The Formal Aspects of Structured Modeling

Arthur M. Geoffrion

Operations Research, Vol. 37, No. 1 (Jan. - Feb., 1989), 30-51.

Stable URL:
http://links.jstor.org/sici?sici=0030-364X%28198901%2F02%2937%3A1%3C30%3ATFAOSM%3E2.0.CO%3B2-I

Operations Research is currently published by INFORMS.

THE FORMAL ASPECTS OF STRUCTURED MODELING

ARTHUR M. GEOFFRION

University of California, Los Angeles, California

(Received May 1987; revision received August 1988; accepted September 1988)

Structured modeling is an approach to the development of a new generation of computer-based modeling environments. This paper, which is part of a series, presents a formal development of the definitions and theory of structured modeling.

The author's paper, "An Introduction to Structured Modeling" (Geoffrion 1987a), is an informal, example-based exposition of structured modeling as an approach to the development of a new generation of computer-based modeling environments. It gives a detailed motivation for and introduction to structured modeling, its uses, and its connections to other modeling approaches, systems, and fields. Familiarity with the first three sections is desirable, although not essential.

The present paper, in contrast, presents a complete, formal development of structured modeling. The aim of completeness makes some overlap with Geoffrion (1987a) inevitable.

The first section briefly reviews the kind of modeling environment which the author hopes to help bring into being. It then motivates structured modeling as a formalism for definitional systems, a modeling approach of great generality.

The second section presents and illustrates the basic definitions of structured modeling. The third gives selected additional definitions and concepts that are useful for reasoning about and communicating structured models.

The fourth section develops related theoretical results, and the final section gives a brief conclusion.

The mathematical prerequisites of this paper are modest. Elementary directed graph theory is the main area requiring some prior familiarity. The terminology used is standard (**node**, **arc**, **directed cycle** and **chain**, **acyclicity**, etc.). **Multiple arcs** (more than one arc between a given pair of nodes) are permitted. The term **rooted tree** means a finite, connected, directed graph with no loops, only one node with outdegree 0, namely the **root**, and all other nodes with outdegree 1. The nodes with indegree 0 are the **terminal** nodes. The immediate descendants of any given node bear a **sibling** relationship to one another. Every node in a rooted tree has a unique **rootpath** that begins with the node and ends with the root. Since arc orientation is obvious under the above definition, one need not bother to indicate orientation when drawing rooted trees. When there is no danger of confusion, we may say simply **tree** instead of "rooted tree."

A **tuple** is a finite nonempty ordered collection of components. A tuple is **segmented** when its components are partitioned in a contiguous way with nonempty segments. It is permissible for there to be but a single segment.

A **partition** has the usual set theoretic definition. A few other mathematical ideas are defined as the need arises.

1. Motivation

The purpose of structured modeling is to provide a foundation for computer-based modeling environments. As explained in Geoffrion (1989), the kind of "modeling environment" we have in mind should: 1) nurture the entire modeling life-cycle, not just part of it; 2) be hospitable to decision and policy makers, not just to modeling professionals; 3) facilitate the ongoing evolution of the models and systems built within it; 4) enable all of its inhabitants to use the same paradigm-neutral language for model definition; and 5) facilitate good management of key resources,

Subject classification: Philosophy of modeling: a formalism for model description. Information systems, decision support systems: foundations for the design of. Networks/graphs, theory: graph-based paradigm for model description.

0030-364X/89/3701-0030 $01.25

namely data, models, solvers, and knowledge derived from these.

Such modeling environments should enable a substantial improvement in the productivity and quality of model-based work, and also in the frequency with which decision and policy makers turn to models for assistance.

The design and construction of such modeling environments is a very ambitious goal. As further explained in Geoffrion (1989), realizing this goal involves surmounting three main challenges:

1. designing a suitable framework for conceptual modeling,
2. designing an executable modeling language that supports this framework, and
3. designing software integration approaches able to deal with the wide spectrum of components constituting a modeling environment.

A conceptual modeling framework will not be suitable unless it is formal, widely applicable, understandable and natural for the main players at each stage of the modeling life cycle, paradigm-neutral and yet compatible with most paradigms for modeling and model manipulation, and consistent with "good" modeling style. An executable modeling language should possess these same properties.

Structured modeling provides a suitable framework for conceptual modeling. We present that framework in detail in Section 2. Elsewhere (Geoffrion 1988), we present a modeling language called SML that supports this framework. A research prototype modeling environment based on SML, called FW/SM, is in an advanced stage of development and will be described in a future paper.

