MANAGEMENT SCIENCE
Vol. 37, No. 12, December 1991
Printed in U.S.A.

FW/SM: A PROTOTYPE STRUCTURED MODELING ENVIRONMENT*

ARTHUR M. GEOFFRION
Anderson Graduate School of Management, University of California, Los Angeles, Los Angeles, California 90024-1481

A research prototype implementation has been evolving for several years in parallel with the vision, theory, language, and applicative aspects of structured modeling. The objective of this article is to describe the capabilities of this implementation as it now stands, and to comment on how it contributes toward fulfilling a previously published agenda for the development of a new generation of modeling environments. The intended audience is all modeling system designers and evaluators, including those who do not happen to take a structured modeling approach.
(STRUCTURED MODELING; MODEL MANAGEMENT SYSTEMS; DECISION SUPPORT SYSTEMS)

1. Introduction

Structured modeling (SM) is a way of thinking about analytic models and the systems which support them. It is based on the idea that an analytic model can be viewed as a diagram of definitional dependencies among objects—each representing a distinct part of the model—that are typed, grouped by similarity, and organized hierarchically. The diagram includes a connection from one object to another object if the second object's definition depends directly on the first object's definition. Moreover, the diagram is intended to be computationally active in the sense that certain types of objects have associated mathematical expressions for computing their values.

Developing a model in the form of such a diagram is largely an exercise in objectivity. In contrast, using a model for problem-solving, design, gaining insight, or other purposes involves largely subjective intentions and agenda. These we choose to view as the "problems" or "tasks" associated with a model. Typical problems and tasks have to do with drawing inferences, determining an acceptable solution, ad hoc query, and optimization. Usually one or more computerized model manipulation tools need to be applied in order to effectively solve the problem or execute the task at hand. For certain standardized problems and tasks, these tools are highly developed and known as "solvers."

Thus, structured modeling views models and model-based systems as having computationally active definitional dependency diagrams as their central focus, with manipulation by solvers and other tools to achieve desired purposes. This viewpoint is compatible with the requirements set forth in Geoffrion (1989c) for a new generation of modeling environments.

There are many potential advantages of this viewpoint, including the abilities to: deal with a wide variety of models and modeling paradigms within a single unified and rigorous formalism, provide easily communicable diagrams for any model to supplement its usual diagram(s) (if any), keep track of important interdependencies among model parts that many current modeling formalisms leave implicit, insulate solvers from model changes and, conversely, exploit hierarchical organization as an approach to managing complexity, describe a model in a way that is both straightforward to understand and computer executable, and easily exploit relational database tools for data management.

* Accepted by Thomas M. Liebling; received June 1990. This paper has been with the author 6 months for 2 revisions.

0025-1909/91/3712/1513$01.25

The theoretical foundation of SM has been formalized in Geoffrion (1989b), and a model description language called SML (Structured Modeling Language) appears in Geoffrion (1990a) and (1990b) (see also 1991a). A literature of more than 100 published and unpublished works on SM has grown up during the last five years or so, but we make no attempt here to survey it.

This article builds directly on the SM and SML papers just cited. Its purpose is to describe and comment on the capabilities of FW/SM, the SML-based research prototype implementation of SM developed at UCLA between 1986 and 1990.

Although SM provides the motivation and context of our discussion, this article is designed to be of use not just to the SM community, but to modeling system designers and evaluators of all persuasions. There are quite a few features of FW/SM that designers and evaluators may find of particular interest. Some derive mainly from the underlying SM framework or the design particulars of SML, such as:

- unusually comprehensive error-checking—in fact, exhaustive in a strong sense—for the formal specification of general model structure, a feature that should help to reduce the time needed to correctly implement, maintain, and enhance models
- some consistency checking of the natural language comments that SML encourages for documenting formal model specifications, and also some consistency and completeness checking of formal specifications by reference to those comments
- automatic generation of several kinds of model reference documentation that facilitate communication, debugging, model maintenance and evolution, and other essential activities
- automatic generation of relational data table designs for model instance data, a feature that facilitates relational database interfaces and achieves a kind of standardization that is essential for efficient implementation of several of FW/SM's capabilities
- automatic evaluation (with warm restart) of indexed families of real- and logical-valued expressions, a feature that facilitates debugging, "What If" studies, and other kinds of model and results analysis
- a browser for the definitional dependency graph, an SM construct useful for visualizing the general structure of any model
- complete independence between the general structure of a class of models and instantiating data, a feature that promotes the reuse of each of these, conciseness, ease of communication, dimensional flexibility, and other advantages
- complete independence between models and solvers, a feature that promotes using multiple solvers with a single model, multiple models with a single solver, and conceptual clarity.

Other features derive to a lesser extent from SM and SML, and to a greater extent from how the special capabilities of the host implementation package were exploited:

- tightly integrated basic services (viz., business graphics, file management, LAN support, macros, outlining, a rich programming language, spreadsheets, tabular database, telecommunications, and word processing) in addition to specialized modeling capabilities, a feature that promotes FW/SM's usefulness over the whole modeling life cycle rather than at just one part of it
- a menu-driven interface that lies dormant when not in use and thus does not interfere with normal use of the host software package, a feature that illustrates how even sophisticated modeling capabilities can be delivered via popular software
- the use of outlining, boldface, and underlining to simplify and enhance an underlying modeling language which, for technical reasons, is restricted to ASCII characters
- a tree-oriented editor for navigation, editing, and display control of the hierarchical structure of any model, a tool useful for managing the complexity of complicated models.

Finally, we mention FW/SM's main solver interfaces, each of which involves connection to a major external system:

- a fully automatic interface to a menu-driven commercial database system, a feature that facilitates queries and report generation for data and results
- a fully automatic interface for linear and integer programming based on MPS format problem files (no matrix generator need be written)
- a control table interface for generalized network flow optimization (no problem file generator need be written)
- a fully automatic interface to a commercial Prolog system, a feature that facilitates logic-based inferencing with respect to the most fundamental aspects of any model's general structure.

About half of these features are discussed at some length from the perspective of SML in Geoffrion (1990b).

FW/SM has been useful on several levels. At one level, it has enabled experimentation with SM concepts and with SML. At another, it has served as a vehicle for studying implementation issues pertinent to the design of modeling environments. And at still another level, experience with FW/SM has sharpened our vision of what an ideal modeling environment should be like from a user perspective, and of SM's prospects for helping to attain that ideal.

This article largely avoids operational and implementation details, which are the subject of separate documents (Geoffrion, Maturana, Neustadter, Tsai, and Vicuña 1990a, b).

The organization of the balance of this article is as follows. The remainder of this introduction surveys the other implementations of SM, reviews the basics of *Framework* (the system on which FW/SM is built), and indicates what prior knowledge of SM is presumed of the reader. §2 presents a descriptive overview of FW/SM's functions from the user viewpoint. §3 opines on the broader significance of selected aspects of FW/SM. The final section reviews the requirements given in Geoffrion (1989c) for a new generation of modeling environments and explains how FW/SM fulfills most of these within the limitations of a research prototype.

1.1. *Other Structured Modeling Implementations*

The first implementation of SM ideas, LEXICON (Clemence 1984), was done in FORTRAN on a mainframe and aimed mainly at supporting optimization for large linear programming problems. The model schema and data are entered as batch files. A link to an advanced optimization system was designed but not built.

IIS (for Integrated Information System, Farn 1985) was built on top of *KnowledgeMan* (Micro Data Base Systems 1985) on an IBM PC and aimed mainly at demonstrating the feasibility of a hybrid information/analytical modeling system. It includes a menu-driven interface to *LINDO* (Schrage 1991) for a restricted class of linear programs.

Because LEXICON and IIS were developed early in the evolution of SM and had quite special objectives, they support only a relatively small and nonstandard subset of today's SML.

ASUMMS grew out of a collaborative research effort involving several faculty and graduate students (Ramirez 1990). It adopts SML as its model description language. The architecture includes user, object, relational (*INGRES*), and physical layers. Model/solver/data independence supports the reusability of software and models. Translators map SML to *SAS*, *IFPS*, and 1-2-3. The hardware environment is a network that spans desktop computers, a minicomputer, and two supercomputers. A successor to ASUMMS called DAMS is now under way (Ramirez 1991).

Several other specialized or partial implementations of SM exist. These include Chari and Krishnan (1990) (a partial Prolog implementation of a logic-based alternative to SML), Hill (1989) (translates model schemas from algebra to SML), Jones (1991) (a prototype SM specialization of a general graphics-based modeling system), Lenard (1987) (built on top of *KnowledgeMan*, linear programming oriented), O'Dell (1988) (permits graphic SML schema input and translates to relational database form), Park (1987) (IS DOS-based), Worobetz and Wright (1991) (built on top of *ORACLE*, OR/SM has PC-SAS as its primary solver), and Wyant (1988) (translates SML schemas from relational database to graphic form). With the exception of Jones' Networks/SM and Worobetz and Wright's OR/SM, the capabilities of each of these implementations have but a modest intersection with the capabilities of FW/SM.

In addition, two new UNIX-based implementations are being or have been built at UCLA as part of doctoral dissertations. One, Desai (1992), uses the extensible database system *EXODUS* (Carey and DeWitt 1987) as its implementation platform. It will exploit novel capabilities that go beyond what traditional database management systems offer, including user-defined data types and access methods, enhanced integrity constraints, and version management. The other, Vicuña (1990), uses attribute grammars to specify formally the complete static semantics of SML, and to generate a syntax-directed editor with incremental type checking and immediate expression evaluation capabilities. It builds on top of the *Synthesizer Generator* (Reps and Teitelbaum 1989).

1.2. *Framework*

FW/SM is built on top of *Framework III*, a well-known integrated multi-function program that runs under DOS (Ashton-Tate 1988). One reason for this choice is that *Framework* makes pervasive use of trees for organizational purposes, as does SM. Another is that it offers many of the basic services needed for a true modeling environment, including outlining, word processing, tabular database, spreadsheets, business graphics, telecommunications, local area networking, and macro recording. It also includes a general-purpose interpreted language called FRED that supports all of these functions. A third reason for choosing *Framework* is that it can transparently run DOS batch files and external programs in other languages, and can move itself temporarily to hard disk or extended memory in order to make more room for loading and executing large programs. This makes it possible to achieve functionalities that would be awkward or inefficient to achieve through *Framework* alone, and to interface *Framework* via files with nearly any stand-alone program.

