the CRH, e.g. the object PersonnelItems has slots for *EmployeeName* and *Salary*, while *EquipmentItems* has a *UnitCost* slot. There are two main subtrees in the CRH, one a hierarchy of budget *items*, indivisible budget expense entities, the other of *sections*, which represent mergers of sub-budgets. Another CONFRM hierarchy contains *ItemTypes*, a collection of several hundred objects, each describing a recognized type of budget expense, e.g. "Telephone Costs" or "Books and Publications." Budget items may be made instances of these objects, through which they may inherit various slot values and constraints.

The ability to maintain multiple presentations of a single set of data is achieved through the use of perspectives. A perspective is a collection of objects and constraints that define a particular view of the full set of budget items. Each perspective has a designated root object. The sibling perspective objects form a tree of arbitrary depth below the root, successively refining the budget organization into sub-budgets. The objects at the leaves of the perspective tree are sets (or sub-budgets) of actual budget items from the canonical hierarchy (see Figure 4).

A *perspective constraint* may be associated with any leaf perspective object, e.g., *Domestic Travel* in Figure 4. Such a constraint describes the conditions whereby a budget item could be a sub-budget of the perspective, and would be loaded automatically when the perspective is activated. Suppose, for example, the user preparing a budget under the NIH perspective adds an item to the "Supplies" section and enters "Furniture" as its description. The perspective constraints linked to the description field will be evaluated and the one governing membership in the Equipment section will fire. The constraint's corrective action removes the item from the Supplies section and adds it to Equipment.

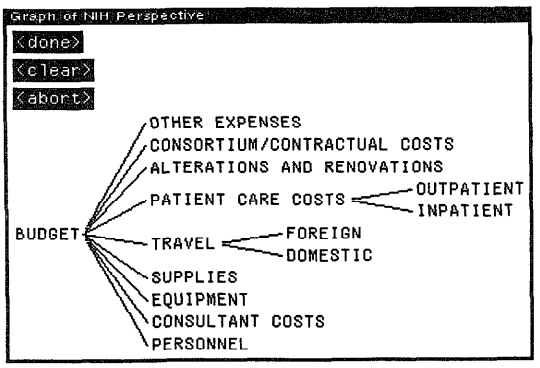

Figure 4: The structure of the NIH perspective. Each node in the graph represents one perspective object.

The sub-budgeting model extends from perspectives to other sets of budget items. New super-budgets can be created by combining two or more budget item sets into a super-set. Each set involved in such a merger maintains its identity and may be viewed individually as before. Sets to be combined may represent different tasks or sub-projects within a project or may represent different time-slices of a single budget. The combining process is recursive in that super-sets may themselves be merged into larger sets.

Returning to the "super-secretary" example discussed in Section 2.2, a leaf node of a perspective hierarchy may be a *super-item* which is the composition of two or more related items, but which we wish the system to treat for most purposes as a single item. The final product is a hybrid of a perspective object and a budget item. A super-item is a leaf node in the perspective tree to which it belongs, but is subject to constraints on perspectives as well as those on budget items.

Controlling how super-items are constrained may provide the key to manipulating budgets at a high level of abstraction. If a manager is working on an abstracted budget for an entire organization, the items s/he sees will generally be super-items. Normally changes to costs in super-items pose complex planning problems in trying to propagate corresponding changes down to the component items. But suppose the system is instructed to treat, for the interim, these super-items as items. They would thus be subject to item constraints rather than perspective constraints, and could be manipulated without resorting to planning processes. The necessary downward propagation of these changes could be deferred until such time as the manager wishes to concentrate on lower budgetary levels. A similar mechanism operating on the root of the perspective could defer upward propagation. We are at present developing this functionality and believe it to be a feasible solution to potentially massive scoping and combinatorial explosion problems inherent in the budgeting process [Duda 87].

2.4 Explanation

An explanation facility has been implemented for FRM that describes, on request, how a location acquired its current value, and if possible justifies the value. If the current value was set by the corrective action of a constraint, the explanation contains an automatically generated description of the constraint's clauses and the arguments used in calculating the value. Explanations are recursive in that the values of these supporting arguments may in turn be questioned. In the case of a user-entered value, the explanation tells when and by whom the value was entered.

2.5 Planning

The FRM planner is called by the constraint manager to determine a sequence of actions to fix a constraint violation. The current simple planner [Chan 87] proceeds hierarchically. The generation of the next sequence of actions is guided by the solution produced at a higher level and by planning heuristics. Some heuristics determine the set of corrective actions which can be chosen at each planning step, while others prune and order the search space (explicit control knowledge is defined in the form of meta-rules). Another type is used to gauge the relative importance of budget expenses. The hierarchical approach and the application of heuristics produce a first solution which minimizes constraint violations. However, there are always many possible ways to achieve a planning goal, e.g., to cut $1,000 from a budget. We are currently working to better exploit the hierarchical representation of budgets and extend the meta-rules to allow the generation of alternative solutions.