A few words are in order about the structured modeling framework for conceptual modeling. It can be viewed from several perspectives, some of which have much in common with mathematical modeling formalisms used in MS/OR, with data modeling formalisms used in data base theory, and with knowledge representation formalisms used in artificial intelligence. This point is discussed in Geoffrion (1987a, b, 1989). For example, as mentioned in Geoffrion (1987a), the structured modeling framework can be viewed as being based on discrete mathematics: it uses a hierarchically organized, partitioned, and attributed acyclic graph to represent a model instance or a class of model instances, with particular attention given to representing semantic as well as mathematical structure. Moreover, it is compatible with the fundamental model manipulations of information retrieval, expression evaluation, solving a simultaneous system, and optimization.

The structured modeling framework can be viewed also as a formalism for definitional systems. We develop this idea in some detail because it provides useful motivation for much of what follows.

1.1. Definitional Systems

What is modeling? A very general answer is that it is the process of giving sharp definition to "knowledge" about some part of "reality." It is in this sense that one may choose to view modeling as constructing a system of definitions. Sometimes an individual definition is called a model *element*.

A "definitional system" comprises, of course, a collection of related definitions. Such a collection should be written in a deliberate style that exhibits properties appropriate to its intended uses. We shall explain five properties exhibited by definitional systems written in structured modeling style, but not until we introduce two small examples of definitional systems.

There is a severe obstacle to be overcome in giving these examples: we do not have access to a consistent set of notational conventions for structured modeling. The reason is that a notation cannot be described properly before knowing what conceptual abstractions it is designed to support, and even if it could be, the only extant notation—SML (Geoffrion 1988)—would take an excessive amount of space to explain. We could appeal to the brief introduction to SML given in Geoffrion (1987a), but we elect to keep this paper self-contained by introducing a simple ad hoc notation. This notation is adequate only for the limited purposes of this paper. We implore readers not to let it influence their judgment of structured modeling.

The ad hoc notational conventions used here are as follows.

- Each definition is numbered.
- Each definition includes an underlined *key phrase* uniquely associated with that definition; it consists of one or more words, with spaces between words replaced by underscores.
- If a given definition is directly conceptually dependent on another definition, then the key phrase of the other definition should appear, in capitals, in the given definition. Otherwise, it should not.
- Only key phrases are capitalized.

The first example is not the sort that one ordinarily thinks of as a "model," but it seems appropriate to begin with a tiny fragment of a definitional system that overtly presents itself as such. It is adapted from

a well known dictionary of mathematics (James and Beckenbach 1976), and includes all of the definitions needed to define a transitive relation.

1. A SET is a collection of particular things.
2. An ORDERED_PAIR is a SET with two members for which one member is designated as the first and the other as the second.
3. A RELATION is a SET of ORDERED_PAIRS (x, y), it being said that x bears the RELATION to y if and only if (x, y) is a member of the SET.
4. A TRANSITIVE_RELATION is a RELATION with this property: if x bears the RELATION to y and y bears the RELATION to z, then x bears the RELATION to z.

The second example is an instance of an ordinary Hitchcock-Koopmans transportation model with two sources and three destinations. See Figure 1 of the Appendix. The Appendix also contains numerous other figures relating to this example. They are placed there rather than in the main body of the paper to avoid interrupting the flow of the main text.

Now we use these examples to illustrate five desirable properties of a definitional system. Please keep in mind that the examples just presented do not necessarily exhibit these properties, but a structured model rendering of them would do so.

Correlation

The first desirable property is that definitions should be correlated, that is, definitional interdependencies should be explicit. This property is the reason why capitalization was used as it was in our ad hoc notation. Consider, for instance, the first example. The following diagram of definitional dependency is easy to construct from the definitions given. There is a node for each such concept and a directed arc each time one concept participates in the definition of another.

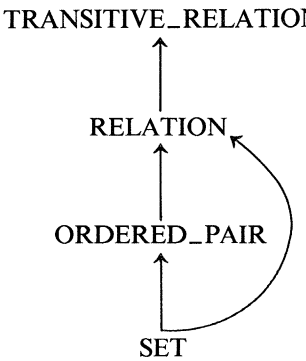

Figure 10 is a similar diagram for the other example.

One advantage of making all definitional dependencies explicit is that it facilitates tracing the origins of any definition as far back as one wishes. Another advantage is that it facilitates tracing the impacts of any definition to even the most remote reaches of the definitional system.

In the context of modeling, tracing origins often arises in the course of developing documentation, designing explanations for others, and looking for reasons why something unexpected happened. Tracing impacts often arises in the course of trying to determine what might have to be changed as a consequence of some particular desired change or correction. In particular, the maintenance, evolutionary development, and integration of models become vastly easier when the interdependencies among model elements are formalized.