Prior familiarity with *Framework* would be helpful, but is not a prerequisite for reading this article because we now briefly summarize the essential concepts needed in what follows. The main ones are: the *Framework* desktop, the tree-oriented editor, and the table editor.

The *Framework* user sees a full-screen "desktop" on which there may be one or more "documents," each of which may be "open" (displayed in a window that can be resized and dragged) or "closed." There are also ten pull-down menus, and short labels sitting on the desktop that represent the disk drives and closed documents. The user's current selection is shown by a highlight that normally is on or in one of the labels, one of the pull-down menus, or one of the open documents. The general feel is somewhat Macintosh-like. Figure 1 shows an example in which there are two documents—BUDGET and FEEDMIX—on the desktop, the second of which is open. Note that there are five disk drives, plus a "Library" that holds abbreviations, macros, templates, and other tools; clicking on any of these will reveal its contents.

Each document corresponds to one DOS file, and consists of a user-designed tree of "frames." A frame is just a kind of window. The root of the tree corresponds to the document as a whole, each nonleaf node to a "containing frame," and each leaf node to a "word frame" containing text or a "database frame" containing a table whose columns can be named and specified as containing any of six data types (logical, numeric, string, formula, date, and time). Leaf nodes can also be "spreadsheet frames" or "graph frames," but these kinds are not needed in what follows. The display of a document toggles between an indented outline of its tree called the "outline view," and a "content view" of its tree based on a nested windows representation that reveals the contents of leaf frames. The user can also zoom any leaf frame so that its window fills the screen.

The open FEEDMIX document of Figure 1 is in outline view, from which one can see that its tree of frames has four first-level containing frames labeled "Schema Section," . . . , "Activity Section." The children of three of these four frames are hidden, but "Elemental Detail Section" is seen to have six leaf frame children, namely "NUTR," . . . , "TOTCOST." It happens that all six are database frames.

Framework's tree-oriented editor, or outliner, for manipulating trees of frames plays a key role. It has facilities to edit tree structure (create, delete, copy, and move nodes and subtrees), to navigate (move toward or away from the root, move to a sibling), and to control display (hide and unhide subtrees or node content).

Framework's table editor, which applies to database frames, also plays a key role in FW/SM. It offers data entry in any of three views (so-called "table," "forms," and "dBASE" views), flexible copy and move operations, row selection ("filtering") for any table via logical expressions and many built-in functions, sorting, column resizing (which can be used to hide columns temporarily), and the ability to compute a variety of column statistics. Word processing facilities also apply to database frames, including the string search, spelling checker, and automatic abbreviation expansion utilities.

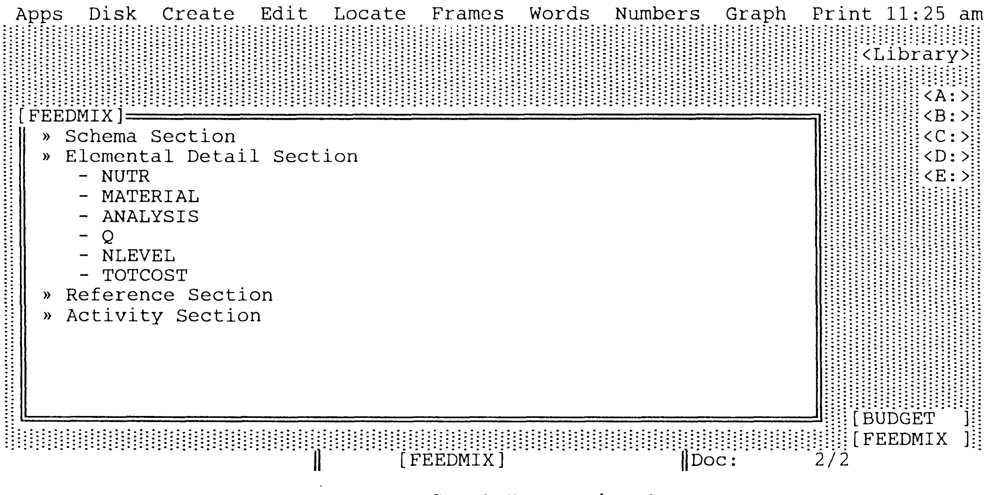
Framework's ability to communicate with other programs is facilitated by its file import/export capabilities in 9 standard formats. It can export any frame or selected part thereof as a file, and import any file as a frame.

Framework has many other capabilities that need not be described for an understanding of what follows.

Except for those few cases in which control passes to an external program, FW/SM's user interface is *Framework*'s user interface. Thus FW/SM is no more difficult to use than *Framework*.

1.3. *Background Knowledge*

Full comprehension requires prior exposure to SM at the level of §1–§3 of Geoffrion (1987). It is not necessary to have detailed knowledge of SM theory or of SML beyond what is explained there. Having a copy of that paper handy while reading this one will



```
Apps  Disk  Create  Edit  Locate  Frames  Words  Numbers  Graph  Print 11:25 am
                                                                     <Library>
                                                                          <A:>
[FEEDMIX]                                                                 <B:>
  » Schema Section                                                        <C:>
  » Elemental Detail Section                                              <D:>
     - NUTR                                                               <E:>
     - MATERIAL
     - ANALYSIS
     - Q
     - NLEVEL
     - TOTCOST
  » Reference Section
  » Activity Section

                                                                    [BUDGET  ]
                                                                    [FEEDMIX ]
                           [FEEDMIX]               Doc:        2/2
```

FIGURE 1. A Sample *Framework* Desktop.

prove convenient, as frequent references will be made to its concepts and figures. Because that paper is referenced so often in this one, we abbreviate its reference to "⟨MS 87⟩".

Familiarity with Geoffrion (1989b) would be helpful but is not necessary.

2. Overview of FW/SM

This section describes the user interface and main capabilities of FW/SM. Most operational and implementation details are omitted, but full details are available in Geoffrion, Maturana, Neustadter, Tsai, and Vicuña (1990a), in a series of internal technical documents (including Geoffrion, Maturana, Neustadter, Tsai, and Vicuña 1990b), and in a companion paper on design and implementation (Neustadter, Geoffrion, Maturana, Tsai, and Vicuña 1992). §3 appraises FW/SM's significance.

FW/SM implements SML as specified in Geoffrion (1990a), and attempts to do so in the spirit of the computer-based modeling environment vision presented in Geoffrion (1989c) (more on that in §4). It runs on IBM PC/AT and PS/2 class machines under DOS. After an orientation to the overall file and workspace structure of FW/SM as seen by the user, we discuss the major components separately.

2.1. *Overall Structure*

FW/SM offers about 20 special functions beyond those of Framework as menu-selectable processes. The names of these processes appear in italics (italics will henceforth be reserved for this use) and will be underlined on first appearance. There is a modest on-line help facility that covers all of the processes.

FW/SM involves four main sets of files:

FW/SM Programs	code that implements the special functions of FW/SM
AUXIL Files	model-specific intermediate files used by certain processes
Solvers	a collection of solvers for model manipulation (presently retrieval, optimization, and logical inferencing)
Models	model-specific "workspaces" in development or previously developed.

Each is discussed briefly.

FW/SM Programs. FW/SM's programs comprise about 70,000 lines of source code in C, Framework's FRED language, PASCAL, and three other languages. All programs run in a way that is transparent to the user, who sees only a menu of FW/SM processes that appears when a special key is pressed.

AUXIL Files. Certain processes which operate on a model schema produce outputs that are not intended for user access, but that are needed by certain other processes. Rather than generating these outputs afresh every time they are needed, FW/SM stores them in a model-specific subdirectory of a subdirectory named AUXIL. This leads to greater client process efficiency so long as the model schema does not change.

Solvers. Four solvers have been interfaced to date:

- Xtrieve, an interface and processor for database queries (Novell 1988)
- GENNET, an optimizer for generalized network flow problems (Brown and McBride 1984)
- LINDO, an optimizer for linear and integer programming problems (Schrage 1991)
- Arity Prolog, a language and processor for logic-based inferencing (Arity 1987).

In each case, there is an unmodified main executable file and associated custom interface programs.

Clearly it is possible to interface other solvers with FW/SM for retrieval, solving simultaneous systems, optimization, inferencing, and other model manipulations.

Models. The store of models will be unique to each installation and perhaps to each user of a modeling environment like FW/SM. At the time of this writing, we maintain a growing library of well over 100 models divided into eight categories: administrative, database, economics, engineering, MS/OR, marketing, test, and miscellaneous. Many are collected in Geoffrion (1990a). They run the gamut from very simple to moderately complex.

Model files are stored in Framework's internal format, and can be organized and stored in any convenient DOS subdirectory structure. Each holds a single Model Workspace, the four standard parts of which are the following first-level containing frames, or windows, of the Framework document that appears on the desktop when a file is loaded. See Figure 1.

"Schema Section"	Here the user composes, browses, and maintains a single SML schema.
"Elemental Detail Section"	Here the user loads, browses, and maintains the data tables—SML elemental detail tables—associated with a single schema.
"Reference Section"	Holds system-generated reference documentation corresponding to the schema and elemental detail; here the user browses and manipulates these exhibits using Framework's editors.
"Activity Section"	Here the user invokes solvers and performs other model manipulations and related activities throughout the modeling life cycle.

Only a Schema Section is mandatory if any of FW/SM's special processes are to be invoked. The user is free to add other first-level frames to supplement the four just mentioned.

The Model Workspace is the primary face of the system seen by the user. We discuss each of its sections individually.

