The SML Language for Structured Modeling: Levels 1 and 2

Arthur M. Geoffrion

Operations Research, Vol. 40, No. 1 (Jan. - Feb., 1992), 38-57.

Stable URL:
http://links.jstor.org/sici?sici=0030-364X%28199201%2F02%2940%3A1%3C38%3ATSLFSM%3E2.0.CO%3B2-R

Operations Research is currently published by INFORMS.

THE SML LANGUAGE FOR STRUCTURED MODELING: LEVELS 1 AND 2

ARTHUR M. GEOFFRION

University of California, Los Angeles, California

(Received November 1990; revision received April 1991; accepted May 1991)

This is the first of two articles on the principal features of SML, a language for expressing structured models. SML is presented in terms of four "levels" of increasing expressive power; this article covers the first two levels, while the sequel covers levels 3 and 4. The lower levels, at least, are easy to learn. Both articles rely entirely on examples and give special attention to the characteristics of SML that, collectively, make it unique. The intended audience includes evaluators of other modeling languages, designers of modeling languages and systems, and those who follow the development of structured modeling.

A new generation of modeling environments is envisioned in Geoffrion (1989c). That vision requires three main design challenges be met: 1) a general conceptual framework for thinking about models, 2) an executable language based on this framework for representing models, and 3) software integration on a grand scale.

The foundations of structured modeling (Geoffrion 1989a) were developed as one possible answer to the first challenge. SML—for Structured Modeling Language—was developed to answer the second challenge. SML has been implemented in a research prototype modeling environment called FW/SM, which also develops an approach to the third challenge (Geoffrion 1991 and Neustadter et al. 1991). See Geoffrion (1987) for a general introduction to structured modeling.

SML was first introduced informally in Geoffrion (1987). Subsequently, SML was specified formally in Geoffrion (1990a), a long report intended for reference rather than for publication. A much more accessible account of SML is the comprehensive tutorial in Geoffrion (1990b), which enables others to understand SML's construction well enough to evaluate its strengths and weaknesses, and to read and write it with confidence. The present pair of articles is essentially a selective condensation of that tutorial.

The aims of this article and its sequel are: a) to present informally the main features of SML via a series of simple models, and b) to explain the characteristics of SML that we believe are particularly noteworthy. Our intended audience includes evaluators of modeling languages, designers of modeling languages and systems, and those who follow the development of structured modeling.

To understand SML and its distinctive characteristics, one first needs to understand the conceptual framework given in Geoffrion (1989a). That framework views every analytic model as a collection of objects wherein each object has a definition that is either primitive or based on the definition of other objects. It is natural to diagram these definitional dependencies as a directed graph whose nodes are the objects. Briefly, the conceptual framework categorizes the nodes into five basic types, groups them into classes by similarity, organizes the node classes into a hierarchy (rooted tree) as a means for managing complexity, and associates mathematical expressions with the node classes whose nodes have computable values. This sketch is sufficient to bring into focus the essence of how structured modeling views every analytical model: as a kind of computationally active graph of the model's constituent objects and the definitional dependencies among them.

Subject classifications: Computers/computer science: modeling language design. Computer/computer science, data bases: semantic data modeling. Information systems, decision support systems: structured modeling.
Area of review: COMPUTING.

0030-364X/92/4001-0038 $01.25

SML is a notational system for this way of viewing analytic models. However, it would be wrong to leave the impression that SML is the only language designed for this purpose. Another such, still in its formative phase, is the logic-based language LSM (Chari and Krishnan 1990). A graph-based language is also available (Jones 1991). Many alternatives are possible, and in fact a number of dialects of SML can be found in other prototype structured modeling implementations cited in Geoffrion (1991).

There are, of course, many modeling languages and associated systems already in existence. To cite just a few, there are:

- AMPL and GAMS in a field of more than 25 languages for optimization modeling (Fourer, Gay and Kernighan 1990, Brooke, Kendrick and Meeraus 1988);
- CAMP for economic planning (Sagie 1986);
- DATAFORM for LP matrix generation (DATAFORM 1987);
- DEMOS for risk analysis and Monte Carlo simulation (Wishbow and Henrion 1987);
- GNGEN for network modeling (Forster 1988);
- IFPS' text-based language for spreadsheet modeling (e.g., Plane 1986);
- LINDO for linear, integer, and quadratic programming model instances (Schrage 1991);
- 1-2-3™ for spreadsheet modeling;
- OPTIMA and PM* for production-distribution-inventory modeling (Karrenbauer and Graves 1989, Krishnan 1990);
- SIMSCRIPT for discrete event simulation modeling (e.g., Markowitz 1979);
- STELLA for continuous simulation modeling (Stella 1989).

However, none of these languages, nor any other with which this writer is acquainted, seems able to meet the challenges set forth in Geoffrion (1989c).

Notable Characteristics of SML

Whatever SML's virtues and shortcomings may be relative to alternative modeling languages, the following characteristics are among those which readers may find of particular interest in the context of modeling language design:

- meticulous attention to the explicit specification of definitional dependencies, which provides semantic information for model debugging, communication, maintenance, integration, evolution, and the automatic production of various kinds of reference documentation that otherwise would not be feasible;
- a sublanguage for making documentary comments that has nearly all the permissiveness of natural language, yet enough rules of composition (concerning the use of underlining and upper case) to enable checks to be made automatically on semantic consistency;
- insightful graphs as a by-product even for models that seem to have nothing to do with graphs;
- support for hierarchical organization of major model components as a means of managing complexity in large models;
- extensive logical capabilities that enable SML to express, in a natural way, logical model features reminiscent of the propositional and predicate calculi, either alone or in combination with other kinds of model features;
- model/problem independence, a property that promotes conceptual clarity and posing multiple problems and tasks relative to a single model;
- the ability to strongly separate the general structure defining a class of models from the data needed to single out a particular model instance, a quality that should facilitate reuse, communication, quality control, and other important activities;
- rules for deriving, from general model structure, a relational data structure design for detailed data that is robust with respect to insertion/deletion/update anomalies (Neustadter 1990), a quality that reduces the burdens of model design, promotes standardization, and sets the stage for exploiting relational data base technology;
- elaborate support for sparsity, a capability that is important not only for large models, but also, in general, if model specification semantics are to be faithful to reality;
- a rigorous foundation based on the formal, explicit semantic framework given in Geoffrion (1989a);
- comprehensive specification of semantics as well as syntax for both general model structure and model instance data, a property that facilitates comprehensive error-trapping;
- the ability to represent a very wide range of models.

This article and its sequel discuss each of these characteristics, plus a few others, in this order.

Four Levels

SML is a single language, but we present it in terms of four levels of increasing expressive power. This approach provides an easy learning path for

newcomers to SML, and offers a useful way to think about SML (and perhaps even modeling in general) after SML has been learned. The lower levels, at least, are easy to learn. The most difficult features are postponed to Level 4, which supports sparse indexing structures.

The scope of SML increases at each level, but even Level 1 has significant application. Expressive power can be sketched as follows.

Level 1 encompasses:

- Definitional Systems (correlated, acyclic, hierarchical, two definition types);
- Structural Models/Graphs (labeled).

Level 2 encompasses:

- Definitional Systems (three additional definition types);
- Structural Models/Graphs (add values, computation);
- Spreadsheets;
- Numeric Formulas;
- Propositional Calculus Models.

Level 3 encompasses simply indexed versions of the above plus:

- Mathematical Programming (simple indexing only);
- Predicate Calculus Models.

Level 4 encompasses richly indexed versions of the above plus:

- Relational Data Base Models;
- Semantic Data Base Models.

The four levels are perfectly upward compatible, that is, any model description at one level is valid at any higher level.

We shall have much more to say about the classes of models that can be expressed at each level, but it may be useful at this point to follow one of the strands through all four levels. We choose graph models (not to be confused with the definitional dependency graph mentioned earlier):

- Level 1 SML can be viewed as a text-based language for representing any graph. Nodes and arcs may be named, and may be organized hierarchically.
- Level 2 allows nodes and arcs to have values that are either specified by the user or calculated by a formula.
- Level 3 allows "similar" nodes and arcs to be indexed and organized in simple (nonsparse) classes, and it introduces data tables for organizing node and arc information according to these classes.
- Level 4 allows the use of index-based subsetting for defining complex (sparse) classes of nodes and arcs.

Although implementations of SML are not a focus of this pair of articles, it should be mentioned that SML's division into four levels is likely to induce a useful four-way classification of the capabilities of any implementation. This works out nicely in the case of FW/SM: Each of FW/SM's capabilities can be classified according to the lowest level at which it is useful (of course, it remains useful at all higher levels). Thus, FW/SM's features can be revealed progressively to users level-by-level in parallel with SML's features. Not only is this sequence natural, but it is effective as well because each feature then can be learned and exercised first in the context of models that are no more complicated than necessary to render the feature meaningful. The Appendix contains the details.