Structured modeling always requires full correlation of model elements. The construct which formalizes definitional dependencies is the *calling sequence* (to be defined in Section 2).

Acyclicity

The second desirable property is that definitional interdependencies should be acyclic when graphed as above, that is, the definitional dependency graph should exhibit no directed cycles.

This property avoids the obvious pitfalls associated with circular definitions. For instance, in the context of the first example, it would be disconcerting if, while following a thread of definitions back from the starting point "transitive relation," one came upon a prior concept that depended on the concept of a transitive relation!

Structured modeling requires that there be no directed cycles among definitional dependencies. It also takes pains to assure that acyclicity will propagate to higher order aspects of the formalism; Propositions 2 and 5 of Section 4 establish this.

Classification

The third desirable property of a definitional system is that each definition should fall into one of a small number of classes that arise from the definitional formalism itself rather than from any particular application of the formalism. For example, some definitions (such as SET in the first example) depend on no other, and hence are "primitive," while all other definitions are "nonprimitive." Also, some definitions (such as DAL_SUP in the second example) are value-bearing while others are not, and the value either can be user supplied or calculated in terms of prior

definitions (such as DAL_SUP and TOTAL_COST, respectively, in the second example).

There are at least two reasons for recognizing different classes of definitions. First, classification is a convenience for the user because different classes of definitions are sometimes used in different ways. For example, a definition that is classified as value-bearing with a noncalculated value presents a data acquisition responsibility for the user that other classes of definitions do not inflict. Second, the functionality of software aimed at supporting definitional systems depends on the classes of definitions to be supported. If value-bearing definitions with calculated values are to be supported, for example, a syntax for a suitable class of expressions is required along with a mechanism to evaluate the expressions found in such definitions.

Structured modeling recognizes five classes—it calls them *types*—of definitions: *primitive entity*, *compound entity*, *attribute*, *function*, and *test* (all to be defined in Section 2). These are the terminal nodes of the following tree:

Classification Tree	Type Name
Primitive	Primitive Entity
Nonprimitive	
Value-Bearing	
Calculated	
Logical-Valued	Test
Not Logical-Valued	Function
User-Supplied	Attribute
Not Value-Bearing	Compound Entity

The difference between function type and test type is minor: the latter has values restricted to True and False. There is no type that is both primitive and value-bearing because the existence of a value implies its association with something, and that something can be assumed without loss of generality to be not value-bearing.

Grouping

The fourth desirable property is that similar definitions should be grouped. Formally, this amounts to a partition in which all definitions in a given cell are "similar" enough for modeling purposes. See, for example, the ten groups of definitions in Figure 1: {1, 2}, {3, 4}, {5, 6, 7}, {8, 9, 10}, and so on.

The main advantage of grouping is that the working dimensionality of the definitional system can be reduced by suppressing (temporarily) the detailed definitions within the partition cells. This enables general structure to be seen without the burden of repetitive detail. This is particularly important when, as is usually the case in real applications, dimensions are large owing to the appearance of many definitions that are only slight variations of one another. Grouping makes it easier to see what is going on, and also points the way to efficient data structures for coping with masses of detail. For instance, in the context of the second example, there will still be 10 groups of definitions no matter what the number of plants or customers.

Moreover, grouping is an essential step toward the critical need to express entire classes of model instances; for example, the class of all Hitchcock-Koopmans transportation models or the class of all classical feedmix models. For analytical and communication purposes, we need to be able to express such model classes in a dimension-free and data-free way. Grouping enables this if one replaces an entire population of similar model elements by a single generic representative.

Structured modeling groups definitions through *generic structure* satisfying the criterion of *generic similarity*. Each group is called a *genus*. These are stepping stones toward the concept of a *model schema*, which turns out to be a powerful way to express most important classes of model instances. All four concepts will be defined in Section 2.

Hierarchy

Last, but by no means least, among the desirable properties of a definitional system is that the groups referred to above should be organized hierarchically, that is, the groups should be identified with the terminal nodes of a tree.

See, for example, the hierarchy of Figure 1: two groups are combined under the category "supply data," two under the category "customer data," three under the category "transportation data," and the last three under no category at all. Thus, the tree has a root node (corresponding to the whole model), three interior nodes (corresponding to the three categories), and ten terminal nodes (corresponding to the ten groups). This is a very simple hierarchical organization. In more complex systems of definitions there can be a much richer tree imposed on the groups of definitions. Major categories can have subcategories, which, in turn, can have subsubcategories, and so on. Figure 16 of Geoffrion (1987a) contains a more realistic example.