2.2. *Schema Section*

We begin by recalling some facts about an SML schema. It is an outline-formatted collection of genus and module paragraphs that describes the general structure of a class of model instances (e.g., Figures 5, 7, and 10 of ⟨MS 87⟩). Figure 2 shows the schema for a simple feedmix model, to which we shall refer frequently in the balance of this paper. It is an updated version of Figure 5 of ⟨MS 87⟩. A genus is simply a class of similar elements, and a module is a collection of related genera or other modules. A schema consists of a formal part (in boldface) and an informal part, called the interpretation, documenting the formal part: module paragraph names begin with an ampersand; the type of each genus (primitive entity, compound entity, attribute or variable attribute, function, test) is declared within a pair of slashes; the definitional dependencies introduced by each genus appear immediately after the genus name in a so-called generic calling sequence delimited by parentheses; lower case letters not within a pair of slashes denote indices; curly braces denote index sets; a colon announces an attribute's type declaration; a semicolon announces the generic rule, that is, the numeric or logical expression associated, respectively, with a function or test genus; and key phrases in the interpretation are underlined.

Although one could hardly guess it from Figure 2, we comment for future reference that the generic rule sublanguage of SML is quite rich; one can see from Geoffrion (1990a) or (1990b) that its expressive power is comparable to that of expressions in the popular C language when the latter is restricted to its basic numeric, logical, and relational types. (Of course, the language paradigms are quite different: SML's generic rule sublanguage is function-oriented, whereas C is a statement-oriented imperative language.) Alternatively, one could say that the expressive power is comparable to that of any of the

&NUT_DATA NUTRIENT DATA
NUTRi/pe/ There is a list of NUTRIENTS.
MIN (NUTRi) /a/ {**NUTR**} **: Real+** For each NUTRIENT there is a MINIMUM DAILY REQUIREMENT (units per day per animal).

&MATERIALS MATERIALS DATA
MATERIALm /pe/ There is a list of MATERIALS that can be used for feed.
UCOST (MATERIALm) /a/ {**MATERIAL**} Each MATERIAL has a UNIT COST ($ per pound of material).
ANALYSIS (NUTRi, MATERIALm) /a/ {**NUTR**} × {**MATERIAL**} **: Real+** For each NUTRIENT-MATERIAL combination, there is an ANALYSIS (units of nutrient per pound of material).

Q (MATERIALm) /va/ {**MATERIAL**} **: Real+** The QUANTITY (pounds per day per animal) of each MATERIAL is to be chosen.

NLEVEL (ANALYSISi., Q) /f/ {**NUTR**} **; @SUMm (ANALYSISim*Qm)** Once the QUANTITIES are chosen, there is a NUTRITION LEVEL (units per day per animal) for each NUTRIENT calculable from the ANALYSIS.

T:NLEVEL (NLEVELi, MINi) /t/ {**NUTR**} **; NLEVELi >= MINi** For each NUTRIENT there is a NUTRITION TEST to determine whether the NUTRITION LEVEL is at least as large as the MINIMUM DAILY REQUIREMENT.

TOTCOST (UCOST, Q) /f/ ; @SUMm (UCOSTm*Qm) There is a TOTAL COST (dollars per day per animal) associated with the chosen QUANTITIES.

FIGURE 2. Feedmix Model Schema.

popular spreadsheet packages (e.g., 1-2-3™) if macro facilities are excluded, although SML's generic rule sublanguage has much more general indexing capabilities.

Given Framework's frame orientation and its versatile outliner for organizing and manipulating frames, it is natural to represent a schema's hierarchical (tree) structure as a Framework outline with the Schema Section itself as the containing frame corresponding to the root. Then each module paragraph is also a containing frame, and each genus paragraph is a word frame. With this approach, the outliner's tree-oriented editing facilities are well matched to the tasks of composing and editing a schema's hierarchical structure.

What appears in Figure 2 actually is a variant of SML that exploits Framework's word processor and outliner to relieve the user of having to enter reserved words to mark the end of the formal part of each paragraph, to mark the end of each paragraph, to delimit key phrases, and to mark the end of the schema (Geoffrion 1990a). The transition from boldface to nonboldface, the use of underlining, and the use of outlining cause the necessary reserved words to be inserted automatically as a first step in processing the schema.

Two FW/SM processes associated with the Schema Section are of particular interest. The first is *SMLCheck*, a fully automatic syntax and static semantics checker for the formal part of all SML paragraphs. In addition to checking all syntax, it checks all of the approximately 90 so-called "Schema Properties" given in Geoffrion (1990a). These are context-sensitive static semantic restrictions guaranteeing the integrity of SML schemas. *SMLCheck* also produces an internal tabular representation of the schema that is needed by many other processes, and stores it in a model-specific subdirectory of AUXIL.

Whereas *SMLCheck* applies to the formal part of an SML schema, *INTERP_CK* applies to the interpretation part. It checks whether SML's rules for key phrase syntax and usage have been followed, and it generates an interactive dialogue aimed at identifying key phrase variants. To use Figure 2 as an example, the user must indicate in response to an *INTERP_CK* query that "NUTRIENT" in the MIN paragraph corresponds to "NUTRIENTS" in the NUTR paragraph.

As is true for all FW/SM processes, the user is responsible for invoking *SMLCheck* and *INTERP_CK* by simple menu selection. They produce diagnostics that pinpoint departures from the syntactic and higher order structure prescribed for SML schemas.

In addition, two other FW/SM processes operate on the Schema Section: *FORMAT* sets frame formatting parameters so that printing and the other FW/SM processes will operate correctly, and *COSMET* (for "Cosmetics") trims and sizes all frames to make good use of screen space.

2.3. *Elemental Detail Section*

SML's elemental detail tables, recall, provide the instantiating data for a single model instance falling within the model class described by the associated schema. Figure 6 of ⟨MS 87⟩ gives sample tables for the schema of Figure 2, and one of these is reproduced near the end of this subsection.

In Framework, it is natural for each elemental detail table to be a database frame. Framework's table-oriented editor is well suited to most keyboard-oriented input and editing tasks arising for elemental detail tables. Framework can import data from external files, but it must be said that more efficient storage structures and links to external files and databases will be needed for most real applications. We comment further on this in §4.4.

Two FW/SM processes associated with the Elemental Detail Section are of particular interest, and will be discussed further in §3. The first, *EDGEN* (for "Elemental Detail Table Generator"), reads the schema in the Schema Section and creates skeletal elemental detail tables in compliance with the Table Structuring Procedure and Table Naming Conventions of Geoffrion (1990a). It is fully automatic unless the user opts to join tables per Step 3 of the aforementioned procedure.

The second process, *FcEval* (for "Function Evaluator"), is a translator that creates a batch-mode evaluator consisting of compiled C code for the generic rules of all function and test genera (these are the objective and constraint expressions in the case of a mathematical programming model). This code, which is stored in a model-specific subdirectory of AUXIL, can be executed at any time thereafter by invoking the *Evaluate* process.

FW/SM offers some additional processes applicable to the Elemental Detail Section. *L/E Utilities* (L/E for "Loader/Editor") is a collection of utilities that facilitates loading and editing the element populations of genera by performing requested relation algebraic operations (e.g., Date 1981) on elemental detail table stubs. (Recall from ⟨MS 87⟩ that the "stub" of an elemental detail table comprises the column(s) to the left of the vertical double lines, and that it is the index for that table.) Included are utilities for:

- 5difference (two stubs),
- intersection (two stubs),
- union (two stubs),
- Cartesian product (two or three stubs),
- natural join (two stubs),
- projection (one stub).

In addition, a utility is provided for deleting rows with duplicate stub tuples. These facilities are preparatory to the "smart loader/editor" discussed in §4.

Processes are needed to implement the "Table Content Rules" of Geoffrion (1990a), which guarantee the integrity of the elemental detail tables. Two such processes have been implemented to date, both of which pertain to Table Content Rule A. *A_RULE* checks the validity of all identifiers appearing in the stub of elemental detail tables corresponding to externally indexed genera. Such an identifier is "valid" if it is defined in the elemental detail table for its associated self-indexed genus. (Every SML genus is unindexed like TOTCOST, or self-indexed like NUTR and MATERIAL, or externally indexed like all the rest in Figure 2.) *UNIQUE* looks for repeating tuples in the same stub (these are not permissible). About nine more processes would be necessary to fully implement the remaining Table Content Rules.

Finally, *MATRIX* transforms an elemental detail table with exactly two stub columns and at least one value column into a two-dimensional array in a database frame. For example, *MATRIX* can transform this elemental detail table associated with the feedmix model's ANALYSIS genus

NUTR	MATERIAL	ANALYSIS
P	std	4
P	add	14
C	std	2
C	add	1

into the array

NUTR	add	std
P	14	4
C	1	2

or its transpose. Two-dimensional arrays are easier to interpret for some purposes than the linearized version used by elemental detail tables.

2.4. *Reference Section*

There are many different kinds of reference documentation that one might want to see in connection with a schema, whether it is newly composed or called up from the model files. The same is true, although to a lesser extent, of elemental detail tables. Instead of trying to anticipate and produce all of the documentation that might ever be needed, FW/SM's strategy is to produce several basic machine-resident documents containing the essentials and let the user manipulate them interactively to obtain the specific reports and analyses of interest.

The manipulation can be by Framework's word processor for text documents, by its outliner and word processor for outlines, and by its table editor and word processor for tabular documents. In addition, Framework's rudimentary graph editor can be used for business graphs, although none of FW/SM's special processes produce these.

The reference document names, the processes that produce them, and their basic type are as follows:

Calling Sequence	*REFGEN*	Table
Genus Graph	*NETDRAW*	Graph
Genus/Module Summary	*EDGEN*	Table
Identifier Dictionary	*ID_DICT*	Table
Matrices		
Adjacency Matrix	*REFGEN*	Table
Reachability Matrix	*REFGEN*	Table
Adjacency/Reachability Matrix	*REFGEN*	Table
Modular Outline	*REFGEN*	Text
Natural Language Summary	*REFGEN*	Text
Table Structure	*EDGEN*	Table
Topological Sort	*REFGEN*	Table

Space limitations permit only a brief introduction to these documents here. See the Appendix of Geoffrion (1991b) for details.