Organization and Typography

The organization of the balance of this article and its sequel is simple. Sections 1 and 2 of each paper give separate treatment of the four levels of SML. Each of these sections has two standard subsections that treat one level of SML as an extension of the previous level:

1. an example-based discussion of the additional kinds of models that can be represented, presented so as not only to illuminate the issue of expressive power, but also to explain some of the principal language features introduced at the current level;
2. a discussion of the broader significance of selected SML characteristics that first become apparent at the current level.

Geoffrion (1990b) contains two additional subsections for each level:

3. a detailed explanation of the new language features;
4. some exercises for the reader (the solutions are given in an appendix).

However, we do not attempt here even a condensation of that material, owing to space limitations. Serious students of SML will need to study that paper, which is freely available from the author.

Section 3 of the companion paper is devoted to significant SML characteristics not treated elsewhere.

In this article and its sequel, we actually employ the minor dialect of SML that is used by the FW/SM prototype (Geoffrion 1991), rather than the formal version described in Geoffrion (1990a). This dialect differs from the formal version only in that it uses indentation, boldface, and underlining to enhance readability by enabling the omission of certain

reserved words needed only for purely technical reasons.

ITALIC CAPITALS are used for the names of all examples.

Familiarity with Sections 1–3 of Geoffrion (1987) and Sections 1–3 of Geoffrion (1989a) is advisable for full comprehension of what follows.

1. LEVEL 1 SML: STRUCTURAL MODELING

Level 1 SML is a tiny subset of the full language that is very easy to learn. It suffices for what others have called structural modeling, which is not to be confused with structured modeling in its general form. One obtains Level 1 SML by imposing two restrictions on full SML: There must be exactly one element per genus, and there may be no value-bearing elements.

Geoffrion (1989a) presents some 28 definitions covering the core concepts and the associated concepts and constructs of structured modeling. The ones that are pertinent to Level 1 SML are listed in Table I.

The reader is encouraged to review these ideas in Geoffrion (1989a), including the detailed illustrations given there for the classical transportation model. We shall illustrate their meaning in the context of another model in Section 1.1.1. Note the following comments:

1. **Definition 6:** The two conventions given in Geoffrion (1989a) for the proper design of a calling sequence are important. Briefly, they say that a calling sequence should: a) call all relevant elements (indirectly if not directly), and b) call no irrelevant elements. Within these limits, there is still latitude to exercise personal taste as to whether an indirect call should be promoted to direct status, or a direct call should be demoted to indirect status (not always possible).
2. **Definition 10:** Generic structure is trivial at Level 1 because, as mentioned at the outset, there is just one element per genus.
3. **Definition 13:** As pointed out at the end of Section 4 of Geoffrion (1989a), the practical meaning of a monotone ordered modular structure is

Table I
Definitions Used in Level 1

Definition	Core Concepts	Informal Explanation
1	*Primitive entity element*	A primitive definition representing any distinctly identifiable entity; the first of five element types
2	*Compound entity element*	A definition based on the definitions of certain primitive entity elements or other compound entity elements; the second of five element types
6	*Calling sequence*	A segmented tuple containing all the elements on which a nonprimitive element's definition depends
7	*Closed collection of elements*	Calling sequences only call other elements in the collection
8	*Acyclic collection of elements*	No definition in the collection is ultimately circular with respect to calls
9	*Elemental structure*	Nonempty, finite, closed, acyclic collection of elements
10	*Generic structure, genus*	An elemental structure partition that does not mix element types; each cell called a *genus*
12	*Modular structure, module, default modular structure*	A rooted tree whose leaf nodes are 1:1 with the genera of a generic structure; each nonleaf node called a *module*; degenerate tree with only one module (the root) called the *default modular structure*
13	*Monotone ordering*	An order for the modular structure tree such that preorder traversal yields all genera in a no-forward-reference order
14	*Structured model*	An elemental structure together with an associated generic structure and monotone ordered modular structure

Definition	Associated Constructs	Informal Explanation
23	*Modular outline*	The indented list representation of the preorder traversal of a (not necessarily monotone) ordered modular structure
24	*Element graph*	A directed acyclic graph representing all elements of an elemental structure and the calling sequence references among them; has certain node and arc attributes
25	*Genus graph*	A directed acyclic graph representing all genera of a generic structure and the calling sequence references among them
27	*Adjacency matrix*	A node-node adjacency matrix for an element graph or a genus graph (shows direct calls only)
28	*Reachability matrix*	A node-node reachability matrix for an element graph or a genus graph (shows indirect as well as direct calls)

simpler than its formal definition might suggest. It simply means that the genera should be arranged sensibly in outline form so that there are *no forward references* among them.

4. **Definitions 24 and 25:** The one-element-per-genus restriction implies that the element graph and genus graph are one and the same for Level 1 (and Level 2) SML, provided one ignores the node and arc attributes of element graphs.
5. We have listed just five of the associated concepts and constructs of Geoffrion (1989a), but in fact all nine are pertinent to SML at Level 1. The other four are unimportant to present purposes.

The rest of this section indicates some of the kinds of models that Level 1 SML can represent, and discusses the significance of selected characteristics of Level 1 SML.

1.1. Level 1 Models

Structural models have been employed for years by a community of people who rely on simple graph models as a way to express structural relationships over an extremely wide range of applications. See, for example, Harary, Norman and Cartwright (1965), and Roberts (1976), which are written from the graph theoretic viewpoint, and Warfield (1976), which addresses the management of qualitative ideas and complexity. Lendaris (1980) provides a good survey. Besides the varied applications which structural modeling finds in references such as these, structural modeling is also of interest to MS/OR as an approach to the very earliest phases of the model formulation process.

Any graph can be represented in Level 1 SML by viewing its nodes and arcs as "objects" to be modeled. If this approach unduly subordinates whatever the graph was originally intended to represent, other Level 1 representations usually are possible that are free of this shortcoming. In any case, Level 1 SML suffices for structural modeling.

Level 1 SML also suffices to represent the simplest kinds of definitional systems. This subsection presents two simple examples: a definitional system and a cyclic graph.

1.1.1. Definitional System: *DOS GLOSSARY*

Definitional systems were discussed in detail in Section 1.1 of Geoffrion (1989a), which advocated that they should be correlated, acyclic, hierarchical, typed, and grouped. Level 1 supports the first three properties and two of the five possible types for the fourth, namely primitive and compound entities. (The fifth property awaits Level 3.)

Figure 1 gives a small system of definitions covering rudimentary concepts relating to files and directories in MS-DOS (a more detailed model is sketched in the sequel paper's discussion of Level 3). Notice that:

- the Schema comprises named genus and module paragraphs, which are organized according to an indented outline; module paragraphs are easy to distinguish because their names always begin with "`&`";
- each genus paragraph hosts one definition of the definitional system;
- each paragraph has a formal part in boldface and an informal part not in boldface;
- the formal part of each genus paragraph begins with its name, includes definitional dependencies (if any) between parentheses, and concludes with a declaration of element type—"`/pe/`" for primitive entity and "`/ce/`" for compound entity;
- the informal part is ordinary English, but underlining is used to declare key phrases and capitals are used to indicate previously declared key phrases and variants thereof.

The glossary shown in Figure 1 consists of ten definitions: FILE, F_NAME, . . ., D_TREE. FILE is a primitive in the sense that it is not defined in terms of anything else in the definitional system. Hence, in the structured modeling framework, FILE is modeled as a "primitive entity" (`/pe/`) element. Note that the remaining elements are defined in terms of previous elements. For example, DIR is defined in terms of FILE, and so FILE is in DIR's "calling sequence" (DIR "calls" file) and DIR is modeled as a "compound entity" (`/ce/`) element.

Figure 2 gives the associated genus graph.

We now use Figure 1 to illustrate the core concepts reviewed at the outset of Section 1: the definition of a file is a primitive entity element (the only one); all other definitions are compound entity elements; the calling sequence (list of definitional dependencies) of each compound entity element appears between parentheses; the collection of all ten elements obviously is closed and acyclic (this is especially evident from the genus graph); the elemental structure is everything in Figure 1 except for the &FILES and &DIRECTORIES paragraphs; the generic structure is essentially the same as the elemental structure given that one chooses the finest possible partition wherein every element is alone in its genus (the genus names are FILE, . . ., D_TREE); the modular structure is a

&FILES FILE definitions

FILE /pe/ A FILE is a collection of related information. All programs, text, and data on disk reside in FILES.

F_NAME (FILE) /ce/ Every FILE has a FILE NAME that follows DOS conventions.

F_EXT (FILE) /ce/ Every FILE has a FILE EXTENSION that follows DOS conventions.

F_ID (F_NAME, F_EXT) /ce/ The FILE ID of a FILE refers to its FILE NAME followed by its FILE EXTENSION, separated by a period.