The main advantage of hierarchical structure is that it is accepted universally as a way to deal with complexity.

Structured modeling achieves hierarchical structure through the concept of *modular structure* with a *monotone ordering* (both concepts to be defined in Section 2).

This concludes our discussion of five desirable properties of definitional systems. In summary, they should be correlated, acyclic, classified, grouped, and hierarchical. The structured modeling framework for conceptual modeling possesses all five properties.

2. The Core Concepts of Structured Modeling

This section presents the core concepts of the structured modeling framework as a collection of formal definitions. They are organized under five headings: elemental structure, generic structure, modular structure, model instances, and model classes. Elemental structure formalizes the first three desirable properties of definitional systems discussed in Section 1, namely correlation, acyclicity, and classification. Generic structure formalizes the fourth property, grouping, and modular structure formalizes the property of hierarchical organization.

Commentary is interspersed to explain the intent of the definitions, but this commentary does not amend or augment the formal framework in any way. The second example of Section 1, which is expanded in the Appendix, illustrates all the definitions. We call it simply "the example."

The reader is forewarned that some of the definitions may seem a bit baroque by comparison with analogous definitions in ordinary mathematics. The reason, in most cases, is that we are trying to capture formally more semantic information than mathematical models usually do.

2.1. Elemental Structure

As explained in Section 1, a model is viewed as a definitional system in which each definition is called an **element**. There are five **element types**; these are presented in the first five definitions.

1. *A **primitive entity** element is undefined mathematically.*

This represents a primitive definition concerning a distinctly identifiable entity. The intended scope of an "entity" is close to that of the noun as a part of speech: person, place, thing, action, concept, event, quality, state, etc. Every model must have at least one primitive entity element. Each is introduced at the discretion and convenience of the modeler without, however, a presumption that it necessarily represents something irreducible or unanalyzable (as pointed out, for example, in Section 1.5 of Sowa (1984), such a presumption would raise serious philosophical questions). Elements 1–2 and 5–7 of the example (see Figure 1) would be modeled as primitive entities.

2. *A **compound entity** element is a segmented tuple of primitive entity elements and/or other compound entity elements.*

This represents a definition that references one or more entities already defined. The definition establishes a new entity that is dependent on other extant entities, often in the form of a relationship or association among them. A compound entity element can represent a member of a set or relation in the sense of discrete mathematics, and even the set or relation itself. Like the primitive entity element, it is not value-bearing. Elements 11–15 of the example would be modeled as compound entities.

3. *An **attribute** element is a segmented tuple of entity elements together with a particular **value** in some **range**.*

This allows a value-bearing property to be defined in connection with an entity or a collection of entities. "Value" is not necessarily numerical (the range set or space is arbitrary).

Elements 3–4, 8–10, and 16–25 of the example would be modeled as attributes. All have the real numbers as their range. Note that no value is specified for elements 16–20 in Figure 1.

Most of the data "coefficients" and decision "variables" of conventional models are represented in structured modeling as attribute elements. It is deliberate that structured modeling does not observe the customary distinction between "coefficients" and "variables." The reason is simply that this distinction tends to be unstable in some phases of most models' life cycle (e.g., in connection with statistical estimation, sensitivity analysis, and what-if analysis). However, Definition 16 does provide for some attribute elements to be classified as "variable" in the sense explained there.

4. *A **function** element is a segmented tuple of elements together with a **rule** that associates a particular **value** in some **range** to this tuple—more precisely, in the case of value-bearing elements, to the values of these elements provided these values fall within a prescribed **domain**.*

This is an extension of the attribute element concept in that function and test elements (see the next definition) can participate in the defining tuple, and the value can be conditional, that is, it can depend on the values of the value-bearing elements involved.

The domain and range are sets or spaces, with no presumption concerning their mathematical structure. Note that the notion of a "rule" used here is an abstract one; any notational system for structured modeling

(e.g., Geoffrion 1988) would have to supply a mathematical or other representation that is computable.

Element 26 of the example would be modeled as a function element. Its rule is a linear function whose domain is a pair of real vectors of identical dimension, and whose range is the real numbers.

5. *A **test** element is like a function element, except that it has a two-valued range {True, False}.*

Test elements facilitate defining the logical aspects of a model. One common use is to take account of the equality and inequality constraints often encountered in conventional models: a test element can be set up for each constraint to indicate whether or not it is satisfied.

Elements 27–31 of the example would be modeled as test elements. The rules for 27 and 28 correspond to linear inequalities viewed as logical expressions, while the rules for 29–31 correspond to linear equalities viewed as logical expressions. These five elements are not thought of as "constraints" in the usual sense of linear programming, but rather as indicators of whether or not the constraints hold for a given set of numerical transportation flows.