The Calling Sequence document lists the direct definitional dependencies of each genus.

"Genus Graph" is the official name of the definitional dependency diagram. Figures 3, 8, and 12 of ⟨MS 87⟩ and §3.3 thereof give examples and explain its importance for communication purposes. The NETDRAW process is an interactive implementation that allows any one node of the genus graph to be displayed centrally on screen along with all adjacent arcs and nodes. The genus graph can be "walked" with the cursor keys, which move a highlight that can change the central node to one of its neighbors.

Figure 3 exemplifies the Genus/Module Summary. The NAME column contains all genus and module names in schema order. The SEQ_NO and PATH columns provide two reference numberings of these names, one strictly sequential and the other with hierarchical structure encoded in a standard way. The TYPE column indicates genus type, with modules listed as "m." The TABLE column gives the name of the elemental detail table, if any, corresponding to each genus. Finally, the KEY PHRASE column gives the first underlined key phrase, if any, in the interpretation part of each genus and module paragraph.

Sorting is a very useful manipulation of the Genus/Module Summary. Sorting on NAME produces an alphabetized dictionary of genus and module names, on TYPE produces a list of the modules and a list of the genera grouped by type, on TABLE produces an alphabetized list of tables and the genera associated with each, and on KEY PHRASE produces an alphabetized list of key phrases. Filtering via logical expressions can also be useful.

The Identifier Dictionary gives an exhaustive list of all identifiers appearing in the stubs of elemental detail tables corresponding to self-indexed genera.

The matrices capture the structure of the genus graph and all module graphs (defined in Geoffrion 1989b). Row and column orders are determined by the schema. The convention is "column depends definitionally on row"; thus, reading by columns indicates which row genera definitionally influence a given column genus, and reading by rows indicates which column genera a given row genus influences. The Adjacency Matrix portrays only direct definitional dependence as reflected in generic calling sequences, while the Reachability Matrix portrays all indirect dependencies (definitional dependence is a transitive relation). The Adjacency/Reachability Matrix uses a coding that indicates both. It follows from the above conventions that, in view of the no-forward-reference property of SML schemas, all three matrices are necessarily upper triangular.

To mention but one possible use of one of these matrices (the Reachability Matrix): if a definitional or value error is discovered in a particular attribute genus, then one can tell at a glance which other function or test genera could possibly be affected.

REFGEN actually produces all three matrices as word frames. However, the DBREF process converts them automatically to database frames. Thus one may manipulate the

NAME	SEQ_NO	PATH	TYPE	TABLE	KEY PHRASE
&NUT_DATA	1	1	m		NUTRIENT_DATA
NUTR	2	1.1	pe	NUTR	NUTRIENTS
MIN	3	1.2	a	NUTR	MINIMUM_DAILY_REQUIREMENT
&MATERIALS	4	2	m		MATERIALS_DATA
MATERIAL	5	2.1	pe	MATERIAL	MATERIALS
UCOST	6	2.2	a	MATERIAL	UNIT_COST
ANALYSIS	7	2.3	a	ANALYSIS	ANALYSIS
Q	8	3	va	Q	QUANTITY
NLEVEL	9	4	f	NLEVEL	NUTRITION_LEVEL
T:NLEVEL	10	5	t	NLEVEL	NUTRITION_TEST
TOTCOST	11	6	f	TOTCOST	TOTAL_COST

FIGURE 3. Genus/Module Summary for Feedmix Model.

matrices using either Framework's word processor alone on a word frame or together with Framework's table editor on a database frame.

The concept of a Modular Outline was introduced in §2 of ⟨MS 87⟩ and illustrated in Figures 4, 9, and 14 of that article. It is automatically available as Framework's "outline view" of the Schema Section, to which there is easy toggle access at all times. This gives a concise representation of the schema's hierarchical structure, and is useful for communication and documentation. Subtree hiding/unhiding can be accomplished easily using Framework's outliner, and is a particularly useful manipulation. For example, in an on-line briefing one may start with everything hidden except the top level of the hierarchy, and selectively unhide and re-hide detail as appropriate. In addition to this version of the Modular Outline, *REFGEN* produces a word frame containing an enhanced Modular Outline that appends the free indices, if any, to each genus name.

The Natural Language Summary was introduced in §3.3 of ⟨MS 87⟩ and illustrated in Figure 15 of that article. As explained here, it is identical to the schema except that it replaces the formal part of each genus paragraph (everything but the interpretation part) by the genus name with free indices appended as in the enhanced Modular Outline produced by *REFGEN*. The Natural Language Summary associated with a schema is especially useful for communication and documentation at a nonmathematical level because it suppresses mathematical details.

The Table Structure document gives the names of all elemental detail tables, in order, and their columns. It also gives the "type" of each table, a technical attribute defined in Geoffrion (1990a).

The Topological Sort document contains the so-called "topological labels" discussed at length just after Proposition 7 of Geoffrion (1989b).

2.5. *Activity Section*

The Activity Section is the venue of choice for most user interactions with solvers. FW/SM provides one process for retrieval, two for optimization, and one for logical inference. They are discussed in that order.

Retrieval. Three illustrative queries that could arise in connection with feedmix models are:

(a) List the materials in decreasing order of unit cost.
(b) List the materials used in quantity greater than 1 pound per day per animal and with unit cost greater than 2 dollars per pound.
(c) List the materials with above average analysis in those nutrients for which the current mix fails the nutrition test.

The ability to answer such queries is important, over a modeling application's life cycle, in order to gain full advantage of the information that resides in the Elemental Detail Section.

Queries that involve but a single elemental detail table warrant distinction from those that involve more than one. The first query above involves just one elemental detail table, while the other two involve two tables. Most queries involving a single table can be answered easily with the help of Framework's table editor. But queries involving multiple tables are awkward for Framework, and best handled via FW/SM's interface to the commercial database system Xtrieve (Novell 1988).

The *XtrieveQuery* process automatically exports the elemental detail tables to Xtrieve in a suitable format, invokes Xtrieve with proper initial conditions, and deposits the user in the menu-driven Xtrieve query interface. Upon exiting, the user returns automatically to FW/SM.

Xtrieve is a quasi-relational system, and so is able to answer most of the usual relational queries, including those that involve multiple tables. The full power of Xtrieve is available to the user, including:

- browsing, sorting, joining, reformatting, and relational selection via complex logical conditions
- the usual aggregate functions for calculating column statistics
- access to a data dictionary summarizing the design of all tables
- the ability to edit the data in any table and to define new tables that may or may not be based on the original elemental detail tables (this does not induce any changes in the Elemental Detail Section, although that would be an easy capability to install)
- a recorder for commonly used keystroke sequences
- various file management and printing facilities, including table export in three formats.

Optimization. Usually there are many possible optimization problems that a user could pose with respect to a given structured model. The user must, therefore, specify a particular optimization problem to be solved. Different interface styles are possible by which the user may make this specification. One way in which they may differ has to do with how much the user needs to know about the layout of the file(s) required by the intended solver. FW/SM has one solver interface that requires the user to know nothing, and another that requires the user to know a lot.

The *MPS_Interface* process for solving linear and integer programs is fully automatic. The user need only prepare a word frame in the Activity Section titled MPS that contains the keyword MINIMIZE or MAXIMIZE followed by the name of the unindexed function genus corresponding to the objective function. The process assumes that the variables coincide with the variable attribute elements, and, unless otherwise stipulated, that the constraints coincide with the test elements (i.e., they must all evaluate to TRUE). For example, the frame

```
MPS══════════════════╗
║ MINIMIZE TOTCOST   ║
╚════════════════════╝
```

would specify the standard linear programming problem for the feedmix model.

MPS_Interface does the following:

1. Read the schema, elemental detail tables, and MPS frame.
2. Verify that the specified problem is indeed a linear program, possibly with some integer variables, after evaluation of all (possibly nonlinear) expressions that do not depend on the variable attribute values.
3. Construct an appropriate standard MPS input file.
4. Run LINDO.
5. Import the optimal solution back into Framework with all variable values clearly identified (LINDO's 8 character name length limitations are handled automatically).

This process could be adapted easily to interface any optimizer that accepts standard MPS input files. Also, it can be commanded to stop after step 3, to allow manual control of the solution process from this point on. An extension to nonlinear programming would not be difficult.

The other solver interface, *GENNETFW*, is for the generalized network flow optimizer GENNET (Brown and McBride 1984). It requires a schema-specific control table in the Activity Section, a nonprocedural device by which the user maps the elemental detail tables into a GENNET input file. This input file organizes the data for arcs in records whose fields are: Tail Node, Head Node, Unit Cost, Upper Limit, Lower Limit, Multiplier. There are conventions for representing the node conservation conditions in records of this same form, including both equality and interval supplies and demands. The user must know these conventions, but they are of no concern here.

An important point is that the control table need never be changed so long as the schema remains the same; inserting, deleting, or updating rows in the elemental detail tables affects neither the content nor the size of the control table. Figure 4 illustrates this

TAIL	HEAD	COST	*U*	*L*	*M*
LINK.PLANT	LINK.CUST	LINK.COST			
PLANT.PLANT	PLANT.PLANT		PLANT.SUP		−1
CUST.CUST	CUST.CUST		CUST.DEM	CUST.DEM	

FIGURE 4. *GENNETFW* Control Table for Transportation Model.

table for the classical transportation model appearing in Figures 10 and 11 of ⟨MS 87⟩. The syntax is straightforward, and the column headings are standard for all *GENNETFW* control tables. Each line generates a group of arcs or node conservation conditions. The first line can be interpreted as follows: generate arcs for all rows in table LINK, with the tail nodes taken from the PLANT column and the head nodes taken from the CUST column; use unit costs from the COST column of table LINK, infinite upper flow limits, zero lower flow limits, and no gains or losses on the flows. The second and third lines generate interval supplies at the plants and equality demands at the customers.

