THE SML LANGUAGE FOR STRUCTURED MODELING: LEVELS 3 AND 4

ARTHUR M. GEOFFRION

University of California, Los Angeles, California

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

This is the second of two articles on the principal features of SML, a language for expressing structured models. The prior article covered levels 1 and 2. The present article covers the remaining levels, with 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 following the development of structured modeling.

The companion paper (Geoffrion 1992) explained SML's place in the structured modeling approach to modeling and modeling environment design, presented the main features of Levels 1 and 2 SML via a series of simple examples, and described six characteristics of SML that we believe are particularly noteworthy. That article is an essential prerequisite for the present one.

This article undertakes a similar mission for SML's Levels 3 and 4. As before, there is considerable attention to the "notable characteristics" of SML that we believe are of interest to evaluators and designers of modeling languages and systems.

One section is devoted to each of Levels 3 and 4. A concluding section describes four SML characteristics not mentioned earlier.

We continue to use *ITALIC CAPITALS* for the names of all examples.

It warrants emphasis that the informal, example-oriented sketch of SML given in this article and its companion are only superficial introductions to SML. Readers who wish to obtain a proper understanding of SML need to study Geoffrion (1990c), which is freely available from the author.

1. LEVEL 3 SML: STRUCTURED MODELING WITH SIMPLE INDEXING

Level 3 adds simple indexing structures, namely sets and Cartesian products. No longer is it true that each genus consists of exactly one element. Indexing structures make it possible for a Schema to achieve the often crucial property of "dimension independence" and thereby be very small by comparison with a complete model instance. This property—which means roughly that the size of a Schema is independent of the amount of instantiating data—and its significance are explained in Geoffrion (1990d). The addition of simple indexing has major implications for SML, as we shall see; it is responsible for essentially all of the many new language features which make their appearance at Level 3.

The introduction to Sections 1 and 2 of the prior article listed the core concepts and associated constructs of structured modeling that were pertinent to SML Levels 1 and 2. No additional concepts or constructs need to be added for Level 3 (or 4).

Lest the reader wonder why we choose not to add the core concept of *generic similarity*, the reason is that it can be shown that generic similarity holds automatically as a by-product of SML notation at Levels 3 and 4 (Geoffrion 1990a). Generic similarity holds vacuously at Levels 1 and 2 because there is only one element per genus. Similarly, we choose not to add the *model schema* concept because every SML Schema necessarily corresponds to one.

The rest of this section indicates some of the kinds of models that Level 3 SML can represent, and discusses the significance of selected characteristics of Level 3 SML.

1.1. Level 3 Models

Level 3 SML adds simple indexing to definitional systems and graphs. It also encompasses several other important classes of models, including finite predicate calculus and certain models in mathematical programming. We discuss each of these four model classes. It should be obvious, and hence, we do not discuss what

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/92/4001-0058 $01.25

indexing can do for spreadsheets, numeric formulas, and other model classes accessible at Level 2.

1.1.1. Definitional System: *DOS GLOSSARY* Revisited, *GRADEBOOK*

Simple indexing adds partial support for the fifth desirable definitional system property advocated in Geoffrion (1989a), namely grouping. But full support for grouping requires Level 4. We present two examples.

DOS GLOSSARY Revisited

Grouping can be illustrated in the context of *DOS GLOSSARY* (see Section 1.1.1 of the prior article). For example, the glossary could be turned into what is essentially a data base by letting FILE be an indexed set and changing /ce/ to /a/ in the &FILES module. Here is one possible outcome of such a change, where interpretations have been omitted for the sake of brevity:

```
&FILES
  FILEf /pe/
  F_NAME (FILEf) /a/ : String 8
  F_EXT (FILEf) /a/ : String 3
  F_SIZ (FILEf) /a/ : Integer+
  F_DATE (FILEf) /a/ : String
```

Notice that the index "f" is introduced by the genus FILE, and that the propagation of this index to the other genera turns them into attributes of the individual files.

A maximally joined Elemental Detail Table for &FILES would be named "FILE" and have this structure:

File	||	F_NAME	F_EXT	F_SIZ	F_DATE
identifier	||	*string*	*string*	*integer*	*string*
:	||	:	:	:	:
:	||	:	:	:	:

Identifiers could, for example, be sequence numbers or long names. The number of rows is indefinite. That the number of files impacts the Elemental Detail Table, but not the Schema, is the essence of the concept of "dimension independence" mentioned earlier.

GRADEBOOK

The second example of a Level 3 definitional system also has characteristics of a data base, but adds a simple mathematical calculation. Consider an instructor's gradebook for a single course. It contains a list of students and their majors, a list of items graded (e.g., homeworks and exams), grades for all items, and a final grade for each student based on certain standard weights for the graded items. Let s index students and i index the graded items. Then the final grade for student s, using obvious names for coefficients, can be written

$$\sum_i \text{WEIGHT}_i * \text{GRADE}_{si}.$$

Figure 1 gives a Level 3 Schema representing this situation. Notice that:

- two paragraphs (STUDENT and ITEM) introduce a dedicated index ("s" and "i", respectively) in a suffix to their name;
- Calling Sequence internals offer new options to cope with the fact that genera can have multiple elements; a call like STUDENTs refers to exactly one element, one like WEIGHT refers to all elements of genus WEIGHT, and one like GRADEs. (notice the dot in place of the index i) refers to all elements of genus GRADE associated with STUDENTs; see Figure 3 for concrete examples;
- @SUMi ( ) plays the role of the summation function usually represented by a Greek sigma.

A completely specified and evaluated model instance is given by Figure 1 together with the maximally joined Elemental Detail Tables shown below.

STUDENT

STUDENT	||	INTERP	MAJOR
Mary	||	Mary Jones	Business
John	||	John Smith	OR

ITEM

ITEM	||	INTERP	WEIGHT
HW1	||	first assignment	0.20
HW2	||	second assignment	0.20
EXAM	||	final exam	0.60

GRADE

STUDENT	ITEM	||	GRADE
Mary	HW1	||	84
Mary	HW2	||	100
Mary	EXAM	||	87
John	HW1	||	75
John	HW2	||	80
John	EXAM	||	85

FINALGRADE

STUDENT	||	FINALGRADE
Mary	||	89
John	||	82

Let us examine the model instance shown here for evidence of the five desirable definitional system

&STUDENT_STUFF

STUDENTs /pe/ There is a list of STUDENTS taking class xxx.

MAJOR (STUDENTs) /a/ : String Each STUDENT has a MAJOR field.

&ITEM_STUFF

ITEMi /pe/ There is a list of graded ITEMS for class xxx.

WEIGHT (ITEMi) /a/ : 0 <= Real <= 1 Each ITEM counts with a certain fractional WEIGHT.

&GRADE_STUFF

GRADE (STUDENTs, ITEMi) /a/ : 0 <= Real <= 100 Each STUDENT receives a certain percentage GRADE for each ITEM.

FINALGRADE (GRADEs., WEIGHT) /f/ ; @SUMi (WEIGHTi * GRADEsi) Each STUDENT receives a FINAL GRADE equal to the WEIGHTed average of the GRADES.

Figure 1. Level 3 Schema for *GRADEBOOK.*

properties advocated in Geoffrion (1989a). This will clarify the applicability of definitional system ideas to models that are conventionally viewed as data bases or mathematical models. Since there are 18 elements (2 students plus 2 majors plus 3 items plus 3 weights plus 6 grades plus 2 final grades), there are 18 "definitions." They are *correlated* because all definitional dependencies are explicit via the calling sequences. They are *acyclic* because, as is evident from the Schema, all elements can be arranged in a sequence so that there are no forward references (calls). They are *classified* into types: primitive entity, attribute, and function (there happen to be no compound entity or test type definitions). They are *grouped* into six clusters because there are six indexed genera. Finally, they are organized in a *hierarchical* manner because the Schema has three modules within the root module.