F_SIZ (FILE) /ce/ Every FILE has a FILE SIZE measured in bytes.

F_DATE (FILE) /ce/ Every FILE has a FILE DATE indicating the date and time of creation or last modification.

&DIRECTORIES DIRECTORY definitions

DIR (FILE) /ce/ A disk DIRECTORY contains an entry for each FILE on a disk. Each DIRECTORY may contain other DIRECTORIES, which are referred to as "subdirectories".

D_NAME (DIR) /ce/ Every DIRECTORY has a DIRECTORY NAME that follows DOS conventions.

D_SIZ (DIR, F_SIZ) /ce/ The sum of the FILE SIZES of all FILES in a DIRECTORY (but not in any subdirectories) is called the DIRECTORY SIZE.

D_TREE (D_NAME) /ce/ The DIRECTORY TREE is a rooted labeled tree whose links represent which DIRECTORIES are contained in which other DIRECTORIES; the nodes (with the exception of the root) are labeled by and one-to-one with the DIRECTORY NAMES.

Figure 1. Level 1 Schema for *DOS GLOSSARY*.

rooted tree with exactly two modules, named &FILES and &DIRECTORIES, whose genus children are evident in Figure 1; the default modular structure would simply be the root with the ten genera as children; the natural monotone ordering associated with Figure 1—clearly it has the no-forward-reference property—is the one which puts genera in top-to-bottom order within each module; and finally, based on the above, Figure 1 corresponds to a structured model. This completes the illustration of all core concepts.

Further clarification is in order for the earlier comment on Definition 6. Consider F_ID in Figure 1. Notice that F_ID depends definitionally on FILE because there are indirect calls to FILE via F_NAME and F_EXT. At the modeler's discretion, F_ID could be made to call FILE directly without altering the meaning of the Schema. However, F_ID must not call F_SIZ, because the F_SIZ concept has no role in defining the F_ID concept; and F_ID must call F_EXT directly because that concept plays a definitional role and there is no way to call it indirectly.

Consider now the associated constructs. Figure 1 shows a particular modular outline if one ignores everything except paragraph names, and Figure 2 is the genus graph, which could be made into the element graph if it were annotated with the necessary node and arc attributes. The adjacency and reachability matrices can be derived easily from either Figure 1 or 2, but we shall not exhibit them.

Finally, we offer additional details concerning the informal "interpretation" part of each paragraph, which is written in a sublanguage of SML intended for documentary purposes. This sublanguage is one of the few things that SML adds to structured modeling as set forth in Geoffrion (1989a). Its only rules have to do with the use of capitalized strings called "key phrases" (more precisely, a key phrase is an upper case letter followed by upper case letters, digits, apostrophes, underscores, slashes, hyphens, or blanks). A key phrase is either underlined, in which case it is uniquely associated with the module or genus being defined in its paragraph, or it is not underlined, in

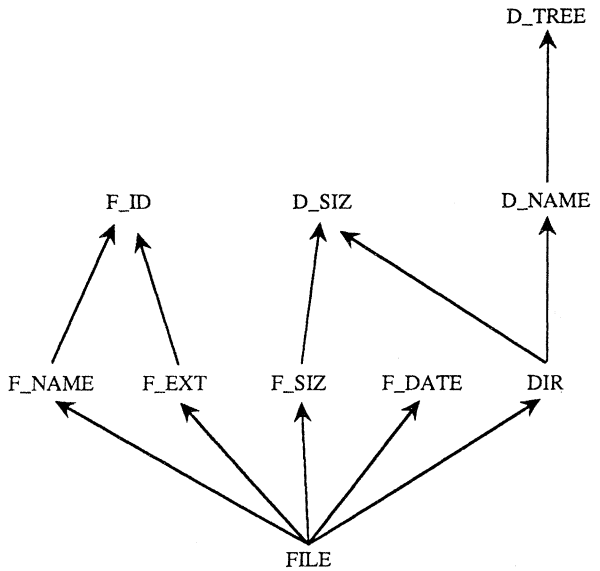


Figure 2. Genus Graph for *DOS GLOSSARY*.

which case it is a reference to some underlined key phrase. (Exactly which key phrase is being referenced is nearly always obvious to human readers, but a computer is likely to require user interaction for accurate recognition owing to the varied grammatical inflections of English.) An underlined key phrase serves to introduce vocabulary for discourse about the concept of its paragraph, and need only obey the rule of uniqueness: Underlined key phrases must be unique within a given schema. Every other key phrase occurs in the course of discourse about the current paragraph or prior ones, and must obey certain rules (see Geoffrion 1990a or b) designed to make sure that calling sequences are complete with respect to definitional dependencies. It follows that, as a result of the structure of SML's interpretation sublanguage, automatic checks can be made on the completeness of calling sequences and on the consistency of the modular structure and of interpretations.

The interpretations of Figure 1 illustrate the proper use of key phrases. Clearly, they are unique and all have correct syntax, with "FILE" associated with the FILE paragraph, "FILE EXTENSION" associated with F_EXT, and so on. Aside from the FILE paragraph itself, every occurrence of "FILE" or "FILES" in an interpretation denotes a reference to the FILE paragraph, which therefore requires the referencing paragraph to call FILE directly or indirectly or, if the reference is made by a module paragraph (as in the case of &FILES), then it is required that the module includes FILE or some genus that calls FILE. Indeed, this rule is obeyed. Note that the plural form is treated exactly like the singular form. In general, all grammatical inflections arising from changes in number, person, tense, etc., are treated exactly like the base key phrase. Note also that the term "DOS" appears several times and obeys the key phrase syntax, but is neither a key phrase nor an inflection of any key phrase. This, together with the problem of recognizing inflections, illustrates why user interaction with a key phrase recognizer is necessary for any foreseeable implementation of SML's interpretation sublanguage.

1.1.2. A Structural Model: *TOURNAMENT*

A directed graph $[V, A]$ is called a *tournament* if, for all $u \neq v$ in V, (u, v) is in A or (v, u) is in A, but not both (Roberts 1976). Such graphs can describe the outcome of round robin athletic competitions, pairwise comparison experiments, and other situations.

Figure 3 shows a particular tournament graph. This graph has node names, but no arc names. Note that there are two directed cycles, a fact which is not at odds with the acyclicity requirement of structured modeling. An SML rendering of this graph appears in Figure 4. The correspondence between Figures 3 and 4 should be evident. In particular, note that the modular structure has the no-forward-reference property.

It is also possible to represent the class of all tournament graphs in SML, but that cannot be done in a useful way within the confines of Level 1.

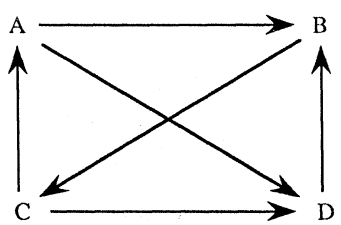


Figure 3. *TOURNAMENT* Graph.

```
&NODES
   A      /pe/
   B      /pe/
   C      /pe/
   D      /pe/

&ARCS
   AB     (A, B)     /ce/
   AD     (A, D)     /ce/
   BC     (B, C)     /ce/
   CA     (C, A)     /ce/
   CD     (C, D)     /ce/
   DB     (D, B)     /ce/
```

Figure 4. Level 1 SML Schema for *TOURNAMENT* (see Figure 3).

1.1.3. Representing Graphs in General

Formally speaking, Level 1 SML can represent any structural model that can be represented by a graph, either directed or undirected, possibly with names for some nodes and/or arcs. Moreover, Level 1 SML allows the nodes and arcs to be organized in a hierarchy (tree) as a tool for managing complexity in large models.

One style of representation is as follows. Most graphs can be expressed in ways other than the one to be described; we do not necessarily advocate the particular modeling style used here, which serves only to exhibit one possibility that always works.

Consider any directed or undirected graph with possibly named nodes and arcs. Create one `/pe/` genus paragraph for every node, and one `/ce/` genus paragraph for every arc with the calls in the calling sequence in tail-to-head order if the arc is directed. Use node and arc names, if any, for genus names; otherwise, genus names are arbitrary. Interpretations are advisable, but may be omitted. Modular structure is arbitrary so long as the no-forward-reference requirement is obeyed. For example, one could use the default modular structure by creating no modules other than the root and putting the `/pe/` paragraphs first in any order, followed by the `/ce/` paragraphs in any order. Clearly, Figure 4 employs this modeling style.

1.1.4. Comments on the Generality of Definitional Systems

The prior examples and discussion, while perhaps compelling in the case of graphs, do not do justice to the generality of the definitional system concept even within the confines of Level 1. Definitional systems are much broader in applicability than might be guessed from a simple glossary application like the one in Section 1.1.1. The following comments elaborate this point.