6. *The segmented tuple portion of an element is called its **calling sequence**. An element B is said to **call** another element A if A appears in B's calling sequence. A **calling sequence segment** has the obvious definition.*

The definitional interdependencies among the elements of a model are a central focus of structured modeling. The calling sequence is the principal abstraction of these dependencies. The segmentation of a calling sequence identifies the different roles played by different calls. Here the term "role" is used in an application context-dependent sense. Calls that play a similar role normally are put in the same segment.

Note that primitive entity elements, such as elements 1–2 and 5–7 in the example, do not have a calling sequence. See Figure 2 for a representative selection of calling sequences for the other elements. For example, element 27 calls element 3 in its first calling sequence segment, and calls elements 16, 17, 18 in its second segment. A comment is in order concerning element 21: why is its tuple not (1; 5)? The answer is that, although 21 does indeed refer to DAL and to PITTS, the purpose of 21 is to describe a property associated with the transportation link DAL_PITTS; element 11 is, therefore, the proper reference. Of course, 11 refers to 1 and 5, so 21 refers *indirectly* to 1 and 5.

Two conventions are necessary for a collection of elements to properly represent definitional dependencies:

a. First, if the semantic interpretation of a given element involves some other recognized element, then that other element should appear at least indirectly in the calling sequence of the given element. An "indirect" call is one where the indirectly called element is in the calling sequence of an element in the calling sequence, or in the calling sequence of an element in the calling sequence of an element in the calling sequence, etc.

b. Second, no calling sequence should contain an element that is patently irrelevant to the calling element's semantic interpretation.

These conventions are not imposed formally in the modeling framework; they pertain only to the intended manner of use.

7. *A collection of elements is **closed** if, for every element in the collection, all elements in the calling sequence of that element are also in the collection.*

Closure is essential to the integrity of definitional dependency information. Clearly, the example's collection of 31 elements is closed.

8. *A closed collection of elements is **acyclic** if there is no sequence {E1, . . . , En} such that E1 calls E2, . . . , En − 1 calls En, where n > 1 and En = E1.*

Our concern is exclusively with systems of elements whose definitional dependencies involve no circular references. This avoids problems of indeterminacy such as arise in circular systems of definitions. Acyclicity is essential in much of what follows.

While acyclicity clearly forbids any element to appear in its own calling sequence, it is neither possible nor desirable to exclude self-reference from the intended meaning of an element.

One way to see that the example's 31-element collection is acyclic is to look at Figure 1: the only references from one element B to another element A occur when A precedes B in the list.

9. *An **elemental structure** is a nonempty, finite, closed, acyclic collection of elements.*

This is the first major part of the definition of a structured model. Nonemptiness avoids trivialities. Finiteness avoids inessential technical difficulties, although an extension to models with an infinite number of elements probably could be made that would be satisfactory for most purposes. The rationale for closedness and acyclicity already has been given.

It is evident from the foregoing that the definitional system of Figure 1 can be formalized as an elemental structure.

2.2. Generic Structure

Generic structure enables similar elements to be grouped together.

In the context of model aggregation, a very important role of generic structure is to identify maximal sets of elements within which aggregation can take place.

10. *A **generic structure** is defined on an elemental structure as a collection of partitions, one for each of the five types of elements. The associated mutually disjoint and exhaustive element sets are called **genera** (the plural of **genus**).*

All the elements of a given genus are supposed to be alike except for the details, and it should be meaningful and natural to speak of a "generic" element. The sense in which elements are alike is difficult to formalize. Yet all large, realistic models seem to possess structure that invites the grouping of model components. One manifestation of this is the ubiquitous use of indices in conventional mathematical models. Indices greatly simplify notation by exploiting such structure.

Not every possible generic structure is useful. We have found it wise to limit consideration to generic structures satisfying the next property. It says, roughly, that the elements in a genus should not be too different in terms of which other elements they depend upon. This acts to limit the allowable coarseness of the generic structure.

11. *A generic structure satisfies the **generic similarity** property if the following is true for every genus (other than primitive entity genera): all elements in the genus have the same number of calling sequence segments, all calls in a given segment are to elements in the same genus, and all elements call the same genera in corresponding calling sequence segments.*

When this property holds, one can speak in the obvious sense of one genus "calling" another, and of a "genus' calling sequence."

A natural generic structure for the example is indicated by the element groupings of Figure 1. For example, the primitive entities are partitioned into plants and customers. That generic structure is shown in Figure 3, with each cell (genus) named for future reference.