Note that indices allow access to individual elements. For example:

STUDENTs	stands for a typical (the *s*th) student;
STUDENT[Mary]	stands for student Mary Jones;
GRADE[John,HW1]	stands for student John Smith's grade on the first assignment.

Moreover, by substituting specific values for indices or tuples of indices, one may access any row of any Elemental Detail Table.

As pointed out in the prior article, the Level 2 nodes and arcs of a graph may now have subnodes and subarcs. This means that a genus graph no longer needs to coincide with its element graph, but can be a *condensation* of it in the graph-theoretic sense (as explained in Geoffrion 1989a).

The genus graph of *GRADEBOOK,* shown in Figure 2, is a condensation of the element graph, shown in Figure 3 for the model instance given above.

1.1.2. Graph Models With Data or Formula Attributes: *MARKOV*

A Markov chain can be represented by a graph in Level 2 SML, as in Section 2.1.2 of the prior article for *RUSSIAN ROULETTE.* But only Markov chains over a particular set of states, or at least over a definite number of states, can be represented at Level 2. There is no way to make the number of states indefinite at Level 2 because that requires the dimension independence property introduced by Level 3. It is necessary to index the set of states, and that is what we do here.

The Schema in Figure 4 represents any stationary, first-order finite state Markov chain in which all transitions are viewed as being possible (perhaps with zero probability). The ability to deal with truly impossible

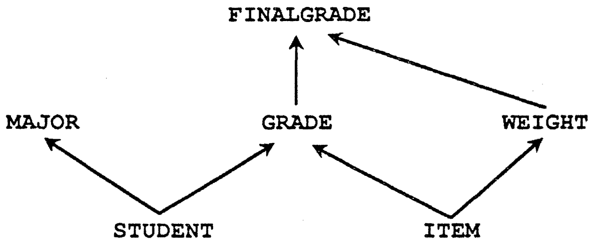

Figure 2. Genus Graph for *GRADEBOOK.*

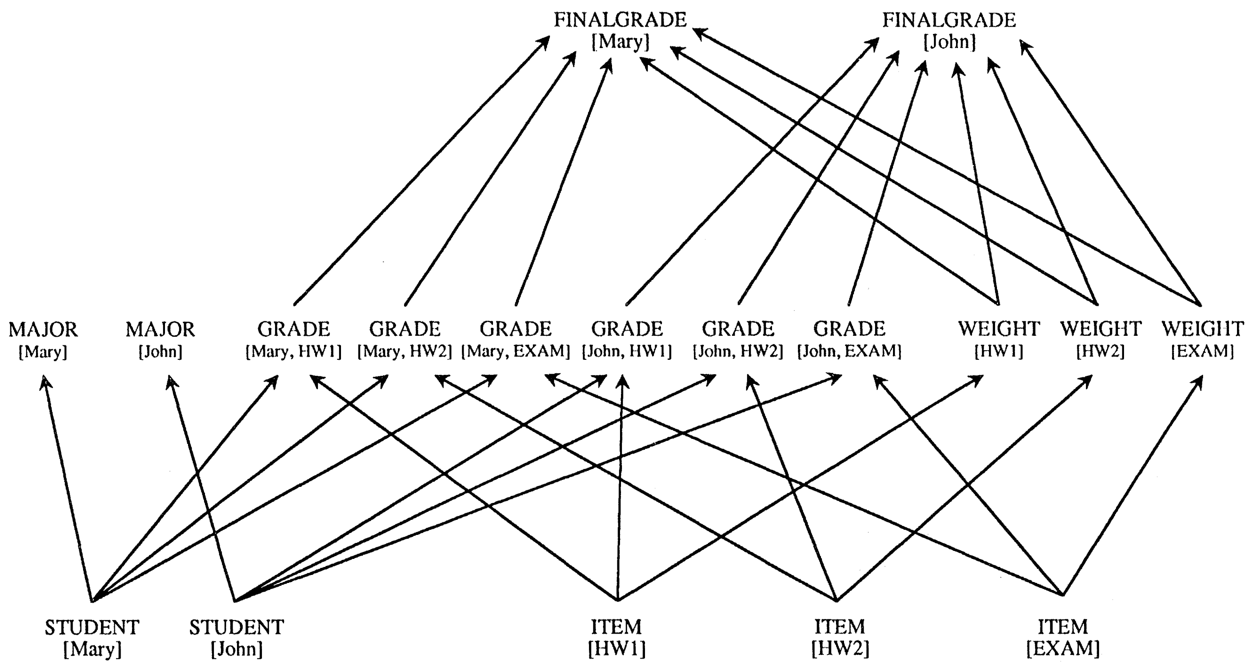

Figure 3. Element Graph for *GRADEBOOK.*

transitions by dropping them from genus TRAN, rather than by assigning them zero probability, awaits Level 4.

A valid set of associated, maximally joined Elemental Detail Tables is as follows. The data are designed to make the resulting model instance coincide with *RUSSIAN ROULETTE.*

STATE

STATE	||
ALIVE	||
DEAD	||

TRAN

STATE^1	STATE^2	||	PTRAN
ALIVE	ALIVE	||	5/6
ALIVE	DEAD	||	1/6
DEAD	DEAD	||	1
DEAD	ALIVE	||	0

T:CHECK

STATE	||	T:CHECK
ALIVE	||	#TRUE
DEAD	||	#TRUE

1.1.3. Mathematical Programming With Dense Indexing: *DENSE TRANSPORTATION*

Level 3 accommodates most mathematical programming models with simple indexing structures, namely sets and full Cartesian products.

We choose a very familiar example: the classical transportation model found in all introductory MS/OR texts. *DENSE TRANSPORTATION* assumes that all conceivable plant-to-customer transportation links exist (see Figure 5). Models in which some links do not exist must await Level 4 SML.

The associated, maximally joined Elemental Detail Tables have the following structure (the same structure as in Figure 11 of Geoffrion 1987):

Table Name	Column Names
PLANT	PLANT || INTERP SUP
CUST	CUST || INTERP DEM
LINK	PLANT CUST || FLOW COST
$	|| $
T:SUP	PLANT || T:SUP
T:DEM	CUST || T:DEM

1.1.4. Predicate Calculus Modeling: *SET COVERING*

Predicate calculus models are conspicuous by their absence in traditional operations research modeling. One reason for this is that much the same effect often can be achieved by integer programming models with 0–1 variables. The ability of 0–1 variables to represent logical model features is one of the principal reasons for the great practical importance of integer programming.

To illustrate, consider the well known set covering problem. Let a set S be given with elements indexed by i, together with some candidate subsets of S indexed by j. Let A be a matrix of 0–1 coefficients, such that A_{ij} is 1 or 0 according to whether or not element i is

STATE i,j /pe/ There is a list of STATES.

TRAN (STATEi, STATEj) /ce/ Direct TRANSITION is conceivable from STATEi to STATEj.

PTRAN (TRANij) /a/ : Real+ <= 1 There is a known TRANSITION PROBABILITY for each conceivable TRANSITION.

T:CHECK (PTRANi.) /t/ ; @SUMj (PTRANij) = 1 For each STATE, there is a CHECK on whether the outbound TRANSITION PROBABILITIES add to unity.

Figure 4. Level 3 SML Schema for *MARKOV.*

in subset j, and let c_j be the cost associated with subset j if it is selected. Furthermore, let X be a vector of 0–1 variables with the interpretation that X_j is 1 or 0 according to whether or not subset j is selected. Then the problem of finding a minimal cost cover of S can be stated as the 0–1 integer linear programming problem

$$\underset{X=0,\,1}{\text{Minimize}} \quad \textstyle\sum_j c_j X_j$$

$$\text{subject to} \quad \textstyle\sum_j A_{ij} X_j \geqslant 1 \quad \text{for all } i.$$

Note that the ith constraint can be interpreted as saying that the ith element of S must be in at least one of the selected subsets.