1. The Schema of Figure 1 is a slightly modified extract of a 166 paragraph Level 1 Schema that completely models the user interface of a popular file and directory manager program.
2. Level 1 definitional systems are useful as a tool for structuring ideas in connection with both reading and writing:
   a. An appreciation for the applications to reading can be gained by taking any text passage(s) of interest and constructing therefrom a definitional or conceptual system in Level 1 SML. The basic idea is to build a model using primitive and compound entity genera to capture the various concepts in what is being read, and the relations among these concepts. This is similar to parsing sentences in English or statements in a computer language, but the result is a structural model instead of a parse tree.
   b. An appreciation for the applications to writing can be gained by a similar exercise. See Wayner (1988, 1990) for related ideas.
3. Although the definitional dependency notion played a key role in the development of the structured modeling formalism and of SML, it is possible to substitute other notions that are also compatible with the structured modeling formalism and SML. Specifically, replacing definitional dependency by any other acyclic relation on a model's elements yields a possible alternative interpretation of the structured modeling formalism, and, hence, a possible new application domain. Appendix A of Geoffrion (1990b) discusses this point further.

1.2. Notable SML Characteristics Apparent at Level 1

At least four notable characteristics of SML are apparent.

Explicit Definitional Dependencies

Geoffrion (1989a) explains on page 32 why *correlation* is a desirable property for definitional systems. The given reasons extend in an obvious way to modeling languages, and are compelling. We next recapitulate one of those reasons.

Making changes in a model raises the possibility of introducing inconsistencies. Most inconsistencies arise in practice because the ramifications—"interdependencies," really—of what was changed were not taken fully into account. The likelihood of such inconsistencies rises rapidly with model complexity. Therefore, it is desirable for the interdependencies among the various components of a model to be visible and explicit in the model and its documentation. Then, when a model is changed, it will be possible to work out the full ramifications of the change, whether direct or indirect.

Additionally, making definitional dependencies explicit is a prerequisite for the two SML characteristics discussed next and for certain SML language features to be mentioned in later sections, and it enables several kinds of reference documentation to be generated automatically (Geoffrion 1991). It also sets the stage for a hypertext (e.g., Conklin 1987)

implementation of a Schema or of model documentation based on a Schema.

SML makes definitional dependencies explicit via calling sequences. Appendix A of Geoffrion (1990b) expands this topic.

Semi-Structured Sublanguage for Documentary Comments

Inadequate documentation has been identified repeatedly as a major factor contributing to failures of modeling projects and the premature demise of modeling systems. It impedes understanding, communication, maintainability, and evolution. See, for example, Gass (1984), and Section 10.9 of Miser and Quade (1985).

Well-written SML produces thorough and readable documentation as a by-product of its notational conventions. We refer particularly to the interpretation part of genus and module paragraphs which, if used properly, render the formal part of a model written in SML essentially self-documenting. Indeed, being self-documenting was one of the explicit design goals of SML (Geoffrion 1990a). In addition, SML's design makes it possible to automatically produce various kinds of supplementary documentation, as the REFGEN process of FW/SM demonstrates (Geoffrion 1991). See also Section 3.3 of Geoffrion (1987).

A self-documenting modeling language not only has the benefits suggested above, but also can lead to better models. To develop one aspect of this point, consider the sublanguage used for genus and module interpretations. As indicated at the end of Section 1.1.1, interpretations are natural language with certain syntax and rules of use imposed for key phrases. These rules make it possible to carry out certain completeness and consistency tests with reference to the formal part of a paragraph. This has the important effect of reducing the chances that the formal part of a Schema will be semantically unrealistic.

In other words, these rules promote a degree of rigorous coordination between the formal and informal parts of a model description, in contrast to the usual situation with a modeling language in which comments are totally unstructured and can contain blatant inconsistencies with respect to the formal model specification.

Note that it is the central role played by definitional dependency which enables the formal and informal parts of an SML Schema to be tied together through rules of use for key phrases, and, thus, which enables the completeness and consistency checks mentioned above.

For a radically different approach to the documentation problem, see Bhargava and Kimbrough (1990). It has been applied to structured modeling by Chari and Krishnan (1990).

Insightful Graphs Always Available

Graphs can facilitate visualizing important aspects of a model. Advocates of modeling paradigms that have natural graphs, such as decision analysis and flow networks, argue strongly that these greatly enhance understanding for all who are involved.

Element graphs, genus graphs, module graphs, and modular structure trees are standard constructs of structured modeling that illuminate model structure. These graphs are available for *every* model written in SML whether or not the model happens to have a natural graph associated with it. All are illustrated in Geoffrion (1989a) for the ordinary transportation model (as expected, none bears any obvious resemblance to the usual network diagram for this class of models). See Geoffrion (1987) for other examples.

Few other modeling languages offer any graphs or diagrams as standard complements to their primary representation.

Hierarchical Organization for Managing Complexity

Hierarchical organization is widely practiced and recognized in many fields as an effective way to deal with complexity, which, in turn, is one of the major banes of understanding and communication. Almost any modeling language can benefit from hierarchical organization applied to the conceptual units of a model and its documentation. SML's organization of its paragraphs according to a modular outline is its main tool for achieving hierarchical organization.

Closely related to the notion of hierarchical organization are the notions of modularization and top-down design, widely esteemed in the software engineering community and elsewhere as effective approaches to the design of complex structures. Modules can be much easier to understand and work with than the whole which comprises them, and a top-down approach based on a hierarchical view of complexity can be an effective way to design them. Thus, it would seem appropriate for a complex model to be designed not as a monolith, but rather as an interconnected collection of modules that is organized hierarchically.

Modularization and top-down design are straightforward to accomplish in SML. See Section 3 of

Geoffrion (1987), and Geoffrion (1989b) for additional discussion and examples.

2. LEVEL 2 SML: STRUCTURAL MODELING WITH VALUE-BEARING ELEMENTS

Level 2 adds a major ingredient to the Level 1 foundation: new kinds of elements that have an associated real, integer, logical, or string value. One can look at Level 2 SML either as an extension of structural modeling that permits elements to have values, or as a no-indexing restriction of full structured modeling (indexing is not needed, as there is still only one element per genus at Level 2).

The introduction to Section 1 listed the core concepts and associated constructs of structured modeling that were pertinent to Level 1 SML. Those plus the ones in Table II are pertinent to Level 2.

The reader is invited to review the material in Geoffrion (1989a) on these ideas, and to work out their meaning in the context of the models of Section 2.1. We do this in Section 2.1.2 for the model *SATELLITE*.

The rest of this section indicates some of the kinds of models which Level 2 SML can represent, and discusses the significance of selected characteristics of Level 2 SML.

2.1. Level 2 Models

Section 1.1 discussed the kinds of models representable in Level 1 SML from two main perspectives, definitional systems and graphs. Level 2 offers extensions of each of these, and can also represent spreadsheet models, formula-oriented models, and propositional calculus models. These five possibilities do not exhaust the useful ways of thinking about the scope of Level 2 SML, but they do convey a reasonable appreciation of its expressive power. We discuss each in turn.

2.1.1. Definitional Systems

Level 1 SML has two types of definitions, namely primitive and compound entity. Level 2 supports three more types: attribute, function, and test. All of the Level 2 examples that follow can be viewed from the definitional systems perspective.

2.1.2. Graph Models With Data or Formula Attributes: *SATELLITE, RUSSIAN ROULETTE*

This subsection considers graphs whose nodes and arcs may have attributes in the form of given or calculated values. Level 2 SML is able, at a formal level, to represent any directed or undirected graph with:

- possibly named nodes and/or arcs;
- given values of numeric, logical, or string type at selected nodes and/or arcs;
- numeric or logical values at selected nodes and/or arcs that are calculated from values at other nodes and/or arcs, so long as none of the calculations is circular, and the expressions describing the calculations fall within a certain (broad) class.

Like Level 1, Level 2 SML also allows nodes and arcs to be organized in a hierarchy for purposes of complexity management in large models.

This subsection develops two examples, *SATELLITE* and *RUSSIAN ROULETTE*, and concludes by considering the general case.

SATELLITE

An earth satellite's orbit can be disturbed by gravitational forces directed toward other objects that it encounters. It is therefore of interest to be able to

Table II
Definitions Used in Level 2

Definition	Core Concepts	Informal Explanation
3	*Attribute element, value, range*	A definition with a user-supplied value in a certain range, based on the definitions of certain primitive or compound entity elements; the third of five element types
4	*Function element, value, rule*	A definition with a value calculated by a certain rule, based on the definitions of certain other elements; the fourth of five element types
5	*Test element, value, rule*	A definition with a logical value calculated by a certain rule, based on the definitions of certain other elements; the fifth of five element types
15	*Complete specification*	A structured model with every detail fully specified
16	*Variable attribute*	An attribute element whose value the modeler expects to change often or to place under solver control
17	*Evaluation*	The task of determining the value of every function and test element

calculate, based on the mass and proximity of such objects, whether or not they pose a threat to the satellite's orbit. Newton's Law of Gravitation makes this calculation an easy one: The force which an object exerts on a satellite is proportional to the product of the masses of the satellite and the object, and inversely proportional to the square of the distance between them. The proportionality factor is the universal gravitational constant, which is about 6.67×10^{-11} in metric system units. We assume that an object is a "threat" if it exerts enough force to accelerate one gram by one millimeter per second^2; that is, if it exerts a force of at least 10^{-6} newton. For a numerical instance of this model, assume further that the satellite has a mass of 100 kg, and that a 2,000 kg object is 200 meters away.