GENNETFW does the following:

1. Read the elemental detail tables and the control table.
2. Construct the GENNET input file specified by the control table.
3. Run GENNET.
4. Import the optimal solution back into Framework with all variable values clearly identified (GENNET's requirement for consecutive integer node names is handled automatically).

Logical Inference. One popular approach to logic-based inference is via the Prolog language, all implementations of which incorporate an inference engine. FW/SM's *PrologQuery* process enables this kind of inference to be accomplished with respect to the basic structural information of an SML schema. It does this by automatically translating this information into Prolog predicates (facts and rules), exporting these along with certain other "stock" predicates to Arity Prolog (Arity 1987), invoking Arity Prolog with proper initial conditions, and depositing the user in the command-driven Arity Prolog environment. The full power of Arity Prolog is available to the user for answering ad hoc or pre-packaged logical queries based on the available schema information. Upon exiting, the user returns automatically to FW/SM.

The basic structural information available for inferencing is the following: all genus and module names, the type of each genus, all definitional dependencies (including, for each, which component of the calling genus' generic calling sequence does the calling), and the schema's hierarchical structure (including the order for each sibling set). The approach used for translating this information into fact-type predicates suitable for Prolog was devised by Chari (1988). No additional details of an SML schema, nor any of the information in elemental detail tables, are translated into predicate form. It would have been straightforward to do so, but this would have been only marginally useful to our research program.

Inferencing capabilities are expanded by the "stock" predicates mentioned earlier. These rule-type predicates, also devised by Chari (1988), capture the relevant schema-related core concepts of Geoffrion (1989b). It is not so much the final predicates corresponding to these rules that are of use—for these must be satisfied by any SML schema that passes *SMLCheck*—but rather the various preparatory predicates introduced to facilitate defining the final predicates.

As an example for those familiar with Prolog, suppose that one wishes to list all primitive entity genera that contribute directly or indirectly to the definition of a particular genus "gname." To do this, one could define the predicate call_pe(X, Y) :- gcalls_ms(X, Y), agt(Y, pe)

and then run it in the form

call_pe(gname, Y).

Here *agt*(*Y*, *pe*) is a fact-type predicate which says that genus *Y* is of primitive entity type, and *gcalls_ms*(*X*, *Y*) is a preparatory predicate which is true if genus *X* definitionally depends directly or indirectly on genus *Y*.

PrologQuery offers a radically different alternative from the Reference Section for accessing selected Schema Section details for purposes of ad hoc query and reporting. These approaches are complementary.

3. Significance of FW/SM

The previous section gave a factual description of the overall structure of FW/SM and the function of each of its special processes. This section, by contrast, is opinionated. It discusses certain features of FW/SM in terms of their broader significance, with particular attention to the implications of FW/SM for designers and evaluators of modeling systems and environments.

We incorporate into this discussion many of the lessons learned from the FW/SM project, but leave to the companion paper Neustadter et al. (1992) most of the more technical lessons.

This section takes a top-down approach, beginning with Framework's desktop environment and continuing with the model collection, the interfaced solvers, the overall design of Model Workspaces, and some of the processes.

3.1. *Framework's Desktop Environment*

The decision to build the prototype on top of Framework not only had a profound effect on overall system architecture, but it also largely determined the user interface and what basic services would be available to the user. We leave a discussion of the first effect to the companion paper cited above.

Framework's user interface—reminiscent of Macintosh's and with a strong flavor of direct manipulation (Shneiderman 1987)—has proven to be easy to use and appropriate to the demands of analytical modeling, at least within the limited range of our experience. This positive assessment is consistent with the current strong commercial trend toward graphical user interfaces, both in general and in the particular case of analytical modeling software.

Framework's basic services also have proven, in our trials, to be convenient and compatible with the demands of analytical modeling. As advocated in ⟨MS 87⟩ and Geoffrion (1989c), a modeling environment should support as much of the full modeling life cycle as possible; this in turn requires

> . . . *a high degree of software integration, especially with respect to tools and utilities for communication* (*e.g., business graphics, telecommunications, and word processing or even desktop publishing*), *for organizing things and ideas* (*e.g., configuration and version control, database management, file management, outlining, and project management*), *and for quantitative analysis* (*e.g., data acquisition, interactive data analysis, graphics, mathematical, spreadsheet, and statistical programs*). (Geoffrion 1989c)

Framework certainly exhibits the requisite degree of software integration, although it does not provide all the services just listed. It does provide outlining, word processing, table editing, file management, macros, and a rich built-in programming language. These have been essential. It also provides spreadsheet modeling, business graphing, and telecommunication facilities, which have proven useful on occasion; and LAN support, which we happen not to have used as yet.

It would be possible to integrate some of the missing services into FW/SM, but it is not necessary to do so for FW/SM users to appreciate the value of well-integrated access to tools and utilities like the ones provided. A few workaday examples will underscore this point: (1) Framework's outliner is superb for navigation, editing, and display control of the hierarchical structure that organizes the genus paragraphs in the Schema Section; (2) Framework's table editor is indispensable for working with the tables in the Elemental Detail Section, and for manipulating the documents in the Reference Section; (3) Framework's copy, move, and cut-and-paste facilities find regular use as a way to transfer material among different parts of a Model Workspace, and between Model Workspaces and external files and documents being written with Framework's word processor.

Two other points deserve mention. One is that FW/SM behaves exactly like Framework until the user decides to invoke one of its processes by calling up the special menu. This style, which leaves the user in full control of a familiar commercial software package rather than having to learn a totally unfamiliar modeling system, may turn out to be one of the few avenues leading to much broader acceptance of modeling tools. FW/SM shows that even sophisticated modeling capabilities can be delivered in this way. In a word, FW/SM illustrates a "piggyback" approach to modeling environment design and implementation which saves the cost of implementing the host package's services.

The second point is that modern word processors offer format and font capabilities like outlining, boldface, italics, underlining, subscripts, and superscripts that can be harnessed to improve the readability of model description languages that, for technical reasons having to do with their processing, are limited to a linear string of standard characters. As explained in §2.2, FW/SM exploits some of these capabilities in order to relieve the user of having to enter certain reserved words that serve purely technical purposes. It seems inevitable that format and font options will increasingly be exploited to enhance modeling language readability.

3.2. *Model Collection*

DOS subdirectories have proven satisfactory for physically organizing our collection of models. We also use a simple structured model for conceptually organizing this collection. For each Model Workspace, it contains the filename, origin, category, maintainer, domicile, and date of last revision. For maximum usefulness this model should, of course, be maintained automatically by the modeling environment instead of requiring manual updating as at present.

One of the three major design challenges posed in Geoffrion (1989c) is that a modeling environment should be based on a conceptual modeling framework of great generality. As explained there (see also ⟨MS 87⟩ and Geoffrion 1989b), structured modeling responds to that challenge by formalizing a general class of definitional systems, for which SML furnishes a notation. The diversity of the more than 100 models in our collection confirms the alleged generality. A printed extract from this collection is available in Geoffrion (1990c).

Some of our models are SML translations of models appearing in the works of other authors. These permit comparisons between SML and FW/SM on the one hand, and alternative modeling styles and systems on the other. A few examples:

- the static Mexican steel industry model from Kendrick, Meeraus, and Alatorre (1983) enables comparison with GAMS (Brooke, Kendrick, and Meeraus 1988) for economic planning
- the production model from Ellison and Mitra (1982), Lucas, Mitra, Darby-Dowman (1983), and Hürlimann and Kohlas (1988) enables comparisons with the UIMP, CAMPS, and LPL systems for linear programming
- the distribution model from Fourer, Gay, and Kernighan (1990) enables comparison with AMPL

- the comprehensive accounting model from LeBlond and Cobb (1985) enables comparison with 1-2-3
- the educational database example from Chapter 27 of Date (1981) enables comparisons with relational, hierarchical, and network database systems
- the shipping industry example from Hammer and McLeod (1981) and the world traveler example from Hull and King (1987) enable comparisons with semantic database systems.

Having the ability to express models from different modeling paradigms in a single language, rather than in disparate languages, makes it easier for FW/SM users to link together disparate models to form a more comprehensive model (⟨MS 87⟩, Geoffrion 1989a). This extends the evident benefits of being able to compose new models by starting with existing ones, much as one might compose a routine business letter by modifying a previous one.

3.3. *Interfaced Solvers*

FW/SM's automatic interfaces with commercial-quality solvers for retrieval, optimization, and inference conform to the sharp separation of models and solvers advocated in ⟨MS 87⟩.

FW/SM shows that a modeling environment can support a wide variety of solvers; e.g., it does not have to cater only to optimizers. This enables synergies that are not possible with just one solver or even one category of solvers. Indeed, we believe that the ability to manipulate a given model in a wide variety of ways will be one of the characteristics that distinguish today's modeling systems from tomorrow's modeling environments.

3.4. *Model Workspace*

The division of each Model Workspace into four main sections (windows) has worked out largely as intended. This is particularly true for the Schema Section, Elemental Detail Section, and Reference Section. For example, the cleavage between the Schema Section and the Elemental Detail Section helps to fulfill the objective of general structure/detailed data independence as advocated in ⟨MS 87⟩.

However, the Activity Section has not been as exclusive a focus for model manipulation activities as originally expected. There seem to be two main reasons for this. First, the *XtrieveQuery* and *PrologQuery* processes transport the user to the interactive user interfaces which these external solvers provide. Second, it is often convenient to create additional frames outside of the Activity Section for a variety of ad hoc purposes, rather than creating these same frames within the Activity Section containing frame.

This experience suggests that it may be appropriate for modeling systems and environments to accommodate a good deal of flexibility as to how users organize model manipulation activities.

3.5. *The Processes*

We now comment on the significance of some of FW/SM's processes.

One point that applies to many of the processes is that they have been implemented so as to advance the important objective of general structure/instantiating data independence. Specifically, *SMLCheck*, *INTERP_CK*, *REFGEN*, *NETDRAW*, *EDGEN*, *FcEval*, and *PrologQuery* all work in the absence of any detailed data. Not all extant modeling systems exhibit this sort of independence. For example, GAMS, a leading modeling system for mathematical programming, requires detailed data before a GAMS schema can be processed in any way.