3 Results and Conclusions

A prototype version of the FRM system integrates all of the components we have described -- FORMAN, CONFRM, the Constraint Manager, PLANNER, and the explanation module. This experimental system demonstrates the advantages of the approach reported, even though it runs with one-half second to 15 second delays on the Xerox 1186 and has not been put into full operational use. It duplicates and significantly exceeds the functionality of an earlier FORTRAN program that we used for budgeting and that had a knowledge of the rules for the Stanford environment built in procedurally. Aside from the obvious improvements of a graphics-based interface, FRM provides a declarative specification of the basic budgeting and presentation rules so these can be changed at will.

The most common budget preparation tools in use today are spreadsheet packages. While these commercial systems are more polished than our prototype system, FRM has a number of powerful capabilities not provided by spreadsheets, including:

- FRM can encode judgmental knowledge and provide suggestions. Constraints do not have to be rigid relationships.
- FRM can handle symbolic as well as numeric constraints, as exemplified by the "super-secretary" constraint.
- Constraints can produce structural changes to the budget by causing new items to be created or deleted as appropriate.
- Constraints can be expressed generically and need not be specifically connected to individual cells. The delayed binding mechanisms in FRM allow constraints to be linked and invoked automatically whenever the triggering situation is detected in the budget form.
- The recursive sub-budget capability allows a flexible partitioning or aggregation of budget elements without specifically having to program the relationships and combination actions cell by cell.
- Different user preferences and institutional requirements for budget formats and information presentation can be accommodated through the mechanism of perspectives.
- The FRM planner can take into account tolerances on budget values in order to jointly satisfy constraints. Constraints can be overridden for specific cases and the planner can "reverse-engineer" line item changes.
- FRM has a simple explanation facility which allows the user to examine the chain of calculations or actions producing an observed value. This facility is not a model-based explanation at present as in [Kosy 84], but suffices for relatively tightly constrained budgeting situations.

In parallel with experimenting with the FRM system, we reproduced some of its functionality in a MicroSoft EXCEL spreadsheet template using the macro facilities available. The EXCEL spreadsheet was extremely brittle in that it was not possible to protect users from overwriting formulas and still give them the ability to manipulate other items. The spreadsheet implementation tightly embeds the inter-element relationships with the data presentation, resulting in a rigid and opaque system. Trying to build in needed flexibility proved very frustrating because of the limited nature of the programming language provided to relate cells or manipulate them in macros. We believe that the FRM constraint-based model provides a much more powerful and flexible environment in which to express budgetary relationships and to support user interactions.

Acknowledgements

We thank Jean-Luc Bonnetain, Jean-Luc Brouillet, Dennis Chan, Craig Cornelius, Don Henager, and Carla Wong for their contributions to the FRM project. And we thank Reid Smith, Eric Schoen, and the Schlumberger Palo Alto Research center for their contribution and support of the CLASS/HYPERCLASS object-oriented system on which FRM is built.

References

[Allen 84] Allen, J.F. Towards a General Theory of Action and Time. *Artificial Intelligence* 23(2):123-154, July, 1984.

[Altman 88] Altman, S. *Knowledge Aquisition and Representation in FRM.* Internal Working Paper KSL 88-45, Stanford University, Knowledge Systems Laboratory, June, 1988.

[Chan 87] Chan, D. *PLANNER: An Intelligent Budget Planner.* Internal Working Paper KSL 87-74, Stanford University, Knowledge Systems Laboratory, June 1987.

[Ciccarelli 84] Ciccarelli, E. *Presentation Based User Interfaces.* Technical Report AI-TR-794, MIT Artificial Intelligence Laboratory, August, 1984.

[Duda 87] Duda, R.O., Hart, P.E., Reboh, R., Reiter, J. and Risch, T. Syntel: Using a Functional Language for Financial Risk Assessment. *IEEE Expert* 2(3):18-31, Fall, 1987.

[Gelman 87] Gelman, A. *CONFRM: Managing Financial Resources with Constraints.* Internal Working Paper KSL 87-14, Stanford University, Knowledge Systems Laboratory, February, 1987.

[Kosy 84] Kosy, D.W. and Wise, B.P. Self-Explanatory Financial Planning Models. In *Proceedings of the National Conference on Artificial Intelligence*, pages 176-181. American Association for Artificial Intelligence (AAAI), August, 1984.

[Ladkin-A 86] Ladkin, P. Time Representation: A Taxonomy of Interval Relations. In *Proceedings of AAAI-86*, pages 360-366. AAAI, 1986.

[Ladkin-B 86] Ladkin, P. Primitives and Units for Time Specification. In *Proceedings of AAAI-86*, pages 354-359. AAAI, 1986.

[Lax 86] Lax, D. and Sebenius, J. *The Manager as Negotiator.* The Free Press, 1986.

[Schoen 83] Schoen, E. and Smith, R.G. IMPULSE, A Display-Oriented Editor for STROBE. In *Proceedings of the National Conference on Artificial Intelligence*, pages 356-358. AAAI, August, 1983.

[SmithR 86] Smith, R.G. and Carando, P. *Structured Object Programming in STROBE.* Technical Report SYS-86-17, Schlumberger-Doll Research, October, 1986.