Figure 5 diagrams the calculation. It comprises seven nodes, all with names written adjacent to them, and unnamed arcs. Inside the oval representing each node is written either a user-supplied value or a formula for calculating the value. Except for the arcs leaving the two nodes at the bottom, arcs represent the transfer of a value from one node to another. The arcs leaving the two bottom nodes serve to indicate the meaning of the three middle nodes. Clearly, Figure 5 falls within the class of attributed graphs identified earlier.

Figure 5 can be modeled as in Figure 6. Notice that:

- all type declarations absent from Level 1 SML now appear ("/a/" for attribute and "/va/" for variable attribute, "/f/" for function, and "/t/" for test);
- ": Real+" declares nonnegative attribute values (in general, a colon always announces a so-called range statement for an attribute genus—that is, a constraint on the allowable value of each element);
- the formula for gravitational force is introduced by a semicolon and written in an obvious notation that uses "^" for exponentiation (in general, a semicolon always announces a so-called generic rule for function and test genera—that is, a numeric-valued or logical-valued expression);
- the logical-valued expression that determines the value of THREAT is written as though it were a constraint, but it is not; the expression evaluates to true or false and does not enforce any condition whatever.

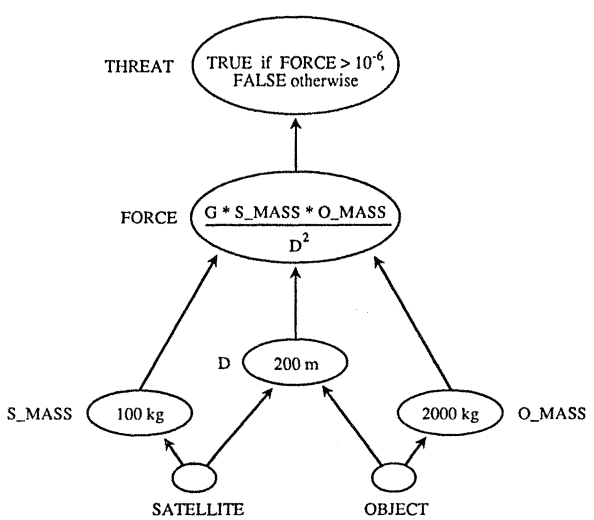


Figure 5. Diagram of *SATELLITE* Model.

All the terms reviewed at the outset of Section 2 can be illustrated with reference to this example: mass and distance are attribute elements, force is a function element, "threat" is a test element, complete specification would require specifying the two mass values and the distance value, object mass and distance are treated as variable attribute elements because they are likely to change, and evaluation involves the calculation of values for force and threat.

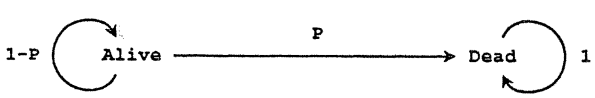
A completely specified and evaluated model instance is given by Figure 6 together with this Elemental Detail Table (maximally joined and named S_MASS):

||	S_MASS	O_MASS	D	FORCE	THREAT
||	100	2000	200	3.335E-10	#FALSE

Note that SML uses "#FALSE" and "#TRUE" for truth values.

RUSSIAN ROULETTE

The trivial Markov chain associated with playing Russian roulette has this graph:



Note that, whereas Figure 5 has data and formula attributes on the nodes, this graph has them on the arcs. A Schema that represents this graph appears in Figure 7.

It is possible to simplify this Schema by merging the paragraphs of the last module with the corresponding paragraphs of the next to last module, but we shall not do this.

If the game is played with a six-chamber revolver,

SATELLITE /pe/ There is a SATELLITE in space.

OBJECT /pe/ There is an OBJECT in space.

S_MASS (SATELLITE) /a/ : Real+ The SATELLITE has a certain SATELLITE MASS in kg.

O_MASS (OBJECT) /va/ : Real+ The OBJECT has a certain OBJECT MASS in kg.

D (SATELLITE, OBJECT) /va/ : Real+ The SATELLITE and OBJECT are a certain DISTANCE apart in meters.

FORCE (S_MASS, O_MASS, D) /f/ ; 6.67 * 10^(-11) * S_MASS * O_MASS / D^2 The OBJECT exerts a certain FORCE on the SATELLITE, in newtons, according to Newton's Law of Gravitation.

THREAT (FORCE) /t/ ; FORCE > 10^(-6) The OBJECT is a THREAT to the SATELLITE if and only if it exerts a FORCE of greater than one millionth of a newton.

Figure 6. Level 2 Schema for *SATELLITE.*

&STATES possible STATES of a Russian roulette player

ALIVE /pe/ ALIVE

DEAD /pe/ DEAD

&TRANSITIONS STATE TRANSITIONS in Russian roulette

LOSE (ALIVE, DEAD) /ce/ At a single play of Russian roulette, an ALIVE player who LOSES TRANSITS from ALIVE to DEAD.

WIN (ALIVE, ALIVE) /ce/ At a single play of Russian roulette, an ALIVE player who WINS stays ALIVE.

DEATH_IS_FOREVER (DEAD, DEAD) /ce/ At a single (virtual) play of Russian roulette, a DEAD player stays DEAD.

&TRANSITION_PROB TRANSITION PROBABILITIES in Russian roulette

P (LOSE) /a/ : 0 <= Real <= 1 There is a certain PROBABILITY OF LOSING in a single play.

WIN_PROB (WIN, P) /f/ ; 1 - P The PROBABILITY OF WINNING in a single play is the complement of the PROBABILITY OF LOSING.

DEATH_IS_FOREVER_PROB (DEATH_IS_FOREVER) /a/ : 1 <= Real <= 1 The PROBABILITY OF STAYING DEAD is one.

Figure 7. Level 2 SML Schema for *RUSSIAN ROULETTE.*

then the associated Elemental Detail Tables would be:

P $\frac{||\ P}{||\ 1/6}$

WIN_PROB $\frac{||\ WIN\_PROB}{||\ \ 5/6}$

DEATH_IS_FOREVER_PROB $\frac{||\ DEATH\_IS\_FOREVER\_PROB}{||\ \ 1.0}$

Level 2 SML can represent, at a formal level, a certain broad class of graphs with attribute values that are given or calculated. The modeling approach given below substantiates this claim.

Consider any directed or undirected graph falling within the class given at the outset of this subsection. Do the following:

A. Model the topological structure and names exactly as explained in Section 1.1.3.

B. For each node or arc having a user-supplied value, create one /a/ genus paragraph for each such value; make it call the corresponding /pe/ or /ce/ genus paragraph and make its range statement declare the appropriate value type.
C. For each node or arc having a calculated value, create one /f/ of /t/ genus paragraph for each such value according to whether the calculated value is numeric or logical; write the generic rule so that it will perform the appropriate calculation, and make calls to the corresponding /pe/ or /ce/ genus paragraph and to all genera used in the generic rule.
D. Optionally, it may be possible to simplify by merging some /pe/ and /ce/ paragraphs in an obvious way with the /a/, /f/ and /t/ paragraphs that call them.
E. Interpretations are optional.
F. Modular structure is arbitrary so long as the no-forward-reference requirement is obeyed. For example, one could use the default modular structure by creating no modules other than the root and putting the /pe/ genus paragraphs first, in any order, followed by the /ce/ genus paragraphs in any order, followed by the /a/ genus paragraphs in any order, followed by the /f/ and /t/ paragraphs in no-forward-reference order (this must be possible by the noncircularity assumption).

RUSSIAN ROULETTE is consistent with this style, although unmade Step D simplifications are possible.

Steps A–F work at a formal level, but in particular cases the result may not incorporate all of the essential definitional dependencies that a modeler may wish to include. The reason is that, like the approach of Section 1.1.3 (of which this is an extension), this approach does not take full account of the key role which definitional dependencies are supposed to play in structured modeling. To comply with the spirit of structured modeling, it is necessary to add a new step:

G. Include additional calling sequence calls to make explicit any essential definitional dependencies.

This addition could lead to circular definitional dependencies, which, of course, would mean that the Schema is no longer true SML. However, suitable representations can nearly always be achieved by using an alternative style for rendering such situations in Level 2 SML.