Figure 6 shows one way in which the underlying model can be represented in Level 3 SML. Note that the summand in the rule of function element TOTCOST uses the @IF Function, whose first argument is a logical-valued expression that determines whether it returns as its value, its second or its third argument. Note also that the rule for the test genus COVER makes use of index-supporting versions of the universal quantifier and the existential

&SDATA SOURCE DATA

PLANTi /pe/ There is a list of PLANTS.

SUP (PLANTi) /a/ : Real+ Every PLANT has a SUPPLY CAPACITY.

&CDATA CUSTOMER DATA

CUSTj /pe/ There is a list of CUSTOMERS.

DEM (CUSTj) /a/ : Real+ Every CUSTOMER has a nonnegative DEMAND.

&TDATA TRANSPORTATION DATA

LINK (PLANTi, CUSTj) /ce/ There is a transportation LINK from each PLANT to each CUSTOMER.

FLOW (LINKij) /va/ : Real+ Every LINK has a nonnegative transportation FLOW.

COST (LINKij) /a/ Every LINK has a UNIT FLOW COST.

$ (COST, FLOW) /f/ ; @SUMi SUMj (COSTij * FLOWij) There is a TOTAL COST associated with all FLOWS.

T:SUP (FLOWi., SUPi) /t/ ; @SUMj (FLOWij) <= SUPi Is the total FLOW leaving a PLANT less than or equal to its SUPPLY CAPACITY? This is called the SUPPLY TEST.

T:DEM (FLOW.j, DEMj) /t/ ; @SUMi (FLOWij) = DEMj Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the DEMAND TEST.

Figure 5. Level 3 SML Schema for *DENSE TRANSPORTATION.*

`Mi /pe/` There is a set of ELEMENTS.

`Nj (M) /ce/` There is a collection of SUBSETS of ELEMENTS.

`C (Nj) /a/` Each SUBSET has a COST.

`A (Mi, Nj) /a/ : Logical` Each ELEMENT either is or is not a MEMBER of each SUBSET.

`X (Nj) /va/ : Logical` Each SUBSET may or may not be CHOSEN.

`TOTCOST (C, X) /f/ ; @SUMj (@IF (Xj, Cj, 0))` There is a TOTAL COST associated with the CHOICE of SUBSETS.

`COVER (A, X) /t/ ; @IANDi (@IORj (@AND (Aij, Xj)))` A particular CHOICE of SUBSETS COVERS all ELEMENTS if and only if every ELEMENT is in at least one SUBSET that is CHOSEN.

Figure 6. Level 3 Schema for *SET COVERING.*

quantifier: @IANDi ( ) has the effect of ∀i, and @IORj ( ) has the effect of ∃j.

In terms of the Figure 6 Schema, the problem of finding the minimal cost cover of *S* can be stated:

$$\underset{X}{\text{Minimize}}\quad \text{TOTCOST}$$

$$\text{subject to}\quad \text{COVER} = \#\text{TRUE}.$$

We submit that it is inherently advantageous to state logical model features in logical form, rather than to convert them to ordinary algebraic form involving 0–1 quantities. The reasons are those stated early in Section 2.2 of the prior article, for the conversion of logical model features to algebraic form is well known to involve modeling tricks with undesirable side effects. If conversion to an integer linear programming model with some 0–1 variables is an appropriate step to gain access to a solver that would be useful, then conversion and deconversion should be done automatically to the extent that this is possible (cf. McKinnon and Williams 1989).

It turns out that Level 3 SML can represent any predicate calculus model with a finite number of formulas over a finite universe. Appendix J of Geoffrion (1990c) presents four additional models of this type and explains how, in general, finite predicate calculus models can be represented in Level 3 SML.

1.2. Notable SML Characteristics Apparent at Level 3

We discuss three notable characteristics of SML that first become apparent at Level 3.

Separation of General Structure and Instantiating Data

General structure refers to a class of model instances that includes nearly all the instances of possible interest. Each model instance can be represented as a particular general structure together with particular *instantiating data.*

In SML, the Schema is the main means for representing general structure, while Elemental Detail Tables are the main means for representing instantiating data. Clearly the two are strictly separated. Moreover, notice that Elemental Detail Tables do not repeat any significant amount of information from the Schema.

Not all modeling languages separate general structure and instantiating data so strictly. For example, DEMOS (Wishbow and Henrion 1987), GAMS (Brooke, Kendrick and Meeraus 1988), IFPS (Plane 1986), LINDO (Schrage 1991), and spreadsheet languages do not. Merging general structure and instantiating data may work well for small models, but this becomes undesirable as models become larger or more complex.

The importance of separation rests on the importance of general structure as a natural focus for most practical modeling work. The ability of a modeling language to achieve a sharp representation of a particular general structure, without contamination by instantiating data, confers many benefits. These include reusability, communicability, conciseness, stability, error avoidance, and dimensional flexibility. These ideas have been detailed elsewhere (see pp. 2–3 of Geoffrion 1990d, pp. 549 and 563 of Geoffrion 1987, and Geoffrion 1989b for the reuse of SML Schemas for purposes of model integration).

If general structure and instantiating data can be separated sufficiently, then it should be possible to use a modern data base system to manage the data. This point has been recognized in recent years by a number of authors including Bürger (1982), whose design for a linear programming system is one of the best-integrated in terms of an algebraic modeling language together with a supporting relational data base system for data manipulation. A telling quote follows:

In MLD, model solutions are obtained by binding a data module to a model module and executing it. This mechanism makes it easy to use the same data with different models, or to solve a model with different data. The binding between a class of models and a class of data modules is defined by a so-called execution module. The concept of an execution module has some interesting implications for an applied environment: model and execution module can be prepared by an 'expert,' while data preparation, model execution, and analysis of the model results then can be carried out by a 'client.'

To put one of Bürger's points a bit differently, one could say that general structure and instantiating data are developed and used for different purposes, often by people with vastly different backgrounds; to mix them is to inhibit efficient divisions of labor and to promote confusion.

The general structure versus instantiating data distinction has analogues in neighboring fields. In the data base management field, where it has been universally followed for perhaps two decades, it is usually referred to as "scheme vs. instance" or "intension vs. extension." In the field of computer programming, an analogous distinction underlies this quote from the guest editor of a special issue of *Computer* (Hong 1986):

In early programs, data and code were often intermixed. Modern-day programs generally have clearer separation between data and program constructs.

This same point is a strong theme in Davis (1986).

Exploitation of Parallel Structure

Geoffrion (1989a) explains (p. 33) why *grouping* is a desirable property for definitional systems. The reasons extend in an obvious way to modeling languages, and are compelling. For example, when several parts of a model have much in common, understanding any one of them can be nearly the same as understanding all of them. This makes it advantageous to treat them as a group, notationally as well as conceptually.

Indexing structures based on sets and relations are the principal means by which parallel structure has been exploited notationally by modeling languages. This topic is treated at length in Geoffrion (1990d), which includes a detailed, example-based treatment of SML's extensive support for indexing structures. The need to provide such support supplies much of the motivation for Levels 3 and 4 of SML.

Predetermined Relational Data Table Structure

The design for all of a Schema's Elemental Detail Tables is uniquely determined (up to optional joining) by rules set forth in Geoffrion (1990a). Thus, the modeler is spared the work (and associated opportunities for error) of designing tables or other data structures to hold instantiating data. The result is always a set of relations in the sense of relational algebra, with the columns to the left of the vertical double line serving as a key. Moreover, Neustadter (1990) shows that, under mild conditions, these tables are free from insertion, deletion, and update anomalies arising from functional dependencies in the sense of relational algebra (e.g., Ullman 1982).