Check General Structure (*SMLCheck*). With the array of compiler-construction tools now available, especially for important classes of context-free grammars (e.g., Aho, Sethi, and Ullman 1986), it is now commonplace for the context-free component of a modeling language to be defined formally and used as the basis for the lexical and syntactic analysis parts of their implementation. Thus lexical and syntactic error-checking have become routine, and *SMLCheck* does indeed accomplish this for SML.

But *SMLCheck* does much more than this: it checks the SML Schema Properties. Surprisingly, these properties are exhaustive in an important sense. Proposition 2 of Geoffrion (1990a) shows that a schema which satisfies all Schema Properties is guaranteed to represent a genuine class of structured models meeting certain desirable qualifications. A Schema Section that satisfies all Schema Properties may not fulfill all of the intentions of the modeler, but at least it is internally consistent and free of essentially all errors that can be checked for in the absence of domain-specific knowledge and meta-knowledge not expressible in SML.

This result is possible only because SML was designed to support an explicit semantic framework, that is, a notation-free conceptual formalism for modeling (Geoffrion 1989b). Such a result is not possible for a language whose semantic framework is implicit, as is the usual case.

A modeling environment like FW/SM whose checks on general model structure are potentially exhaustive in the sense described should make it easier and quicker to achieve correct models than is possible in a modeling environment whose checks are *ad hoc* and not necessarily exhaustive. This same point applies with respect to FW/SM's potential enforcement of exhaustive checks on model instance data, to which we now turn.

Check Instantiating Data (*A_RULE* and *UNIQUE*). The significance of *A_RULE* and *UNIQUE* is that they represent some of the Table Content Rules, which play the same sort of role with respect to elemental detail tables as Schema Properties play with respect to a schema.

Proposition 1 of Geoffrion (1990a) essentially asserts that, if a Schema Section passes *SMLCheck* and a corresponding Elemental Detail Section abides by all Table Content Rules and has 100% fill for all attribute value columns, then the result must be a genuine structured model instance. This model instance may or may not fulfill all of the intentions of the modeler, but—to repeat a comment made with respect to Proposition 2—at least it is internally consistent and free of essentially all errors that can be checked for in the absence of domain-specific knowledge and meta-knowledge not expressible in SML.

FW/SM does not enforce all Table Content Rules at present, although there is no technical reason why it could not be made to do so. However, there is another implementation using a more advanced software development technology for which the enforcement of all Table Content Rules has been designed and largely implemented (Vicuña 1990). It also implements the enforcement of all Schema Properties. This shows that it is possible for a structured modeling environment to enforce syntactic and semantic restrictions, on both general structure and data, that are complete in the sense of the two propositions cited above.

Check Documentary Comments (*INTERP_CK*). SML imposes certain mild conventions on the interpretation part of genus and module paragraphs (see Appendix 5 of Geoffrion 1990a or §2 of Geoffrion 1990b). This encourages modelers to be conceptually precise, and enables *INTERP_CK* to perform several checks on the consistency between the formal and informal parts of a schema. These checks can detect some kinds of inconsistency and incompleteness in the formal part, and inconsistency in the informal part. We believe that schemas which pass these checks are generally higher in quality, more useful, more maintainable, more readable, and better documented than those prepared without documentary comments obeying such conventions.

Our experience has been consistent with this belief. Taking the time to write interpretations for all schema paragraphs usually pays dividends, most obviously in terms of model quality. We have often had the experience of writing a schema quickly without any interpretation at all, only to discover mistakes later when going back and filling in the interpretations. Consequently, we recommend that modeling system designers consider providing for plain language documentation with a mild but purposeful discipline that is machine-checkable.

Generate Skeletal Tables (*EDGEN*). The elemental detail table designs created by *EDGEN* constitute a relational database scheme (e.g., Date 1981). This sets the stage for the elemental detail tables to be supported by a relational database system. This is an important opportunity, although FW/SM does not truly exploit it since FW/SM's database frames and table editor do not constitute a true "relational database system." The *XtrieveQuery* process, which we discuss later, comes much closer.

For reasons that are widely accepted by the database community, it is desirable for the data tables in a modeling environment to be free of insertion, deletion, and update anomalies arising from functional dependencies (e.g., Date 1981). Those produced by *EDGEN* essentially are; see Neustadter (1990) for a comprehensive treatment of this result.

The fact that *EDGEN* is automatic confers several advantages. One is that the modeler need expend no effort deciding how to organize instantiating data. This also removes opportunity for error. Another is that standardized table design rules give all models a common "look and feel." A third is that it facilitated the implementation of FW/SM's data-driven processes.

Compile Generic Rules into C (*FcEval*) **and Execute** (*Evaluate*). One of the things that distinguishes analytical modeling from data modeling is the frequent occurrence of mathematical expressions, especially indexed families of expressions. In structured modeling, these families are represented by indexed function and test genera. The generic rule of each such genus gives a typical expression that applies, with appropriate specializing changes, to each element. It is necessary to be able to evaluate all these expressions.

Since the advent of spreadsheets, users are unlikely to be satisfied with any modeling system or environment that fails to provide the immediate feedback of a resident evaluator. Nor should they be, for immediate evaluation is one of the most useful of all debugging tools for both models and results derived by model manipulation (e.g., optimization). It is also an essential facility for answering "What If" questions, for doing many kinds of model analysis and results analysis, and for generating reports. To give just one example, it is important in most applications of optimization to be able to recalculate the complete consequences of making manual overrides to the "optimal" solution determined by the solver. See ⟨MS 87⟩ for further motivation.

FcEval automatically produces a suitable resident evaluator. It compiles each generic rule into executable code that computes the value of every element of its genus. This code runs whenever the *Evaluate* process is invoked. Note the following points:

1. *FcEval* separates compilation based on general structure from execution based on instantiating data. Thus, it need only be invoked once so long as the Schema Section is not changed in a way that could affect generic rule evaluation, and arbitrary edits can be made to the elemental detail tables (insertions, deletions, and updates of all kinds) without having to reinvoke *FcEval*. This "warm restart" capability contributes to the efficiency of *Evaluate* in the usual case where only some of the data change from evaluation to evaluation. Moreover, *FcEval* produces compiled C code rather than interpreted code, which would be less efficient.
2. *Evaluate* puts evaluation results into the elemental detail tables where they are readily accessible to the user, to Framework's table editor, and to other processes.

The *FcEval* compiler was one of the most time-consuming of all processes to implement because, as noted in §2.2, the generic rule sublanguage is quite expressive.

The equivalent of *FcEval* is unusual in today's modeling systems. For example:

- Database systems can conveniently handle only the simplest kinds of mathematical expressions.
- Spreadsheet packages usually rely on interpreted rather than compiled code, and require laborious individual entry of expressions in an indexed family unless it happens to be possible to use the copy command to generate the proper family of individual expressions from a single family member.
- AMPL (Fourer, Gay, and Kernighan 1990) has no resident evaluator, relying instead on whatever optimizer it is linked to. In any case, nonprogrammer users must recompile model structure and data *ab initio* if even a single edit is made to the data, and the results of optimizer-executed evaluation are likely to be in a flat file format of limited direct utility. (It is noteworthy that extensions of AMPL's operating environment are in progress which include development of capabilities similar to resident evaluation.)
- GAMS (Brooke, Kendrick, and Meeraus 1988) does have a resident evaluator but, like AMPL, nonprogrammer users must recompile model structure and data *ab initio* if even a single edit is made to the data. Moreover, evaluation results are presented in a flat file format of limited direct utility.

A resident evaluator with the properties mentioned in points 1 and 2 above is likely to be an important component of the next generation of modeling systems and environments.

Generate Reference Documentation. FW/SM's *REFGEN*, *ID_DICT*, *EDGEN*, and *NETDRAW* processes produce an unusually wide variety of reference documents, as mentioned in §2.4. Bear in mind that these documents relate only to the model itself, and are distinct from the kinds of reports produced by solvers (e.g., optimizers produce reports pertaining to an optimal solution).

FW/SM's reference documents are useful for many purposes, including helping modelers see aspects of a model from different viewpoints, debugging model specifications, quickly answering common kinds of queries concerning general structure or details, facilitating communication, and mitigating the ubiquitous problem of inadequate documentation for both modeling and nonmodeling professionals.

Most mature modeling systems can produce some sort of reference documentation. By comparison, FW/SM has two main advantages. The first derives from its having been built on top of an integrated package like Framework. Users do not require much computer expertise in order to be able to do useful and sometimes quite sophisticated manipulation of the documents, which are all machine-resident as word frames or database frames (except for *NETDRAW*'s display). Users don't need to know an operating system and have a stable of utilities and specialized editors, as they probably would with a Unix-based modeling environment like ANALYTICOL (Childs and Meacham 1985). They just need to know Framework, which was designed for users with limited computer expertise.

The significance of this advantage is not that it encourages future modeling environments to be built using a particular integrated package, but rather that it has enabled us to confirm through personal experience the importance, in the realm of reference documentation, of the kind of software integration called for in the quote given in §3.1.

The second advantage of FW/SM is that it is based on a conceptual modeling framework that gives a central role to capturing definitional dependencies when specifying the general structure of a model. Thus it is possible for FW/SM to produce documents that cannot be produced without this kind of information, namely (at present): the Calling Sequence report, the Topological Sort, the Adjacency and Reachability Matrices, and (a version of) the Genus Graph. We remark that definitional dependency information is valuable not only to enable certain kinds of reference documentation, but also for such uses as integrated modeling (see §3.2 of ⟨MS 87⟩) and model maintenance and evolution, the efficiency of which depends importantly on knowing all implications of any given change.

Export Tables and Invoke Menu-Driven Query Interface (*XtrieveQuery*). The *XtrieveQuery* process demonstrates the possibility that similar automatic interfaces to relational DBMSs may be achievable for any modeling environment that separates its specification of general structure from its specification of instantiating data, and that organizes its data in relations with predictable structure. The desirability of such interfaces was argued in ⟨MS 87⟩.