One such alternative style would be to begin with Step C without any /pe/ or /ce/ calls, and then to add /pe/ and /ce/ genera as necessary to bring out all essential definitional dependencies. This style usually works well when all values are associated with nodes (and none with arcs). The *SATELLITE* Schema is consistent with this style.

2.1.3. Spreadsheets: *INCOME STATEMENT*

Graphs of the type described in Section 2.1.2 encompass spreadsheet models without circular references. One need only view each nonempty spreadsheet cell as a node with its row-and-column address as its name. There would be one arc for each external reference of each cell.

Since noncircular spreadsheets are known to suffice for a remarkably wide variety of modeling applications, the same must be true of Level 2 SML. Of course, it does not follow that all such spreadsheet applications would be "natural" to reformulate in Level 2 SML; Level 3 usually is much more satisfactory for spreadsheet models in which, as is the usual case, groups of similar cells occur.

INCOME STATEMENT is based on the financial model appearing in the last chapter of LeBlond and Cobb (1985). It illustrates how Level 2 SML can represent a typical spreadsheet. Figure 8 shows the income statement as it appears in the book. Figure 9 gives the corresponding Schema, where interpretations have been omitted as self-evident.

Not given here are the one-row, one-column Elemental Detail Tables (one for each /a/ and /f/ genus) necessary to represent the particular numerical instance of Figure 8. If desired, it should be an easy matter, in all but the most rudimentary modeling systems based on structured modeling, to construct a report that is linked to the contents of these tables, and mimics the layout of Figure 8.

Gross Sales	$732,730
Less: Returns and Allowances	4,167
Net Sales	728,563
Cost of Goods Sold	468,947
Gross Margin	259,616
Operating Expenses	201,042
Depreciation	12,016
Earnings Before Interest and Taxes	46,558
Interest Expense	7,043
Earnings Before Taxes	39,515
Income Taxes	10,342
Earnings After Taxes	29,173
Cash Dividends	0
Net Income	$29,173

Figure 8. Income Statement from LeBlond and Cobb (1985).

```
FIRM /pe/

&INCOME_STATEMENT

   &GROSS_MARGIN

      GROSS_SALES (FIRM) /a/

      RETURNS_AND_ALLOWANCES (FIRM) /a/

      NET_SALES (GROSS_SALES, RETURNS_AND_ALLOWANCES) /f/ ;
      GROSS_SALES - RETURNS_AND_ALLOWANCES

      COST_OF_GOODS_SOLD (FIRM) /a/

      GROSS_MARGIN (NET_SALES, COST_OF_GOODS_SOLD) /f/ ;
      NET_SALES - COST_OF_GOODS_SOLD

   &EARNINGS_BEFORE_TAXES

      OPERATING_EXPENSES (FIRM) /a/

      DEPRECIATION (FIRM) /a/

      EARNINGS_BEFORE_I&T (GROSS_MARGIN, OPERATING_EXPENSES,
      DEPRECIATION) /f/ ; GROSS_MARGIN - (OPERATING_EXPENSES
      + DEPRECIATION)

      INTEREST_EXPENSE (FIRM) /a/

      EARNINGS_BEFORE_TAXES (EARNINGS_BEFORE_I&T,
      INTEREST_EXPENSE) /f/ ; EARNINGS_BEFORE_I&T -
      INTEREST_EXPENSE

INCOME_TAXES (EARNINGS_BEFORE_TAXES) /f/ ; .26 *
EARNINGS_BEFORE_TAXES

EARNINGS_AFTER_TAXES (EARNINGS_BEFORE_TAXES, INCOME_TAXES)
/f/ ; EARNINGS_BEFORE_TAXES - INCOME_TAXES

CASH_DIVIDENDS (FIRM) /a/

NET_INCOME (EARNINGS_AFTER_TAXES, CASH_DIVIDENDS) /f/ ;
EARNINGS_AFTER_TAXES - CASH_DIVIDENDS
```

Figure 9. Level 2 SML Schema for *INCOME STATEMENT*.

The range of all attributes is the reals; this is SML's default in the absence of a range statement, which is optional.

2.1.4. Numeric Formulas: *BEAM DEFLECTION*

Level 2 SML furnishes a language for writing many kinds of numeric formulas. Constants needed by formulas can be represented by numeric-valued attribute elements, variables by numeric-valued variable attribute elements, and the numeric formulas themselves by function elements. Formulas are executed by evaluation in the structured modeling sense.

The simple *BEAM DEFLECTION* model from structural mechanics is illustrative. It predicts the deflection of a beam that is supported at both ends and is subject to a point load in the middle. Let **L** be the length of the beam, **I** be its cross-sectional moment of inertia, **E** be its modulus of elasticity, and **P** be the midpoint load. Then the deflection is given by the formula $PL^3/48\ EI$. Figure 10 gives an SML Schema for this formula.

An Elemental Detail Table (maximally joined and named BEAM_NAME) with sample data for a 2-inch diameter aluminum rod, after evaluation, is:

|| BEAM_NAME	L	I	E	P	DEF
|| MyBeam	36	0.049	10^7	100	0.198

```
BEAM /pe/
```
There is a simple elastic BEAM supported at both ends.

```
BEAM_NAME (BEAM) /a/ : String
```
The BEAM has a NAME.

```
L (BEAM) /a/ : Real+
```
The BEAM has a known LENGTH in inches.

```
I (BEAM) /a/ : Real+
```
The BEAM has a known cross-sectional MOMENT OF INERTIA in inches to the fourth power.

```
E (BEAM) /a/ : Real+
```
The BEAM has a known MODULUS OF ELASTICITY in pounds per square inch. This has to do with the stiffness of the material from which the BEAM is made.

```
P (BEAM) /va/ : Real+
```
The BEAM bears a known LOAD, in pounds, placed exactly at its midpoint.

```
DEF (L, I, E, P) /f/ ; P * L^3 / (48.0 * E * I)
```
The BEAM exhibits a midpoint DEFLECTION, in inches, that depends on LENGTH, MOMENT OF INERTIA, MODULUS OF ELASTICITY, and LOAD.

Figure 10. Level 2 SML Schema for *BEAM DEFLECTION*.

2.1.5. Propositional Calculus Modeling: *CONTRACT*

Level 2 SML furnishes a language for writing any propositional calculus model with a finite number of sentences. See Appendix F of Geoffrion (1990b) for a detailed discussion of this claim. Atomic formulas can be represented by attribute or variable attribute genera of logical type. Sentences in propositional logic, which are obtained by combining atomic formulas via such logical connectives as *and*, *or*, *not*, *implies*, and *equivalent*, can be represented by test genera. Structured modeling's evaluation operation determines the truth or falsity of any such sentence when all atomic formulas are given specific truth values.

To illustrate, consider a company in the midst of collective bargaining. There is a current draft of the contract, and four optional clauses named **W**, **X**, **Y**, and **Z** are on the table. There is a certain federal regulation that holds if and only if **W** and either **X** or **Y** are adopted. And there is a certain union demand that will be met if and only if **Y** and **Z** are not both adopted.

An SML Schema describing this situation appears in Figure 11. Note that **@AND**, **@OR**, and **@NOT**, respectively, represent conjunction, disjunction, and negation.

An Elemental Detail Table (maximally joined and named **W**) with sample data, after evaluation is:

||	W	X	Y	Z	T:FED_REG	T:UNION_DEM
||	#TRUE	#FALSE	#FALSE	#TRUE	#FALSE	#TRUE

```
CONTRACT /pe/
```
There is a union CONTRACT.

```
W (CONTRACT) /va/ : Logical
```
CLAUSE W is optional.

```
X (CONTRACT) /va/ : Logical
```
CLAUSE X is optional.

```
Y (CONTRACT) /va/ : Logical
```
CLAUSE Y is optional.

```
Z (CONTRACT) /va/ : Logical
```
CLAUSE Z is optional.

```
T:FED_REG (W, X, Y) /t/ ; @AND (W, @OR (X, Y))
```
FEDERAL REGULATION TEST determines whether a certain Federal regulation holds: it does if and only if CLAUSE W and either of CLAUSES X and Y are in the CONTRACT.

```
T:UNION_DEM (Y, Z) /t/ ; @NOT (@AND (Y, Z))
```
UNION DEMAND TEST determines whether a certain union demand is met: it does if and only if CLAUSES Y and Z are not both in the CONTRACT.

Figure 11. Level 2 SML Schema for *CONTRACT*.

2.2. Notable SML Characteristics Apparent at Level 2

We discuss two notable characteristics of SML that first become apparent at Level 2.

True Logical Capability

The ability to express any propositional calculus model with a finite number of sentences is more than a curiosity. It is a useful modeling tool that adds to a modeling language's generality not only for the specialized purposes of propositional calculus, but also for many other purposes when blended with language features that support numeric computation. In fact, we shall see that Level 3 SML can do finite predicate calculus.