This is significant for several reasons. First, it renders SML highly compatible with relational data base theory and systems. Relational data base technology and associated commercial software can readily be used to implement Elemental Detail Tables, to process ad hoc retrieval queries against these data via simple or sophisticated user interfaces, and to give easy data access to application software written in high level languages like C and Pascal. See, for example, Ramirez (1990) and Worobetz and Wright (1991), who describe systems based on SML that exploit this compatibility. See also Dolk (1988), Farn (1985), and Lenard (1987).

Second, SML's relational data base compatibility is transparent. The modeler only has to worry about getting the Schema's definitional system right, and not about what the relations will turn out to be and whether they will be free of update anomalies. For example, the modeler did not have to deliberately specify the Cartesian product relations in *MARKOV* and *DENSE TRANSPORTATION*; they were inferred from the associated calling sequences in the Schema.

Third, standardization benefits follow from the fact that a Schema implies the design of its associated Elemental Detail Tables. People who have to deal with multiple SML models can depend on being able to view the data in exactly the same way in all models, thereby eliminating the confusion and extra learning time occasioned by collections of models that organize their data differently for no good reason.

2. LEVEL 4 SML: STRUCTURED MODELING WITH FULL SUPPORT FOR SPARSITY

The main addition at Level 4 is full support for sparse indexing structures. This contrasts with the dense indexing structures found at Level 3. This capability turns out to be essential for many important applications (e.g., most of those involving mathematical programming). This section indicates some of the new kinds of models which Level 4 SML can represent, and discusses the significance of a tenth characteristic of SML that becomes apparent at Level 4.

2.1. Level 4 Models

Level 4 SML introduces sparsity options into all the model classes that fall within the scope of Level 3. In addition, some entirely new classes of models now can be represented, including all relational data bases and the better part of most semantic data bases. This subsection discusses a sparse mathematical programming model and selected data base models, but does not discuss what indexing can do for the other model classes accessible at Level 3.

2.1.1. Mathematical Programming With Sparse Indexing: *PRODUCTION PLANNING*

Appendix A contains *PRODUCTION PLANNING*, an SML rendering of a mathematical programming model appearing originally in Ellison and Mitra (1982), and subsequently in Lucas, Mitra and Darby-Dowman (1983), and in Hürlimann and Kohlas (1988). These references facilitate a direct comparison of SML with three other languages, namely CAMPS, LPL, and UIMP, although that will not be done here.

The most conspicuous new language feature used in Appendix A is the introduction of a so-called index set statement immediately after most genus type declarations. Its purpose is to at least partially specify the element population associated with a genus, either by an expression which makes the population explicit or by imposing constraints on the population to reflect the desires of the modeler.

Two other new language features are evident in Appendix A. Both are used in the T:BAL paragraph. One is exemplified by Y⟨i−1:i⟩k in the calling sequence, which gives finer interval control over which elements are called; it says that T:BALi calls Y_{ik} and $Y_{i-1,k}$. The other is elementary index arithmetic for simple variables, exemplified by Y⟨i−1⟩k in the generic rule, which gives more flexibility of the sort needed to deal conveniently with leads and lags in multitime period models.

Sparsity is essential in this model. It is created by the index set statements of three genus paragraphs—COMPAT, SCOST, and CAP—and is inherited by the PCOST, X, T, and T:CAP genera by a mechanism described in Geoffrion (1990c). All other genera have dense index sets, that is, all possible elements exist.

Appendix A also provides filled-in Elemental Detail Tables containing an optimal solution of an associated linear programming problem. The optimal solution was determined by a solver that treated the /va/ genera (X, Y, and Z) as "variables," the /t/ genera (T:AVAIL, T:CAP, T:DEM, and T:BAL) as "constraints," and PROFIT$ as the objective function to be minimized. Given the optimal values for the variables, structured modeling evaluation produced the values shown for the test genera, for PROFIT$, and for the function genus MUSE.

2.1.2. Relational Data Modeling: *HAPPY VALLEY*

It has been shown (e.g., Farn 1985) that any relational data base scheme and associated functional dependencies can be represented by a Level 4 SML Schema, and that any relational data base (data included) can be represented by a Level 4 SML Schema together with filled out Elemental Detail Tables. Moreover, this can be done using only a small subset of SML's features; for example, it is not necessary to use function or test genera or any but the very simplest kinds of index set statements.

Figure 7 presents an SML rendering of a relational data base scheme used extensively in Ullman (1982) to illustrate a variety of ideas.

Sparsity is essential in this model. It is created by the index set statement of OFFER (the only paragraph that is not in Level 3 SML), and is inherited by the PRICE genus. All other genera have dense index sets.

Note the appearance of $m1(o)$ in paragraph ORD. Actually a Level 3 language feature, it stands for a user-specified function from order to members (this function must be specified in an Elemental Detail Table). The meaning of $i1(o)$ in the same paragraph is similar.

A valid set of associated, maximally joined Elemental Detail Tables is given in Table I. The sample data are Ullman's (p. 97).

2.1.3. Semantic Data Modeling: *TRAVEL*

In recent years there has been a great deal of research extending the relational data model and exploring alternative data models with greater expressive power. Many of these efforts can be collected under the general heading of "semantic data modeling" (e.g.,

`&MEMBERS` The MEMBER SECTION of the database.

`MEMm /pe/` There is a list of current MEMBERS.

`MNAME (MEMm) /a/ : Char` Every MEMBER has a MEMBER NAME.

`MADDR (MEMm) /a/ : Char` Every MEMBER has a MEMBER ADDRESS.

`BAL (MEMm) /a/` Every MEMBER has a BALANCE.

`ITEMi /pe/` There is a list of ITEMS offered for sale.

`&SUPPLIERS` The SUPPLIER SECTION of the database.

`SUPs /pe/` There is a list of current SUPPLIERS.

`SNAME (SUPs) /a/ : Char` Every SUPPLIER has a SUPPLIER NAME.

`SADDR (SUPs) /a/ : Char` Every SUPPLIER has a SUPPLIER ADDRESS.

`OFFER (SUPs, ITEMi) /ce/ Select` Each SUPPLIER offers to sell certain ITEMS; the list of possibilities is known collectively as the OFFERINGS.

`PRICE (OFFERsi) /a/ : Real+` Each OFFERING has its PRICE in dollars per unit quantity.

`&ORDERS` The ORDER SECTION of the database.

`ORDo (MEMml(o), ITEMil(o)) /ce/` There is a list of ORDERS, each from one MEMBER for one ITEM. (Use "order number" as the identifier.)

`QTY (ORDo) /a/ : Real+` Every ORDER has an ORDER QUANTITY.

Figure 7. Level 4 SML Schema for *HAPPY VALLEY*.

Hull and King 1987, Peckham and Maryanski 1988).

Level 4 SML captures many of the features required for semantic data modeling. *TRAVEL* in Appendix B provides an illustration. As for the previous two examples, sparsity is essential. It is created by the index set statements of SPEAKS, GOES_TO, BUSINESS_TRAVELER, TOURIST, and ENJOYS, and is inherited by the OCCUPATION and WORDS_FOR genera. All other genera have dense index sets.

In addition to the just mentioned index set statements in five paragraphs, this Schema employs just one other language feature not available at Level 3: the @EXIST Function appearing in the next to last genus. It returns its second or third argument as its value according to whether or not the item written as the first argument—which must be a reference to a single element—exists.

Chari (1988) gives a careful discussion of how contemporary semantic data modeling compares with structured modeling. His general conclusion is that SML can represent most of the principal semantic data modeling components, and could represent virtually all of them if one minor extension were made. Even without any extensions, Farn (1985) has proven by using first-order logic techniques that structured modeling subsumes the Entity-Relationship data model, which is, by far, the most popular of the semantic data models.

2.2. Notable SML Characteristic Apparent at Level 4

We discuss a characteristic of SML that first becomes apparent at Level 4.