A DBMS interface gives access to a query interface and query processor for efficiently answering a wide variety of questions with respect to a model's data, optimal variable values (if an optimizer has been invoked), and other calculated values. The lack of such query capabilities in most extant modeling systems constitutes one of the greatest anachronisms of contemporary modeling practice.

Perform Optimization (*MPS_Interface* and *GENNETFW*). The *MPS_Interface* and *GENNETFW* processes suggest that optimization-oriented modeling systems like those for the AMPL (Fourer, Gay, and Kernighan 1990), GAMS (Brooke, Kendrick, and Meeraus 1988), and LINGO (Cunningham and Schrage 1991) languages may have the potential to evolve into modeling environments that support much more of the modeling life cycle than just the optimization phase. This would bring about a qualitative improvement in their usefulness.

FW/SM's two optimization interfaces are not designed for large-scale problem solving, but—as pointed out in §2.5—they do enable experimentation with two different design positions about how much the user is required to know. It is not clear at this time which style of interface is best in which circumstance. There are advantages to requiring the user to understand both the model and the desired solver well enough to be able to fill out a simple control table linking the two. On the other hand, there are advantages to minimizing the work required of the user.

Whichever interface style is adopted, no computer programming skill should be required of the user, and it should be easy to switch optimizers if more than one is available that applies to the model at hand. These are properties that surely will be found in all successful modeling environments of tomorrow.

Export Basic Schema Structure and Invoke Arity Prolog (*PrologQuery*). Support for the model formulation process is one of the weak links of today's modeling systems. This process requires a great deal of ad hoc reasoning about partial or interim model structures, and much additional ad hoc reasoning takes place with respect to the general structure of models as they are instantiated, communicated, solved, and used in various other ways throughout the modeling life cycle. Thus there is a clear need for the kind of logical inference capabilities that *PrologQuery* makes available for ad hoc reasoning about the most fundamental aspects of general model structure. Such capabilities are unusual among contemporary modeling systems, except perhaps for those based on logic modeling (e.g., Bhargava 1990, Chari and Krishnan 1990, Krishnan 1991).

FW/SM's simultaneous support for logical inference together with solvers for retrieval and optimization presages the advent of modeling environments that offer inferencing capability in addition to the more usual solution capabilities around which current modeling systems are built.

4. Conclusion

The main task of this section is to appraise the degree to which FW/SM fulfills the vision presented in Geoffrion (1989c) for a new generation of modeling environments. That article, it should be noted, only mentions structured modeling in passing.

The article argues that a modeling environment needs five properties in order to achieve the primary goals of higher quality, higher productivity, and greater popularity for model-based work. It should:

(1) nurture the entire modeling life cycle, not just part of it
(2) be hospitable to decision- and policy-makers, not just to modeling professionals
(3) facilitate the maintenance and ongoing evolution of the models and systems built within it
(4) enable all of its inhabitants to "speak" the same paradigm-neutral language for model definition
(5) facilitate good management of key resources, namely data, models, solvers, and results derived from these.

The article goes on to detail the design implications of these properties, and thence to infer three major design challenges:

(A) the need for a general conceptual framework for modeling
(B) the need for an executable model description language based on this framework
(C) effective software integration covering (i) linkable libraries of data sources, models, solvers, and derived results, (ii) programs that achieve the necessary kinds of "executability," and (iii) the tools and utilities needed for total life cycle support of modeling work.

We discuss each of these challenges.

4.1. *Challenge A*: *Conceptual Framework*

⟨MS 87⟩ and Geoffrion (1989b) put forth structured modeling as an answer to Challenge A. This framework is demonstrably general in applicability, rigorous, paradigm-neutral and yet compatible with most modeling paradigms in MS/OR, and suitable for use as a foundation for meeting Challenge B. See, for example, the discussion of §3.2 of the present article. These are four of the six requirements that Geoffrion (1989c) lists for a framework to meet Challenge A. A fifth is that it should be "understandable and natural for the main players at each stage of the modeling life cycle." It is too soon in the history of structured modeling to reach a definitive judgement on this requirement, but the appearance of FW/SM and a comprehensive SML tutorial (Geoffrion 1990b) set the stage for a proper evaluation.

The final requirement is that the conceptual modeling framework should be compatible with "good" modeling style. We believe that structured modeling meets this admittedly subjective requirement, and arguments to this effect have been made elsewhere. Some of the reasons: it separates general structure from instantiating data, separates models from problems from solvers, exploits repetitive structure (that is the main task of genera), organizes models hierarchically in modules, and documents interdependencies.

FW/SM's role with respect to Challenge A is to offer the opportunity for hands-on experimentation with one possible realization of the structured modeling framework, thereby enabling individuals to judge for themselves this framework's potential with respect to the six requirements.

4.2. *Challenge B*: *Executable Modeling Language*

Geoffrion (1990a) puts forth SML as an answer to Challenge B and studies its theoretical adequacy. We mentioned two of the results of that study in §3.5 of the present article.

Geoffrion (1989c) points out that five of the six requirements relating to Challenge A also apply to any executable modeling language aspiring to support the chosen modeling framework. Because SML was designed specifically to support the structured modeling framework, it was possible to design SML so that it satisfies these requirements to about the same degree as the structured modeling framework satisfies them in the context of Challenge A.

FW/SM proves that SML is "executable," but does it provide the necessary functionality? Geoffrion (1989c) lists four desirable kinds of functionality as error-trapping, automatic documentation, solver interface setup, and a smart loader/editor for detailed data. We discuss each in turn.

Error-Trapping. The theory behind the *SMLCheck* process shows that syntactic and semantic error-trapping can be complete in an important sense at the level of model classes, as explained in §3.5. A similar statement holds relative to error-trapping at the level of detailed data, although FW/SM is as yet quite incomplete in this regard. Vicuña (1990) shows that complete error-trapping is possible at both levels.

Automatic Documentation. Thanks to *REFGEN* and a few other processes, FW/SM can automatically produce a variety of useful kinds of model documentation.

Solver Interface Setup. FW/SM's ability to automatically set up easy, no-programming interfaces to external solvers for optimization, query, and logical inference is demonstrated by the *GENNETFW*, *MPS_Interface*, *XtrieveQuery*, and *PrologQuery* processes.

Smart Loader/Editor. A smart loader/editor for detailed data should be set up automatically, be customized to the general model structure at hand, and should offer: (a) data structures able to hold elemental detail table data, together with a passive interface for entry and editing, and (b) an active interface for data entry and editing. The active interface should be able to: (b1) automatically create genus element populations that are fully defined in terms of other genus element populations (e.g., enter a Cartesian product where the general structure calls for one), (b2) diagnose what operations are needed to achieve a state of internal data consistency, and (b3) carry out, or at least assist in performing, the needed operations. FW/SM offers (a) via *EDGEN*. It does not yet offer (b). The technology of *FcEval* can be adapted to achieve (b1). Function (b2) is possible because SML aims to make all definitional dependencies explicit via the generic calling sequence, and because the Table Content Rules of Geoffrion (1990a) constitute a comprehensive characterization of internal data consistency (as explained in §3.5 of this article). Function (b3) can build on functions (b1) and (b2). Detailed design is under way.

In summary, FW/SM's main role with respect to Challenge B is to demonstrate that most (perhaps all) of the necessary executability can be achieved, and to enable SML's worth to be judged on the basis of hands-on experience.

4.3. *Challenge C: Software Integration*

FW/SM illustrates one possible approach to achieving the level of software integration called for by Challenge C. Namely, add new modeling capabilities to a popular software package that already provides a highly integrated set of services needed by any modeling environment. The first three sections of this article show that FW/SM adequately meets Challenge C at the level of a research prototype.

The companion paper on FW/SM's implementation (Neustadter et al. 1992) discusses FW/SM's approach to software integration from a programming viewpoint. More generally, it would appear that the technical problems posed by Challenge C are rapidly being mitigated by intense commercial activity in the software development community aimed at database connectivity, graphical user interfaces, interoperability, multitasking, networking, and more advanced application development tools and operating systems.

4.4. *Closing Comments*

It should go without saying that the modeling environment ideals presented in Geoffrion (1989c) are not immutable, but rather serve as a point of departure. We agree strongly with the viewpoint expressed in Barstow et al. (1984) in the context of programming (not modeling) environments. The following quote from the editors' closing chapter is equally applicable in the present context:

> The process of developing programming environments is highly experimental. *We have no hope of abstractly defining the "ideal" programming environment, which can then be implemented. Our best strategy, in fact our only viable one, is to continue to develop environments, experiment with them to identify those aspects which are either helpful or useless, and to incorporate these lessons into the next experiment.*

We have approached FW/SM in this spirit. We have used it to confirm theoretical results (e.g., various results about the internal consistency of SML), to probe conjectures (e.g., concerning the generality of SML and the usefulness of hierarchical structure for managing complexity), to test the feasibility of design options (e.g., three major options for designing a resident evaluator for function and test genera), and to demonstrate structured modeling ideas to outsiders in quest of their feedback. We have also used it to experimentally guide the evolution of our ideas about modeling environments. In fact, SML itself and the vision presented in Geoffrion (1989c) were influenced to a considerable degree by experience with an evolving FW/SM.

Our plans for FW/SM's future include extending its capabilities, using it to study the learnability and effectiveness of SML for various classes of users, and studying the applicability of FW/SM and the ideas it incorporates to practical modeling work and to the evolutionary improvement of other modeling systems. Our experience with FW/SM, together with the experience of others who are presently exploring alternative design concepts and implementation technologies, will guide our future efforts relative to FW/SM and new research prototypes.

The first successor prototype entered development in July of 1991. It is being programmed in C++ and OSF/Motif under UNIX on a Sun SPARCstation, relies heavily on a commercial database system, adopts the object-oriented paradigm where applicable, and places great emphasis on extensibility—especially with respect to the incorporation of new solvers.