Not all modeling languages have true logical-valued variables and the ability to represent general logical-valued expressions. Most can simulate at least some logical model features with the help of numeric-valued quantities, sometimes supplemented with logical operators (which might, for example, treat any nonzero number as "true"), but we believe that this is an undesirable substitute for the real thing.

The basis for this belief is the obvious desirability of representing a model in as natural a way as possible. In particular, modeling tricks—like numerical representations of inherently logical model features—should be avoided whenever possible because they usually: a) inflict semantic distortions (e.g., numeric quantities are never truly logical), b) undermine the value of error-checking routines since the underlying error-checking rules are ignorant of the use of modeling tricks, and c) reduce the ease with which models can be communicated, understood, and used correctly, especially when such activities involve, as they so often do, people who are not modeling professionals.

Separation of Models From Problem Statements and Solvers

Elsewhere we have made a sharp distinction between models, problems, and solvers (Section 2.3 of Geoffrion 1987). Basically, we take: a) a *model* to be a representation of some aspects of reality, b) a *problem* or *task* to be a description of something to be done with a model, and c) a *solver* to be a manipulator of a model according to some definite procedure for solving a problem or performing a task.

These are useful distinctions to make for a number of reasons. First, these distinctions encourage the same model to be used with different solvers (perhaps to solve different problems or to carry out different tasks), and it encourages the same solver to be used with different models. Such reuse saves time and resources.

A second reason is the pursuit of conceptual clarity. Nonspecialists can easily become confused otherwise. For example, every consultant has had a client ask "Can you handle such-and-such a feature?" The true answer is often Yes and No: "Yes" in that the feature could be included in the model, but "No" in that an otherwise excellent solver would become inapplicable. This answer will not be understood unless the client knows the conceptual difference between a model and a solver, and knows that a great variety of problems and tasks lies between the two. For another example, a user who is presented with an "LP model" may fail to understand that its data base has stand-alone value for ad hoc retrieval, that the objective function and a constraint can switch roles in order to drive on a different criterion, or that the model can be used for static simulation on a casewise basis. It would be better for the user to be presented with a "model" on which many problems and tasks may be posed, some of which may include optimization performed with the help of standard solvers.

A more subtle reason for distinguishing between model and solver is that not doing so typically leads to predicating model design on a particular solver's limitations. This inhibits keeping track of modeling decisions that might warrant reversal if and when a more capable solver becomes available.

Clearly, SML makes it very difficult to confuse a model with a problem, task, or solver because SML provides only for representing models. Not all modeling languages have this characteristic. For example, most modeling languages for mathematical programming make it mandatory to combine the statement of a model with problem particulars (i.e., objective function, variables, and constraints must be specified as such).

Of course, there is no reason why a structured modeling environment could not allow problems and tasks to be stated in a natural way using SML as a point of departure. We give ten problem/task examples expressed in natural language, except that key phrases (in capitals) have been used where possible. In each case for which solver software is to be invoked, both the model and an appropriate problem/task must be communicated to the solver in a way that it can understand.

SATELLITE

1. "What is the mass of the SATELLITE?" This requires retrieval of the value of S_MASS.

2. "For the model instance given, what value of DISTANCE would cause the FORCE to be exactly 10^{-6} newton?" This requires solving one equation in one variable, a kind of problem sometimes called "goal-seeking." It can also be viewed as a special kind of "satisfaction," which we take in general to be the problem of finding variable values that satisfy a simultaneous system.
3. "For the model instance given, what happens to FORCE if OBJECT MASS increases to 5000 kg?" This requires changing a variable attribute value and re-evaluating.
4. "In general, what is the first derivative of FORCE as a function of DISTANCE?" This requires symbol manipulation.
5. "Given a joint probability distribution on OBJECT MASS and DISTANCE of closest approach, what fraction of encountered OBJECTS will pose a THREAT?" This question can be answered approximately by a Monte Carlo simulation experiment that runs the *SATELLITE* model numerous times, once for each draw from the given distribution.

RUSSIAN ROULETTE

6. "Given a ⅙ PROBABILITY OF LOSING, what is the expected number of plays for which a player will remain ALIVE?" This requires probabilistic inference.

CONTRACT

7. "Find a subset of CLAUSES W, X, Y, and Z to include in the CONTRACT, such that the FEDERAL REGULATION TEST and UNION DEMAND TEST are both satisfied." This is a satisfaction task in logic.
8. "Is there a subset of CLAUSES W, X, Y, and Z to include in the CONTRACT, such that the FEDERAL REGULATION TEST and UNION DEMAND TEST are both satisfied?" This is an existence problem in logic. (An existence problem is like a satisfaction problem, but does not require displaying a solution if one exists.)
9. "If there is a set of truth values for W, X, Y, and Z, such that T:FED_REG and T:UNION_DEM are both true, is the solution necessarily unique?" This is a uniqueness problem in logic.
10. "Given a set of truth values for W, X, Y, and Z, such that T:FED_REG and T:UNION_DEM are both true, is X necessarily true?" This requires logical inference.

These examples illustrate ten kinds of problem/task: retrieval, goal-seeking, what if (i.e., re-evaluation), symbol manipulation based on the differential calculus, Monte Carlo simulation, probabilistic inference, satisfaction, proof of existence, proof of uniqueness, and logical inference. This list is far from exhaustive. For instance, there is also optimization, which motivates several of the examples to be given in the companion article.

Developing good SML-based model manipulation languages for expressing these and other kinds of problems and tasks presents a worthy challenge in its own right. Such a language is an essential part of what is often called a *solver interface*, of which we shall see some examples at Level 3. However, where evaluation is concerned, we believe that the evaluation mechanism should be so tightly integrated with an SML implementation that it almost appears to be part of SML itself.

APPENDIX

FW/SM Processes Classified by Lowest Possible Level of Use

The functionality of the FW/SM research prototype has been described fully elsewhere (Geoffrion 1991). We shall not repeat the details here, but we do wish to point out that certain of FW/SM's *processes* are useful even at the lower levels of SML. In fact, the four levels of SML induce a natural stratification of FW/SM's processes.

We describe first the processes that are useful in connection with SML Level 1 and higher, then those which are useful for Level 2 and higher, and finally we describe the processes which are useful for Level 3 and higher. There are no processes that are useful only at Level 4.

Each process provides one functionality or coherent group of functionalities, and is invoked by selecting it from a tree-structured menu of processes.

A.1. FW/SM Processes Useful for Level 1 and Higher

FORMAT Format the Schema

This process assures proper indentation when a Schema is printed, and also does some preparatory work for *SMLCheck*.

SMLCHECK Check Schema Syntax and Schema Properties

SML specification includes not only a formal context-free grammar describing SML's lexical and syntactic structure, but also an exhaustive collection of so-called Schema Properties describing the context-

sensitive conditions needed for a Schema written in this grammar to be consistent with the conceptual framework given in Geoffrion (1989a). *SMLCheck* detects and reports violations of Schema Properties as well as Schema syntax.

INTERP_CK Check the Interpretation Part of the Schema

SMLCheck delegates some of its responsibilities with respect to the relatively informal interpretation part of a Schema to *INTERP_CK*, which checks the syntax of underlined key phrases, interactively recognizes so-called "referenced key phrases," and checks the rules for proper use of referenced key phrases.

COSMET Size Schema Frames and Delete Extra Blank Lines

This purely cosmetic process beautifies on-screen displays of the Schema.

REFGEN Generate Reference Documentation

REFGEN automatically generates various reference documents on the Schema useful for communication, debugging, model maintenance and evolution, and other purposes. (Other reference documentation is generated by *EDGEN*, a process to be described in Section A.2.)

DBREF Translate Adjacency/Reachability Matrices From Text to Tables

The genus graph adjacency and reachability matrices produced by *REFGEN* are text displays. *DBREF* converts them to true tables, so that they can be manipulated by Framework's table editor.

NETDRAW Browse the Genus Graph

NETDRAW generates a Schema's genus graph as a graphical display that the user can interactively browse to better understand a model's general structure.

PrologQuery Prolog-Based Schema Query

This process provides a fully automatic interface to a commercial Prolog system (Arity 1987) in such a way that logic-based inferencing can be done with respect to all Level 1 Schema information except for the Interpretations. The process translates Schema information into Prolog predicates (facts and rules), exports these along with certain other "stock" predicates to the Prolog system, invokes the Prolog system with proper initial conditions, and deposits the user in the command-driven Prolog environment. The full power of Arity Prolog is available to the user for answering queries (drawing logical inferences) based on the available Schema information.