Sparsity Support

The importance of supporting sparsity derives from the importance of achieving accurate descriptions of general structure as it relates to parallel structure (see the first two parts of Section 1.2). The more expressive a language's mechanisms for describing sparsity, the more accurately its models can be made to describe reality, the less work a modeler needs to do to instantiate a given model class, and the more readily errors can be detected automatically. There are other benefits as well.

Elaborate support for sparse index sets is one of SML's most conspicuous characteristics. Geoffrion (1990c) identifies and discusses four distinct mechanisms, two of which may be unique to SML. The introduction to Level 4 SML given in the previous subsection is not detailed enough to support a proper explanation of these mechanisms here.

3. CONCLUSION

Selected characteristics of SML were discussed at the end of each section in both articles according to the

Table I
Elemental Detail Tables for *HAPPY VALLEY*

MEM

MEM	||	MNAME	MADDR	BAL
BB	||	Brooks, B.	7 Apple Rd.	10.50
WF	||	Field, W.	43 Cherry La.	0.00
RR	||	Robin, R.	12 Heather St.	−123.45
WH	||	Hart, W.	65 Lark Rd.	−43.00

ITEM

ITEM	||	INTERP
CU	||	Curds
GR	||	Granola
LE	||	Lettuce
SS	||	Sunflower Seeds
WH	||	Whey
UF	||	Unbleached Flour

SUP

SUP	||	SNAME	SADDR
SUN	||	Sunshine Produce	16 River Street
PUR	||	Purity Foodstuffs	180 Industrial Rd.
TAS	||	Tasti Supply Co.	17 River Street

OFFER

SUP	ITEM	||	PRICE
PUR	CU	||	0.80
PUR	GR	||	1.25
PUR	UF	||	0.65
PUR	WH	||	0.70
SUN	GR	||	1.29
SUN	LE	||	0.89
SUN	SS	||	1.09
TAS	LE	||	0.79
TAS	SS	||	1.19
TAS	WH	||	0.79

ORD

ORD	||	MEM~ml	ITEM~il	QTY
1	||	BB	GR	5
2	||	BB	UF	10
3	||	RR	GR	3
4	||	WH	WH	5
5	||	RR	SS	2
6	||	RR	LE	8

level of SML at which they first became apparent:

Level 1
- Explicit Definitional Dependencies
- Semi-Structured Sublanguage for Documentary Comments
- Insightful Graphs Always Available
- Hierarchical Organization for Managing Complexity.

Level 2
- True Logical Capability
- Separation of Models from Problem Statements and Solvers.

Level 3
- Separation of General Structure and Instantiating Data
- Exploitation of Parallel Structure
- Predetermined Relational Data Table Structure.

Level 4
- Sparsity Support.

Four additional characteristics pertaining to SML as a whole also deserve mention.

Explicit Semantic Framework

Elsewhere we have argued that modeling environment architecture should commence with the design of an explicit *semantic framework* for conceptual modeling, and that this framework should possess certain properties (Geoffrion 1989c). It was further argued that any modeling language offered within the modeling environment should rest upon the chosen framework as its supporting foundation.

The structured modeling project has followed this approach, beginning with the semantic framework put forth in Geoffrion (1989a), continuing with the design of SML to embody this framework, and following through with the development of a prototype structured modeling environment that implements SML (Geoffrion 1991).

This approach seems, unfortunately, to have little in common with the genesis of most modeling languages and systems. Very few make specific reference to a foundational semantic framework. SIMSCRIPT is one of the exceptions, having been designed in part to embody the entity-attribute-set worldview (Markowitz 1979). In contrast, most modeling languages appear to have been designed with only an implicit foundation in mind, if any. For example, algebraic modeling languages for mathematical programming seem to be based loosely on the kind of algebra commonly used to describe mathematical programs, together with such notions as objective function, variable, and constraint. These languages are a great improvement over matrix generators, for reasons that are well explained by Fourer (1983), but their designs are ad hoc and inspired more by the linear programming paradigm than by the cognitive process of modeling.

We believe that modeling languages should be built on more rigorous foundations. Besides the inherent virtues of rigor and the argument given in Geoffrion (1989c), this position enables the advantages mentioned in the discussion of the next characteristic. In addition, note that a single foundational framework has the potential to support multiple specialized languages and systems more gracefully than may be possible ad hoc in the absence of a unifying foundation, for foundational theory applies to all derivative languages and foundational concepts need only be learned once by users.

Examples of the language design approach advocated here can be found in the neighboring field of programming languages; think of the original LISP (based on the lambda calculus; see McCarthy 1960) and Prolog (based on the Horn clause subset of first order logic; see Colmerauer 1985). Might the durability and popularity of these languages have something to do with their explicit, rigorous foundations?

Exhaustive Context-Sensitive Semantic Restrictions

The fact that SML is founded on an explicit semantic framework has made it possible to supplement the usual context-free syntax of its definition with enough so-called Schema Properties and Table Content Rules (few of which have been woven into these two papers) to *guarantee the integrity of model classes and model instances written in SML*, in the sense explained below. Such a claim appears to be unique to SML.

This claim is detailed and proven in Sections 4 and 5 of Geoffrion (1990a). Loosely speaking, Proposition 2 in that paper shows that a Schema satisfying all Schema Properties must represent a "genuine" class of structured models. Proposition 1 shows that if a Schema, together with Elemental Detail Tables with fully specified attribute value columns, satisfy all Schema Properties and Table Content Rules, then they specify a "genuine" structured model instance. Here "genuine" means complying with the semantic framework of Geoffrion (1989a). Of course, these propositions cannot ensure that the model class or model instance truly fulfills what the modeler intended, but they do guarantee internal consistency and freedom from essentially all errors that can be checked for in the absence of domain-specific knowledge and metaknowledge not expressible in SML.

The significance of Propositions 1 and 2 is that it should be easier, and take less time, to develop correct new models and to maintain existing ones with SML than with other modeling languages that lack a complete set of semantic restrictions. But this hypothesis has not yet been tested in a systematic way.

See Chapter 2 of Vicuña (1990) for a more complete discussion of the value of such semantic restrictions and a detailed analysis of three contemporary modeling languages from this point of view. This dissertation also gives a formal attribute grammar specification of all SML Schema Properties and Table Content Rules. Moreover, it uses this specification as the main input for a tool that generates a language-directed editor (e.g., Reps and Teitelbaum 1987) for SML. This illustrates a second persuasive motivation for making a modeling language's semantic restrictions explicit: So that they can be used in conjunction with modern software development tools to generate portions of a modeling environment. As Vicuña explains, this approach extends beyond language-directed editors to other important functionalities.

Standardization Through Generality

Other things being equal, there are significant advantages to reducing the number of different modeling paradigms or representational styles: a) communication is facilitated because fewer paradigms or styles must be learned and remembered by those who work with them, b) model integration is facilitated because more of the things to be integrated are expressed in a common paradigm or style, and c) concentration on fewer paradigms and styles justifies greater investment in each.

Thus, it is advantageous for a modeling language to have the broadest practical scope of applicability. The wider the scope, the greater its potential value as a standardized modeling paradigm and representational style.

It is natural to think of "scope" relative to the model classes arising in MS/OR and kindred model-based fields, but actually it suffices to think of "scope" relative to a single business or other type of organization. The benefits of standardization can be obtained by using a single modeling language in just one organization, or even in one part of one organization.

One reason for the phenomenal success of spreadsheet modeling is its emergence as a kind of modeling language standard owing to its great flexibility to represent model instances. That this should have occurred even though it fares very poorly for representing model classes, can be interpreted as testimony to the value of standardization through generality.

The examples and discussion given in the first two sections of this and the companion paper indicate the remarkable generality of SML. Many additional examples can be found in Geoffrion (1990b), which contains models drawn from a variety of application areas. It would appear that SML has the potential to become a modeling language standard within some organizations, and perhaps even for some classes of models across organizations.