Note that a partition can have but a single cell (e.g., the compound entity partition); and a cell can have just one element (e.g., $). Note also that a transportation model with 10,000 customers would still have only 10 genera.

Generic similarity can be checked by studying Figure 2, whose rows are grouped by genus. Consider genus T:SUP. Both of its elements have two calling sequence segments, the calls in each segment are always to the same genus, the first segment calls only SUP for both elements, and the second segment calls only FLOW for both elements. Thus, generic similarity holds for this genus. Similar checks work for the other genera.

2.3. Modular Structure

Modular structure is designed to recognize the hierarchical "conceptual structure" by which groups of genera take on higher semantic meaning. As elements are organized into genera, so may genera be organized into conceptual units (modules), which, in turn, may be organized into higher level conceptual units, etc., until the whole becomes the model itself as the root module. In this manner, models of arbitrary complexity can be rendered more manageable through meaningful hierarchical organization.

12. *A **modular structure** is defined on a generic structure as a rooted tree whose terminal nodes are in* 1:1 *correspondence with the genera. The nonterminal nodes are called **modules**. The **default modular structure** corresponds to the simplest possible such rooted tree, namely the one with only one module (the root).*

Since the default modular structure is always permitted, it is never limiting to assume that a given generic structure has a modular structure associated with it.

One possible modular structure for the example is indicated by the indentation used in Figure 1. It has three modules. That structure is shown explicitly in Figure 4, in which each module has been given a short name beginning with an ampersand.

The logic of this modular structure is that part of the model has to do with sources of supply, part with customers, and part with transportation. Each of these parts can be thought of as a distinct conceptual unit.

The default modular structure, by contrast, omits &SDATA, &CDATA, and &TDATA, and has &TRANS as the root and all 10 genera as terminal nodes.

The next definition orders the modular structure tree in a way that ties modular structure closely to the underlying calling relationships among the genera. The full significance of this ordering will not be

apparent until Definition 23 of Section 3 and Propositions 5–7 of Section 4.

13. *A **monotone ordering** of a modular structure defined on a generic structure satisfying similarity is specified by an **order** for each sibling set. These orders are extended in the usual way to obtain a strict partial order over all nodes except the root whereby any two nodes can be compared so long as neither lies on the rootpath of the other. This strict partial order is **monotone** in the following sense: if genus B calls genus A, then A comes before B in the strict partial order.*

The definition of the usual extension of sibling set orders referred to in the definition is as follows (e.g., p. 77 of Aho, Hopcroft, and Ullman 1983): if N1 and N2 are sibling nodes and N1 comes "before" N2, then N1 and all of its descendants come before N2 and all of its descendants.

Figure 5 shows one possible order for each sibling set for the modular structure of Figure 4. Figure 6 shows them as a directed graph along with their usual extension, which clearly is a strict partial order. It is easy to verify directly with the help of Figure 6 that this strict partial order is monotone.

To help the reader reconstruct the dashed lines in Figure 6, we remark that three of them are generated by taking (N1, N2) equal to (&SDATA, &CDATA) when extending the sibling orders: the arc from CUST to &SDATA stipulates that N1 comes before all descendants of N2, the arc from &CDATA to SUP stipulates that all descendants of N1 come before N2, and the arc from CUST to SUP stipulates that all descendants of N1 come before all descendants of N2. No other possible choices for N1 and N2 lead to arcs not already explicit or implicit (by transitive closure) in Figure 6.

Monotone-ordered modular structures may seem technically burdensome but, as explained near the end of Section 4, there is a very simple way to deal with them in practice.

2.4. Model Instances

All of the components necessary to define a structured model are now at hand.

14. *A **structured model** is an elemental structure together with a generic structure satisfying similarity and a monotone-ordered modular structure.*

Structured models are not always specified in complete detail. The next two definitions recognize this possibility.

15. *A **completely specified** structured model requires explicit enumeration and specification of: all elements in detail, including all calling sequences, attribute values, and function and test element rules (but not values); a generic structure satisfying similarity; and a monotone-ordered modular structure. Otherwise, a structured model is said to be **incompletely specified**. A completely specified elemental structure has the obvious definition.*

Unless otherwise indicated, the term "structured model" means a completely specified structured model.

16. *Attribute elements whose values are discretionary, and, hence, likely to change or to be placed under solver control, may be designated as **variable attributes**; these may be entire attribute genera or arbitrary subsets thereof. An **A-partially specified** structured model or elemental structure is one that is completely specified except for the values of its variable attribute elements.*

Variable attribute element values play much the same role as "variables" in many kinds of conventional models. Note that omitting values for attribute elements does not cast into doubt generic similarity, nor does it interfere with the specification of a monotone-ordered modular structure.