In closing, it should be said that evolving research prototypes of this complexity are very difficult to sustain in a university setting. The demands for professional skills, financial resources, and sheer patience are severe. Nevertheless, it is this writer's firm belief that research on modeling environments requires experimental developments of this sort if it is to achieve its full potential for success.[1]

[1] Partially supported by the National Science Foundation, the Office of Naval Research, Ketron Management Science, Inc., and Shell Development Company. The views contained in this report are those of the author and not of the sponsors.

I am pleased to acknowledge the excellent design and implementation work done by Sergio Maturana, Laurel Neustadter, Yao-Chuan Tsai, and Fernando Vicuña. Their comments on a draft of this paper also were invaluable, as were those of Richard Ramirez and Gordon Wright.

References

AHO, A. V., R. SETHI AND J. ULLMAN, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA, 1986.

ARITY, "Arity Prolog Version 5.0," Arity Corporation, 30 Domino Drive, Concord, MA 01742, 1987.

ASHTON-TATE, *Framework III*, Ashton-Tate, 20101 Hamilton Ave., Torrance, CA 90502, 1988.

BARSTOW, D. R., H. E. SHROBE AND E. SANDEWALL (EDS.), *Interactive Programming Environments*, McGraw-Hill, New York, 1984.

BHARGAVA, H., "A Logic Model for Model Management: An Embedded Languages Approach," Ph.D. Thesis, The Wharton School, University of Pennsylvania, 1990, 163 pp.

BROOKE, A., D. KENDRICK AND A. MEERAUS, *GAMS: A User's Guide*, The Scientific Press, Redwood City, CA, 1988.

BROWN, G. G. AND R. D. MCBRIDE, "Solving Generalized Networks," *Management Sci.*, 30, 12 (December 1984), 1497–1523.

CAREY, M. J. AND D. J. DEWITT, "An Overview of the EXODUS Project," *Database Engineering*, 10, 2 (June 1987), 47–54.

CHARI, S., "Knowledge Representation Using Structured Modeling," Ph.D. Thesis, Anderson Graduate School of Management, UCLA, 1988, 225 pp.

——— AND R. KRISHNAN, "Towards a Logical Reconstruction of Structured Modeling," Working Paper 7-89, Decision Systems Research Institute, SUPA, Carnegie-Mellon University, October 1990, 42 pp.; *Decision Support Systems*, (to appear).

CHILDS, C. AND C. R. MEACHAM, "ANALYTICOL—An Analytical Computing Environment," *AT&T Tech. J.*, 64, 9 (November 1985), 1995–2007.

CLEMENCE, JR., R. D., "LEXICON: A Structured Modeling System for Optimization," Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1984.

CUNNINGHAM, K. AND L. SCHRAGE, *LINGO Optimization Modeling Language*, University of Chicago, Chicago, 1991, 110 pp.

DATE, C. J., *An Introduction to Database Systems.* Vol. 1, (Third Ed.), Addison-Wesley, Reading, MA, 1981.

DESAI, S., "Extensible Database Systems as a Platform for Modeling," Ph.D. Thesis in progress, Anderson Graduate School of Management, UCLA, 1992.

ELLISON, E. F. D. AND G. MITRA, "UIMP: User Interface for Mathematical Programming," *ACM Trans. Math. Software*, 8, 3 (September 1982), 229–255.

FARN, C. K., "An Integrated Information System Architecture Based on Structured Modeling," Ph.D. Thesis, Graduate School of Management, UCLA, 1985.

FOURER, R., D. M. GAY AND B. W. KERNIGHAN, "A Mathematical Programming Language," *Management Sci.*, 36, 5 (May 1990), 519–554.

GEOFFRION, A. M. (MS 87), "An Introduction to Structured Modeling," *Management Sci.*, 33, 5 (May 1987), 547–588.

———, "Reusing Structured Models via Model Integration," *Proc. Twenty-Second Annual Hawaii Internat. Conf. System Sciences*, held in Kailua-Kona, January 1–3, IEEE Computer Society Press, Washington, 1989a, 601–611. To be reprinted in R. Blanning and D. King (Eds.), *Current Issues in Decision Support Systems*, IEEE Computer Society Press.

———, "The Formal Aspects of Structured Modeling," *Oper. Res.*, 37, 1 (January-February 1989b), 30–51.

———, "Computer-Based Modeling Environments," *European J. Oper. Res.*, 41, 1 (July 1989c), 33–43.

———, "SML: A Model Definition Language for Structured Modeling," Working Paper 360, Western Management Science Institute, UCLA, revised August 1990a, 129 pp.

———, "The SML Language for Structured Modeling," Working Paper 378, Western Management Science Institute, UCLA, August 1990b, 160 pp. Two-part extract *Oper. Res.*, (to appear).

———, "A Library of Structured Models," Informal Note, Anderson Graduate School of Management, UCLA, revised August 1990c, 274 pp.

———, "Indexing in Modeling Languages for Mathematical Programming," Working Paper 371, Western Management Science Institute, UCLA, revised July 1991a, 180 pp. Extract to appear in *Management Sci.* Portions appeared as "A Taxonomy of Indexing Structures for Mathematical Programming Modeling Languages," in *Proc. Twenty-Third Annual Hawaii Internat. Conf. System Sciences*, held in Kailua-Kona, January 2–5, 1990; Vol. III, IEEE Computer Society Press, Washington, 463–473.

———, "FW/SM: A Prototype Structured Modeling Environment," Working Paper 377, Western Management Science Institute, UCLA, revised July 1991b, 36 pp.

———, S. MATURANA, L. NEUSTADTER, Y. TSAI AND F. VICUÑA, "User Documentation for FW/SM Release X90-09," Anderson Graduate School of Management, UCLA, December 1990a, 100 pp.

———, ———, ———, ——— AND ———, "Technical Documentation for FW/SM," Anderson Graduate School of Management, UCLA, April 1990b, 102 pp.

HAMMER, M. AND D. MCLEOD, "Database Description with SDM: A Semantic Database Model," *ACM Trans. Database Systems*, 6, 3 (September 1981), 351–386.

HILL, D. S., "A Prototype for Converting Linear Programming Models to Structured Modeling Graphs," Master's Thesis in Information Systems, Naval Postgraduate School, March 1989, 137 pp.

HULL, R. AND R. KING, "Semantic Database Modeling: Survey, Applications, and Research Issues," *ACM Comput. Surveys*, 19, 3 (September 1987), 201–260.

HÜRLIMANN, T. AND J. KOHLAS, "LPL: A Structured Language for Linear Programming Modeling," *OR Spektrum*, 10 (1988), 55–63.

JONES, C. V., "Attributed Graphs, Graph-Grammars, and Structured Modeling," *Ann. Oper. Res.*, (1991) (to appear).

KENDRICK, D. A., A. MEERAUS AND J. ALATORRE, *The Planning of Investment Programs in the Steel Industry*, The Johns Hopkins University Press, Baltimore, MD, 1983.

KRISHNAN, R., "PDM: A Knowledge-Based Tool for Model Construction," *Decision Support Systems*, (1991) (to appear).

LEBLOND, G. T. AND D. F. COBB, *Using* 1-2-3, (Second Ed.), Que Corporation, Indianapolis, 1985.

LENARD, M., "An Implementation of Structured Modeling Using a Relational Database Management System," paper presented at Second Annual Conf. Integrated Modeling Systems, University of Texas, Austin, October 1987, 20 pp.

LUCAS, C., G. MITRA AND K. DARBY-DOWMAN, "Modeling of Mathematical Programs: An Analysis of Strategy and an Outline Description of a Computer Assisted System," Report TR/09/83, Department of Mathematics and Statistics, Brunel University, UK, October 1983.

MICRO DATA BASE SYSTEMS, *KnowledgeMan*, Micro Data Base Systems, Inc., P.O. Box 248, Lafayette, IN 47902, 1985.

MS 87. See Geoffrion (1987).

NEUSTADTER, L., "On the Structure of Data in SML Models," Research Paper, Anderson Graduate School of Management, UCLA, March 1990, 62 pp.

———, A. GEOFFRION, S. MATURANA, Y. TSAI AND F. VICUÑA, "The Design and Implementation of a Prototype Structured Modeling Environment," *Ann. Oper. Res.*, Model Management, B. Shetty (Ed.), (1992) (Forthcoming).

NOVELL, *Xtrieve*: *Interactive Query Manual*, Novell Development Products Division, 6034 W. Courtyard, Suite 220, Austin, TX, 1988.

O'DELL, D. D., "The Design and Implementation of a Visual User Interface for a Structured Model Management System," Master's Thesis in Information Systems, Naval Postgraduate School, March 1988, 170 pp.

PARK, J. Y., "A Computer-Aided Tool for Decision Support Model Building Based on Structured Modeling," Master of Engineering Thesis, Korea Advanced Institute of Science and Technology, December 1987, 57 pp. plus appendices.

RAMIREZ, R., "The ASUMMS Project: An Overview," Dept. of Decision and Information Systems, Arizona State University, January 1990, 10 pp.

———, "Architecture and Implementation of the DAMS System," ISUMMS Technical Report No. 4, Iowa State University, July 1991, 44 pp.

REPS, T. W. AND T. TEITELBAUM, *The Synthesizer Generator*, Springer-Verlag, New York, 1989.

SCHRAGE, L., *LINDO*: *An Optimization Modeling System*, (4th Ed.), Scientific Press, Redwood City, CA, 1991.

SHNEIDERMAN, B., *Designing the User Interface*, Addison-Wesley, Reading, MA, 1987.

VICUÑA, F., "Semantic Formalization in Mathematical Modeling Languages," Ph.D. Thesis, Computer Science Department, UCLA, June 1990.

WOROBETZ, N. D. AND G. P. WRIGHT, "Data Model Representation and Implementation in OR/SM, a Model Management System," draft working paper, Krannert Graduate School of Management, Purdue University, 1991, 47 pp.

WYANT, M. A., "Design and Implementation of a Prototype Graphical User Interface for a Model Management System," Master's Thesis in Information Systems, Naval Postgraduate School, March 1988, 98 pp.