Executability

A modeling language can be promising in concept but impractical as a basis for achieving essential modeling environment functionality. The FW/SM prototype, which implements SML, shows that SML does not fall into this trap. These ideas are discussed in detail in Section 2.2 of Geoffrion (1989c) and in Section 4.2 of Geoffrion (1991). The four kinds of executability discussed there are error-trapping, automatic documentation, solver interface setup, and a smart loader/editor for instantiating data. The ability of SML to support these kinds of executability can be explained to a great extent by several of the SML characteristics discussed previously.

SML's executability is demonstrated independently in Vicuña (1990) which, as mentioned, achieves an independent implementation of SML using attribute grammar technology on a UNIX work station. The language-directed editor for SML described there enables even newcomers to SML to write Schemas that satisfy all SML syntax and Schema Properties because it rejects all mistakes. It can do the same for Elemental Detail Tables by enforcing Table Content Rules, although not all of these rules have been implemented.

It is remarkable that Vicuña's may be the first language-directed editor for any modeling language. We believe that this is an important kind of functionality for modeling environments of the future. A language-directed editor for any modeling language can improve productivity by preventing errors, and can relieve users of the burden of having to keep many language details constantly in mind. Such an editor is a special boon for highly expressive languages like SML, which necessarily involve considerable complexity. In SML's case, since its Schema Properties and Table Content Rules are exhaustive in the sense explained earlier, a language-directed editor can *guarantee* that every model built with it will be genuine with respect to structured modeling's semantic framework.

An alternative technology for achieving a language-directed editor and evaluator for structured modeling, namely graph grammars, is developed in Jones (1991). This is a particularly promising direction of work because it marries structured modeling, whose core concepts rely on attributed graphs, with graph-based modeling systems, whose interfaces have great visual appeal. One of the most challenging design problems in this regard is to determine which constructs of text-based SML can be replaced more effectively by graphical constructs for model representation purposes. It is not clear at this time which hybrid of these two styles will prove best in the end.

APPENDIX A

Level 4 Schema and Sample Elemental Detail Tables for *PRODUCTION PLANNING*

This model is introduced in Section 2.1.1. The general structure and sample data are based on Section 3 of Ellison and Mitra (1982). We give an algebraic formulation before giving an SML version.

The letter l (used as the index for machines) is easily confused with the numeral *1*. A symbol change was not made to preserve the greatest possible similarity to the cited reference.

A.1. Formulation in Ordinary Algebra

Ellison and Mitra actually present a single model instance. However, it is easy to discern a class of similar model instances, and that is what is presented here. We choose to be somewhat laconic in the documentary style of this formulation because the SML formulation gives complete details.

Indices

i	time periods (I periods in all);
j	production modes;
k	products;
l	machines.

Coefficients

t_{ijkl}	the time standard (hours per unit) for producing product k in period i in mode j on machine l;
a_{ijl}	the availability (hours) in period i and mode j of machine l;
p_{ik}	the selling price ($/unit) in period i of product k;
d_{ik}	the demand (units) in period i for product k;
s_{ik}	the storage cost ($/unit-period) for one-period carryover in period i for product k, excluding $i = I$;
h_{ik}	the storage capacity (units) for one-period carryover in period i for product k, excluding $i = I$;
r_k	the resale value ($/unit) for product k at the end of the final period;
c_{ijkl}	the production cost ($/unit) in period i for producing product k in mode j on machine l.

Variables

x_{ijkl}	the production quantity (units) in period i and mode j for product k on machine l
y_{ik}	the storage quantity (units) in period i for product k (in the last period, this activity is really liquidation);
z_{ik}	the sales quantity (units) in period i for product k.

Objective Function (Maximand)

$$\sum_i\sum_j\sum_k\sum_l(p_{ik} - c_{ijkl})x_{ijkl} - \sum_{i<I}\sum_k s_{ik}y_{ik} + \sum_k(r_k - p_{Ik})y_{Ik}$$

Constraints

Machine availability limits

$$\sum_k t_{ijkl}x_{ijkl} \leqslant a_{ijl} \quad \text{for all } i, j, l. \qquad (1)$$

Material balance in the first period

$$\sum_j\sum_l x_{1jkl} = y_{1k} + z_{1k} \quad \text{for all } k. \qquad (2a)$$

Material balance in periods after the first

$$\sum_j\sum_l x_{ijkl} + y_{i-1k} = y_{ik} + z_{ik} \quad \text{for all } i \geqslant 2, k. \qquad (2b)$$

Storage capacity limits

$$y_{ik} \leqslant h_{ik} \quad \text{for all } i \leqslant I, k. \qquad (3)$$

Demand requirements

$$z_{ik} \geqslant d_{ik} \quad \text{for all } i, k. \qquad (4)$$

Variable nonnegativity

$$y_{ik} \geqslant 0 \quad \text{for all } i, k,$$

$$x_{ijkl} \geqslant 0 \quad \text{for all } i, j, k, l. \qquad (5)$$

Sparsity

The above formulation is almost nonsparse. The only sparsity comes from the fact that s_{ik} and h_{ik} are not defined for the last period. This leads to the $\sum_{i<I}$ operator in the second term of the objective function and to the exclusion of $i = I$ in constraint set (3).

A second and much more subtle kind of sparsity is evident from the data given in Ellison and Mitra: *not all products can be produced on all machines.* That is, certain kl combinations are incompatible. This is totally overlooked in the above algebraic formulation, as it was in Ellison and Mitra's algebraic formulation. It is possible to accommodate this incompatibility by making t_{ijkl} extremely large for the incompatible kl combinations, but that would be semantically corrupt.

The only semantically correct way to represent the second kind of sparsity is to introduce a set of pairs that gives the compatible kl combinations, and to honor that set everywhere k and l appear in combination with one another: that is, in the definitions of t_{ijkl} and c_{ijkl} and x_{ijkl}, in the summations appearing in the second term of the objective function and in constraints (1) and (2), and in the second part of constraint (5). We leave such a modification of the above algebraic formulation as an exercise for the reader. The notation becomes considerably messier, as it usually does for conventional algebraic notation with any but the simplest kinds of sparsity.

A.2. SML Formulation

The SML formulation given below takes full account of sparsity. It should be straightforward for the reader to see its correspondence with the algebraic formulation.

PERi /pe/ There is an ordered list of time PERIODS (seasons).

PRODk /pe/ There is a list of PRODUCTS.

&PRODUCTION PRODUCTION DATA

MACHl /pe/ There is a list of MACHINES.

COMPAT (PRODk, MACHl) /ce/ Select {PROD} × {MACH} where k covers {PROD}, l covers {MACH} Certain PRODUCT-MACHINE combinations are COMPATIBLE and some are not. For each PRODUCT there is a MACHINE that is COMPATIBLE, and for every MACHINE there is a PRODUCT that is COMPATIBLE.

MODEj /pe/ There is a list of production MODES.

PCOST (PERi, MODEj, COMPATkl) /a/ ({PER} × {MODE}) × {COMPAT} There is a PRODUCTION COST ($/unit) for each COMPATIBLE combination of PERIOD, MODE, PRODUCT and MACHINE.

X (PERi, MODEj, COMPATkl) /va/ ({PER} × {MODE}) × {COMPAT} : Real+ A nonnegative PRODUCTION QUANTITY (units) must be decided for each COMPATIBLE combination of PERIOD, MODE, PRODUCT, and MACHINE.

T (PERi, MODEj, COMPATkl) /a/ ({PER} × {MODE}) × {COMPAT} : Real+ There is a TIME STANDARD (hours/unit) for the production of each PRODUCT on each COMPATIBLE MACHINE, by PERIOD and MODE.