In practice, models are incompletely specified when first formulated. The degree of specification gradually increases as details are settled and data are developed until the degree of specification reaches a "final" level (usually either complete or A-partial) appropriate to the intended purpose. Usually the final level of specification is attained more than once; there may be a succession of models over time (as in data base applications) or a variety of model cases to be studied (as in many management science applications), but all of these are simply different specifications of the same basic model.

17. ***Evaluation** is the task of determining the values of the function and test elements of an elemental structure.*

Because of elemental structure acyclicity, evaluation always can be performed in a single pass based on the order resulting from a topological sort. See Proposition 4 in Section 4 and the subsequent discussion. Alternatively, one-pass evaluation can be guided by the modular outline defined in Section 3.

A structured model itself provides no means for performing evaluation. This is a task requiring some mechanism external to the model. Ideally, such a

mechanism should be an integral part of a structured modeling system.

Evaluation can turn out to be an ill posed task. To illustrate, an attribute value may not fall within the domain of definition of a function element's rule, or a rule may inadvertently involve division by zero. We wish to preclude this possibility by assuming hereafter the following property unless it is explicitly dropped.

18. *An **evaluateable** elemental structure is one whose specification, if not complete, can be completed so that all function and test element argument values are in their respective domains, and so that evaluation is a mathematically well posed task.*

The running example provides an illustration of a structured model: the 31 elements of Figure 1, together with the generic structure of Figure 3 and the monotone-ordered modular structure of Figures 4 and 5.

Elements 16–20 would all be treated as variable attribute elements in the context of the standard optimization problem usually associated with this model, namely finding values for all transportation flows so as to minimize TOTAL_COST subject to a positive outcome for all of the tests defined in elements 27–31.

This structured model is incompletely specified owing to the absence of values for the FLOW elements, i.e., the model is A-partially specified. To be completely specified, the missing attribute element values would have to be supplied.

Evaluation is straightforward: given nonnegative values for the FLOW elements, one may calculate the value of $ and the values of the elements of T:SUP and T:DEM in any order. Any such model is evaluateable because there is no way for the arguments of the rules of $, T:SUP, and T:DEM to violate the domains over which evaluation is a mathematically well posed task.

2.5. Model Classes

Practical and theoretical applications of models typically involve not a single model instance with particular data, but rather an entire class of model instances that are very similar in character. To put it another way, many uses of models require focusing on the general form of a model rather than on the data needed to specify a particular model instance. The notions of "general form" and "model class" are really one, and can be formalized as follows. This definition makes no presumption concerning evaluateability.

19. *A **model schema** is any prescribed class of structured models that satisfies isomorphism in this sense: given any two models in the class, their modules and genera can be placed in* 1 : 1 *correspondence in such a way that: a) node adjacency is preserved in the modular structure trees, b) corresponding genera have the same number of calling sequence segments and call corresponding genera from each segment, and c) corresponding genera are of identical element type.*

Perhaps the easiest way to specify a model schema is via an incompletely specified structured model, where the nature of the incompleteness is controlled carefully. For example, it is easy to see that any A-partially specified structured model can be viewed as a model schema. In most applications, however, the model schemata of greatest interest involve more than simply omitting some element values; for instance, it is common to leave indefinite even the number of elements in certain genera.

Note that the definition of a model schema has nothing to do with the ordering (monotone or otherwise) of a modular structure tree. The isomorphism requirements do not mention ordering because it is only the existence of a monotone ordering that is important in most model classes arising in practice, and not the particular ordering chosen for what may be subjective reasons.

We conclude this section with some examples of a model schema. One is the A-partially specified structured model mentioned at the end of Section 2.4. Any complete specification of attribute values yields a structured model in the class associated with the A-partially specified model. The technical requirements of Definition 19 are met trivially.

A slightly more general model schema is obtained by allowing the one just mentioned to have any monotone ordering whatever for the modular structure of Figure 4. (As will be seen later, there are exactly 24 monotone orderings.) Since the only change is to allow alternative orderings of the modular structure tree, we see from the comment following Definition 19 that the isomorphism requirements are still satisfied.

A still more general model schema is obtained by allowing any subset of all possible transportation links to exist. Note that this makes the element population of the LINK genus arbitrary. Whatever choice may be made for this population, the obvious compensating changes must be made in the genera which call LINK directly, and these changes induce others in the genera which call those genera. A moment's reflection shows that the isomorphism requirement of Definition 19 still holds.