A.2. FW/SM Processes Useful for Level 2 and Higher

The FW/SM processes of interest for Level 2 SML are those described in Section A.1 plus:

EDGEN Generate Skeletal Elemental Detail Tables

EDGEN generates empty Elemental Detail Tables, with a flexible option to join contiguous tables. It also automatically generates two additional reference documents beyond those created by *REFGEN*. The tables all should have exactly one row at Level 2, but they can have multiple rows at Levels 3 and 4. The Elemental Detail Tables are relations in the sense of relational algebra, and FW/SM has a table editor that permits their direct manipulation.

FcEval Compile Generic Rules into C

The *EDGEN* process does not create a mechanism for doing evaluation, that is, for computing values of function and test elements. *FcEval* provides this capability by compiling generic rules into C code that can be run by the *EVALUATE* process. This code need never be regenerated so long as the Schema stays the same, thus giving the benefits of "warm restart" for multiple evaluations.

EVALUATE Perform Evaluation

EVALUATE performs generic rule evaluation using compiled C code, provided *FcEval* has been invoked previously. Results are put into the Elemental Detail Tables where they are readily accessible to Framework's table editor and to other processes. Immediate evaluation capability is useful for debugging, answering "what if" questions, for doing many kinds of analysis of the model and of derived results, and for generating reports.

A.3. FW/SM Processes Useful for Level 3 and Higher

The FW/SM processes of interest for Level 3 SML are those described in Sections A.1 and A.2 plus:

L/E UTILITIES Utilities for Loading/Editing Tables

This process, which comprises several subprocesses, supports various set and relational operations commonly needed when building and maintaining Elemental Detail Tables.

MATRIX Build Matrix From Table With Two Key Fields

Elemental Detail Tables with two key fields for attribute, function, or test genera amount to a linearized representation of a matrix. *MATRIX* transforms such an Elemental Detail Table into the corresponding matrix.

ID_DICT Build Identifier Dictionary

ID_DICT creates a data base of all Identifiers appearing in the defining Elemental Detail Tables associated with self-indexed genera. Information on defining genera and identifier interpretations is included.

TABLE RULES Check Table Content Rules

There is a collection of so-called Table Content Rules that are exhaustive in a strong sense for guaranteeing the internal consistency of the Elemental Detail Tables (Geoffrion 1990a). This process is a partial implementation of these rules.

XtrieveQuery Menu-Based Elemental Detail Table Query

This process exports all Elemental Detail Tables to Xtrieve, a commercial quasirelational query interface and processor (Novell 1988), invokes Xtrieve with proper initial conditions, and deposits the user in the menu-driven Xtrieve environment. The full power of Xtrieve is available to the user for answering queries concerning data and results, and for generating reports.

GENNETFW Invoke Generalized Network Flow Solver

This is a control table interface that builds a suitable problem file for generalized network flows and invokes the GENNET optimizer.

MPS_INTERFACE Invoke Linear/Integer Programming Solver

This is a completely automatic interface that builds a suitable MPS problem file for linear and integer programming and invokes the LINDO optimizer (Schrage 1991).

ACKNOWLEDGMENT

This work was partially supported by the National Science Foundation, the Office of Naval Research, and Shell Development Company. The views expressed are those of the author and not of the sponsors. I am indebted to the many students and colleagues who contributed to the development of SML during its long gestation in the 1980s. My greatest single debt is to Fernando Vicuña, without whom SML would never have reached its present state of development. Special thanks go also to the implementors of SML in the form of the FW/SM research prototype: Sergio Maturana, Laurel Neustadter, Yao-Chuan Tsai, and Fernando Vicuña. Finally, I thank Laurel Neustadter, Richard Ramirez, and Gordon Wright for their valuable comments on some of this material.

REFERENCES

ARITY. 1987. *Arity Prolog Version 5.0.* Arity Corporation, 30 Domino Drive, Concord, Mass.

BHARGAVA, H. K., AND S. O. KIMBROUGH. 1990. On Embedded Languages for Model Management. In *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, Vol. III, J. Nunamaker, Jr. (ed.). IEEE Computer Society Press, Washington, D.C., 443–452.

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

CHARI, S., AND R. KRISHNAN. 1990. Towards a Logical Reconstruction of Structured Modeling. Working Paper 7-89, Decision Systems Research Institute, SUPA, Carnegie-Mellon University, Pittsburgh. *Decision Support Systems* (to appear).

CONKLIN, J. 1987. Hypertext: An Introduction and Survey. *Computer* **20,** 17–41.

DATAFORM. 1987. *DATAFORM User Manual,* Release 2.0. Ketron Management Science, Inc., Arlington, Va.

FORSTER, M. 1988. A General Network Generator. In *Mathematical Models for Decision Support*, G. Mitra (ed.). Springer-Verlag, Berlin.

FOURER, R., D. M. GAY AND B. W. KERNIGHAN. 1990. A Mathematical Programming Language. *Mgmt. Sci.* **36,** 519–554.

GASS, S. I. 1984. Documenting a Computer-Based Model. *Interfaces* **14**(3), 84–93.

GEOFFRION, A. M. 1987. An Introduction to Structured Modeling. *Mgmt. Sci.* **33,** 547–588.

GEOFFRION, A. M. 1989a. The Formal Aspects of Structured Modeling. *Opns. Res.* **37,** 30–51.

GEOFFRION, A. M. 1989b. Reusing Structured Models via Model Integration. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences.* IEEE Computer Society Press, Washington, D.C., 601–611.

GEOFFRION, A. M. 1989c. Computer-Based Modeling Environments. *Eur. J. Opnl. Res.* **41,** 33–43.

GEOFFRION, A. M. 1990a. SML: A Model Definition Language for Structured Modeling. Working Paper

360, Western Management Science Institute, UCLA, Los Angeles.

GEOFFRION, A. M. 1990b. The SML Language for Structured Modeling. Working Paper 378, Western Management Science Institute, UCLA, Los Angeles.

GEOFFRION, A. M. 1991. FW/SM: A Prototype Structured Modeling Environment. Working Paper 377, Western Management Science Institute, UCLA, Los Angeles. *Mgmt. Sci.* **37**, 1513–1538.

HARARY, F., R. NORMAN AND D. CARTWRIGHT. 1965. *Structural Models: An Introduction to the Theory of Directed Graphs.* John Wiley, New York.

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

KARRENBAUER, J. J., AND G. W. GRAVES. 1989. Integrated Logistics Systems Design. In *Proceedings of the 18th Annual Transportation and Logistics Educators Conference*, J. M. Masters and C. L. Coykendale (eds.). St. Louis.

KRISHNAN, R. 1990. A Logic Modeling Language for Model Construction. *Dec. Support Systems* **6,** 123–152.

LEBLOND, G. T., AND D. F. COBB. 1985. *Using 1-2-3.* 2nd ed. Que Corporation, Indianapolis.

LENDARIS, G. G. 1980. Structural Modeling–A Tutorial Guide. *IEEE Trans. Syst. Man. Cybernet.* **SMC-10,** 807–840.

MARKOWITZ, H. M. 1979. SIMSCRIPT. In *Encyclopedia of Computer Science and Technology*, J. Belzer, A. G. Holzman and A. Kent (eds.). Marcel Dekker, New York.

MISER, H. J., AND E. S. QUADE. 1985. *Handbook of Systems Analysis.* North-Holland, New York.

NEUSTADTER, L. 1990. On the Structure of Data in SML Models. Research Paper, Anderson Graduate School of Management, UCLA, Los Angeles.

NEUSTADTER, L., A. GEOFFRION, S. MATURANA, Y. TSAI AND F. VICUÑA. 1991. The Design and Implementation of a Prototype Structured Modeling Environment. In *Model Management*, B. Shetty (ed.). Springer-Verlag, Berlin (to appear).

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

PLANE, D. R. 1986. *Quantitative Tools for Decision Support Using IFPS.* Addison-Wesley, Reading, Mass.

ROBERTS, F. S. 1976. *Discrete Mathematical Models.* Prentice-Hall, Englewood Cliffs, N.J.

SAGIE, I. 1986. Computer-Aided Modeling and Planning (CAMP). *ACM Trans. Math. Software* **12,** 225–248.

SCHRAGE, L. 1991. *LINDO: An Optimization Modeling System*, 4th ed. Scientific Press, Redwood City, Calif.

STELLA. 1989. *A User's Guide to STELLA.* High Performance Systems, 13 Dartmouth College Highway, Lyme, N.H.

WARFIELD, J. 1976. *Societal Systems: Planning, Policy, and Complexity.* John Wiley, New York.

WAYNER, P. 1988. It's APT To Write. *Byte* **13,** 375–384 (November).

WAYNER, P. 1990. BOXWEB: A Structured Outline Program for Writers. Department of Computer Science, Cornell University, Ithaca, N.Y. Presented at the Third International Conference on Computers and Writing.

WISHBOW, N., AND M. HENRION. 1987. Demos User's Manual, Version 3. Department of Engineering and Public Policy, Carnegie-Mellon University, Pittsburgh.