MUSE (Tij.l, Xij.l) /f/ ({PER} × {MODE}) × {MACH} ; @SUMk (Tijkl * Xijkl) For each MACHINE there is a utilization measure called MACHINE USE (hours) by PERIOD and MODE which converts PRODUCTION QUANTITIES into hours according to the TIME STANDARDS.

AVAIL (PERi, MODEj, MACHl) /a/ ({PER} × {MODE}) × {MACH} : Real+ Each MACHINE has an AVAILABILITY (hours) in each PERIOD and MODE.

T:AVAIL (MUSEijl, AVAILijl) /t/ ({PER} × {MODE}) × {MACH}; MUSEijl <= AVAILijl An AVAILABILITY TEST checks, for each combination of PERIOD, MODE, and MACHINE, whether MACHINE USE is within machine AVAILABILITY.

&STORAGE STORAGE DATA

SCOST (PERIi, PRODk) /a/ (Filter(i<−1) {PER}) × {PROD} For each PRODUCT there is a STORAGE COST ($/unit-period) for carryover from one PERIOD (except the last) to the next.

CAP (PERi, PRODk) /a/ (Filter (i<−1) {PER}) × {PROD} For each PRODUCT there is a limited STORAGE CAPACITY (units) for carryover from one PERIOD (except the last) to the next.

Y (PERi, PRODk) /va/ {PER} × {PROD} : Real+ A nonnegative STORAGE QUANTITY (units) must be decided for each PRODUCT in each PERIOD. (For the last PERIOD, Yik is really a liquidation activity rather than a storage activity.)

T:CAP (Yik, CAPik) /t/ (Filter (i<−1) {PER}) × {PROD} ; Yik <= CAPik There is a STORAGE CAPACITY TEST for each PRODUCT in each PERIOD (except the last) to check whether the STORAGE QUANTITY is within the STORAGE CAPACITY.

&SALES SALES DATA

RVAL (PRODk, PER<−1>) /a/ {PROD} Each PRODUCT has a RESALE VALUE ($/unit) at the end of the final PERIOD.

PRICE (PERi, PRODk) /a/ {PER} × {PROD} There is a SELLING PRICE ($/unit) for each PRODUCT in each PERIOD. This is a “vintage” price, that is, it depends on when the PRODUCT is made rather than when it is sold.

Z (PERi, PRODk) /va/ {PER} × {PROD} A SALES QUANTITY (units) must be decided for each PRODUCT in each PERIOD.

DEM (PERi, PRODk) /a/ {PER} × {PROD} : Real+ There is a DEMAND (units) for each PRODUCT in each PERIOD.

T:DEM (Zik, DEMik) /t/ {PER} × {PROD} ; Zik >= DEMik There is a DEMAND TEST for each PRODUCT each PERIOD to check whether the total SALES QUANTITY is greater than or equal to the DEMAND.

T:BAL (Xi.k., Y<i−1:i>k, Yik, Zik) /t/ {PER} × {PROD} ; @SUMj SUMl (Xijkl) + @IF (i=1, 0, Y<i−1>k) = Yik + Zik There is a MATERIAL BALANCE TEST for each PRODUCT each PERIOD.

PROFIT$ (PRICE, PCOST, X, SCOST, Y, RVAL) /f/ 1 ; @SUMi SUMj SUMk SUMl ((PRICEik − PCOSTijkl) * Xijkl) − @SUMi<1:−2> SUMk (SCOSTik * Yik) + @SUMk ((RVALk − PRICE<−1>k) * Y<−1>k) The TOTAL PROFIT ($) takes into account all revenues and costs. (The validity of the formula for TOTAL PROFIT depends upon: a) the vintage interpretation of SELLING PRICE mentioned earlier, and b) an assumption that all unsold merchandise in the final PERIOD is liquidated rather than stored.)

We give a valid set of associated, maximally joined Elemental Detail Tables in Table II. The values for the /va/ elements are for an optimal solution to an associated linear programming problem, namely the one which maximizes PROFIT$ by choosing values for X, Y, and Z subject to all /t/ elements being true.

APPENDIX B

Level 4 Schema and Elemental Detail Table Structure for *TRAVEL*

This model is introduced in Section 2.1.3. The Schema is an equivalent rendering of the main example in Hull and King (1987). Thus, one can say that this model falls within the semantic data base modeling tradition.

&ADDRESS ADDRESS DATA

ADDRESSa /pe/ There is a list of ADDRESSES.

STREET (ADDRESSa) /a/ : Char There is a STREET associated with every ADDRESS.

CITY (ADDRESSa) /a/ : Char There is a CITY associated with every ADDRESS.

ZIP (ADDRESSa) /a/ : Char There is a ZIP associated with every ADDRESS.

&BUSINESS BUSINESS DATA

BUSINESSb /pe/ There is a list of BUSINESSES.

BNAME (BUSINESSb) /a/ : Char Unique Every BUSINESS has a unique BUSINESS NAME.

LOCATED_AT (BUSINESSb, ADDRESSa1(b)) /ce/ Every BUSINESS is LOCATED AT at some particular ADDRESS.

Table II
Elemental Detail Tables for *PRODUCTION PLANNING*

PER

PER	‖	INTERP
1	‖	Summer
2	‖	Winter

PROD

PROD	‖	INTERP
P1	‖	Nuts
P2	‖	Bolts
P3	‖	Washers

MACH

MACH	‖	INTERP
M1	‖	Machine 1
M2	‖	Machine 2
M3	‖	Machine 3

PCOST

PER	MODE	PROD	MACH	‖	PCOST	X	T
1	N	P1	M1	‖	2	0.000	4
1	N	P1	M2	‖	4	0.000	7
1	N	P1	M3	‖	1	13.333	3
1	N	P2	M1	‖	3	20.000	5
1	N	P2	M2	‖	3	0.000	6
1	N	P3	M1	‖	4	0.000	6
1	N	P3	M2	‖	2	16.667	6
1	O	P1	M1	‖	3	19.556	3
1	O	P1	M2	‖	5	0.000	6
1	O	P1	M3	‖	2	15.000	2
1	O	P2	M1	‖	4	5.333	4
1	O	P2	M2	‖	4	4.667	5
1	O	P3	M1	‖	5	0.000	5
1	O	P3	M2	‖	3	13.333	5
2	N	P1	M1	‖	3	0.857	5
2	N	P1	M2	‖	5	0.000	8
2	N	P1	M3	‖	2	12.500	4
2	N	P2	M1	‖	4	17.619	6
2	N	P2	M2	‖	4	0.000	7
2	N	P3	M1	‖	5	0.000	7
2	N	P3	M2	‖	3	15.714	7
2	O	P1	M1	‖	4	22.500	4
2	O	P1	M2	‖	6	0.000	7
2	O	P1	M3	‖	3	13.333	3
2	O	P2	M1	‖	5	0.000	5
2	O	P2	M2	‖	5	7.381	6
2	O	P3	M1	‖	6	0.000	5
2	O	P3	M2	‖	4	9.286	6

MUSE

PER	MODE	MACH	‖	MUSE	AVAIL	T:AVAIL
1	N	M1	‖	100	100	#TRUE
1	N	M2	‖	100	100	#TRUE
1	N	M3	‖	40	40	#TRUE
1	O	M1	‖	80	80	#TRUE
1	O	M2	‖	90	90	#TRUE
1	O	M3	‖	30	30	#TRUE
2	N	M1	‖	110	110	#TRUE
2	N	M2	‖	110	110	#TRUE
2	N	M3	‖	50	50	#TRUE
2	O	M1	‖	90	90	#TRUE
2	O	M2	‖	100	100	#TRUE
2	O	M3	‖	40	40	#TRUE

PRICE

PER	PROD	‖	PRICE	Z	DEM	T:DEM	T:BAL
1	P1	‖	10	47.889	25	#TRUE	#TRUE
1	P2	‖	10	30	30	#TRUE	#TRUE
1	P3	‖	9	30	30	#TRUE	#TRUE
2	P1	‖	11	49.190	30	#TRUE	#TRUE
2	P2	‖	11	25	25	#TRUE	#TRUE
2	P3	‖	10	25	25	#TRUE	#TRUE

COMPAT

PROD	MACH	‖
P1	M1	‖
P1	M2	‖
P1	M3	‖
P2	M1	‖
P2	M2	‖
P3	M1	‖
P3	M2	‖

MODE

MODE	‖	INTERP
N	‖	Normal
O	‖	Overtime

SCOST

PER	PROD	‖	SCOST	CAP
1	P1	‖	1	20
1	P2	‖	1	20
1	P3	‖	1	0

Y

PER	PROD	‖	Y
1	P1	‖	0
1	P2	‖	0
1	P3	‖	0
2	P1	‖	0
2	P2	‖	0
2	P3	‖	0

T:CAP

PER	PROD	‖	T:CAP
1	P1	‖	#TRUE
1	P2	‖	#TRUE
1	P3	‖	#TRUE

RVAL

PROD	‖	RVAL
P1	‖	2
P2	‖	2
P3	‖	1

PROFIT$

‖	PROFIT$
‖	1490.41

Table III
Elemental Detail Table for *TRAVEL*

Table Name	Designs Column Name
ADDRESS	ADDRESS ‖ INTERP STREET CITY ZIP
BUSINESS	BUSINESS ‖ INTERP BNAME ADDRESS~a1
LANGUAGE	LANGUAGE ‖ INTERP
DESTINATION	DESTINATION ‖ INTERP
PERSON	PERSON ‖ INTERP PNAME ADDRESS~a2
SPEAKS	PERSON LANGUAGE ‖
GOES_TO	PERSON DESTINATION ‖
BUSINESS_TRAVELER	PERSON ‖ OCCUPATION BUSINESS~b1
ACTIVITY	ACTIVITY ‖ INTERP
TOURIST	PERSON ‖
ENJOYS	PERSON ACTIVITY ‖
LANG_COUNT	PERSON ‖ LANG_COUNT T:LINGUIST

LANGUAGEg /pe/ There is a list of LANGUAGES.

DESTINATIONd /pe/ There is a list of possible DESTINATIONS.

&PERSON PERSON DATA

PERSONp /pe/ There is a list of PERSONS.

PNAME (PERSONp) /a/ : Char Unique Every PERSON has a unique NAME.

LIVES_AT (PERSONp, ADDRESSa2 (p)) /ce/ Every PERSON LIVES AT at some particular ADDRESS.

SPEAKS (PERSONp, LANGUAGEg) /ce/ Select where p Covers {PERSON} Every PERSON SPEAKS at least one LANGUAGE.

GOES_TO (PERSONp, DESTINATIONd) /ce/ Select Some PERSONS GO TO one or more DESTINATIONS.

&TRAVELER TRAVELER DATA

BUSINESS_TRAVELER (PERSONp) /ce/ Select Some PERSONS are also BUSINESS TRAVELERS.

OCCUPATION (BUSINESS_TRAVELERp) /a/ : Char Every BUSINESS TRAVELER has an OCCUPATION.

WORKS_FOR (BUSINESS_TRAVELERp, BUSINESSb1 (p)) /ce/ Every BUSINESS TRAVELER WORKS for a particular BUSINESS.

&TOURIST TOURIST DATA

ACTIVITYt /pe/ There is a list of ACTIVITIES.

TOURIST (PERSONp) /ce/ Select Some PERSONS are also TOURISTS.

ENJOYS (TOURISTp, ACTIVITYt) /ce/ Select Different TOURISTS ENJOY different ACTIVITIES.

&LINGUIST LINGUIST DATA

LANG_COUNT (SPEAKSp.) /f/ ; @SUMg (@EXIST (SPEAKSpg, 1, 0)) L COUNT is the number of LANGUAGES that a PERSON is able to SPEAK.

T:LINGUIST (LANG_COUNTp) /t/ ; LANG_COUNTp >= 2 A LINGUIST is a PERSON who SPEAKS at least two LANGUAGES. (Note: #TRUE indicates a LINGUIST, #FALSE a non-LINGUIST.)

The associated, maximally joined Elemental Detail Tables have the structure given in Table III.

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

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

BÜRGER, W. F. 1982. MLD: A Language and Data Base for Modeling. Research Report RC 9639, IBM T. J. Watson Research Center, Yorktown Heights, N.Y.

CHARI, S. 1988. Knowledge Representation Using Structured Modeling. Ph.D. Dissertation, Anderson Graduate School of Management, UCLA, Los Angeles.

COLMERAUER, A. 1985. Prolog in 10 Figures. *Commun. ACM* **28,** 1296–1310.

DAVIS, R. 1986. Knowledge-Based Systems. *Science* **231,** 957–963.

DOLK, D. R. 1988. Model Management and Structured Modeling: The Role of an Information Resource Dictionary System. *Commun. ACM* **31,** 704–718.

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

FARN, C. K. 1985. An Integrated Information System Architecture Based on Structured Modeling. Ph.D. Dissertation, Anderson Graduate School of Management, UCLA, Los Angeles.

FOURER, R. 1983. Modeling Languages Versus Matrix Generators for Linear Programming. *ACM Trans. Math. Software* **9,** 143–183.

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. A Library of Structured Models. Informal Note, Anderson Graduate School of Management, UCLA, Los Angeles.

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

GEOFFRION, A. M. 1990d. Indexing in Modeling Languages for Mathematical Programming. Working Paper 371, Western Management Science Institute, UCLA, Los Angeles. Extract to appear in *Mgmt. Sci.* Portions appeared as "A Taxonomy of Indexing Structures for Mathematical Programming Modeling Languages," in *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences,* Volume III. IEEE Computer Society Press, Washington, D.C., 463–473.

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

GEOFFRION, A. M. 1992. The SML Language for Structured Modeling: Levels 1 and 2. *Opns. Res.* **40,** 38–57.

HONG, S. J. 1986. Guest Editor's Introduction. *Computer* **19,** 12–15.

HULL, R., AND R. KING. 1987. Semantic Database Modeling: Survey, Applications, and Research Issues. *Computing Surveys* **19,** 201–260.

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

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

LENARD, M. 1987. An Object-Oriented Approach to Model Management. In *Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences,* Volume I. IEEE Computer Society Press, Washington, D.C., 509–515.

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

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.

MCCARTHY, J. 1960. Recursive Functions of Symbolic Expressions and Their Computation by Machine. *Commun. ACM* **7,** 184–195.

MCKINNON, K. I. M., AND H. P. WILLIAMS. 1989. Constructing Integer Programming Models by the Predicate Calculus. *Annals Opns. Res.* **21,** 227–246.

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

PECKHAM, J., AND F. MARYANSKI. 1988. Semantic Data Models. *ACM Computing Surveys* **20,** 153–189.

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

RAMIREZ, R. 1990. The ASUMMS Project: An Overview. Department of Decision and Information Systems, Arizona State University, Tucson.

REPS, T., AND T. TEITELBAUM. 1987. Language Processing in Program Editors. *Computer* **20,** 29–40.

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

ULLMAN, J. D. 1982. *Principles of Database Systems*, 2nd ed. Computer Science Press, Rockville, Md.

VICUÑA, F. 1990. Semantic Formalization in Mathematical Modeling Languages. Ph.D. Dissertation, Computer Science Department, UCLA, Los Angeles.

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

WOROBETZ, N. D., AND G. P. WRIGHT. 1991. Data Model Representation and Implementation in OR/SM, A Model Management System. Draft Working Paper, Krannert Graduate School of Management, Purdue University, West Lafayette, Ind.