Finally, a model schema more general than any of those above can be obtained by allowing not only the population of LINK to be arbitrary, but also the populations of PLANT and CUST. The obvious induced changes must be made in the genera that call PLANT or CUST directly or indirectly, in order for the result to be a genuine structured model. Although it is easy to see that the requirements of Definition 19 hold, it is not so easy to specify this model schema much more formally than the quite informal and ambiguous description just given. A suitable notational system is needed for structured models and model schemata. As mentioned elsewhere, that is the subject of Geoffrion (1988).

3. Associated Concepts and Constructs

The previous section presented the core concepts of structured modeling. Here we give definitions of several associated concepts and constructs that do not extend the modeling framework in a formal sense, but which facilitate working with it.

The first definition enables one to tailor model views that contain just the right level of detail for the intended target audience or conceptual analysis.

20. *A **view** of a modular structure is any subtree (i.e., a subgraph of the original rooted tree that is also a rooted tree) that keeps the original root and that does not separate original siblings (i.e., if two nodes have the same parent in the original tree, then they are either both in or both out of the subtree). A view always inherits an ordering from the modular structure if the modular structure has one. The **master view** corresponds to the subtree that is the original tree itself. Synonyms for "master view," "view," and "node of a subtree" are, respectively, **master conceptual structure**, **conceptual structure**, and **conceptual unit**.*

Every view other than the master view constitutes a simpler hierarchical conceptual structure than the master view. The simplification takes place by combining conceptual units from the bottom up in their natural hierarchical order.

It should be noted that the term "view" has another meaning in the literature on data base management systems (e.g., Ullman 1982).

Examples of Definition 20 and the next two definitions will be given after Definition 23.

The next definition is preparatory to the one following it.

21. *Associated with every view is a **genus partition** with one cell for every terminal node of the subtree; the genera in the cell corresponding to a given terminal node are the descendants of that node, that is, those genera whose rootpath in the original tree includes the given terminal node.*

The genus partition identifies the genera constituting the smallest conceptual units associated with a view. For the master view, the genus partition has exactly one genus per cell.

It is desirable for conceptual structures to inherit the lack of circular references in the underlying elemental structure. This is the concern of the next definition.

22. *Given a modular structure defined on a generic structure satisfying similarity, a view is **acyclicity-preserving** if the associated genus partition has the property that no subset of its cells can be arranged in a sequence $\{C1, \ldots, Cn\}$ such that some genus in $C1$ calls some genus in $C2$, ..., some genus in $Cn-1$ calls some genus in Cn, where $n > 1$ and $Cn = C1$. The modular structure itself is said to be acyclicity-preserving if every possible view is acyclicity-preserving.*

Next we present an equivalent representation of modular structure that is particularly useful. It displays all modules and genera along a single dimension in such a way that there are no forward references (see Proposition 6 in Section 4).

23. *The **modular outline** of an ordered modular structure (whether monotone or not) is the indented list representation corresponding to the preorder traversal. The outline for any view is defined similarly.*

Both of the concepts involved here, namely indented list representation of a tree and preorder traversal for ordered trees, are standard in computer science (e.g., Knuth 1973, pp. 309 and 334). In simple terms this means that all nodes of the modular structure tree are listed vertically, one to a line, with the indentation of each node proportional to the length of its rootpath; the root node is listed first, the nodes of each subtree are contiguous and begin with the root of the subtree, and siblings are always listed in their given order.

Other equivalent representations of the modular structure tree as an indented list are possible. For example, postorder or inorder traversal could be used. However, those representations appear less natural for present purposes than the one based on preorder traversal.

We now illustrate Definitions 20–23, beginning with the last of these.

The preorder traversal of the ordered modular structure of Figure 5 is: &TRANS, &SDATA, PLANT, SUP, &CDATA, CUST, DEM, &TDATA, LINK, FLOW, COST, $, T:SUP, T:DEM. The corresponding indented list representation is the modular outline shown in Figure 7. In all, there are 24 monotone orderings of the modular structure of Figure 4. Six can be generated from Figure 7 by permuting the last three genera. These six can be doubled by permuting the &SDATA and &CDATA modules. Twelve more can be generated by reversing the positions of FLOW and COST.

If Figure 7 is the master view, then eight other views are possible, as shown in Figure 8. They all inherit a monotone ordering. To illustrate an interpretation, View 2 chooses not to preserve the details of LINK, FLOW, and COST in support of the conceptual unit &TDATA. Its subtree is shown in Figure 9.

The genus partition associated with View 2 (Figure 8) is:

{PLANT} {SUP} {CUST} {DEM}
{LINK, FLOW, COST} {$} {T:SUP} {T:DEM}.

For View 5 it is

{PLANT} {SUP} {CUST, DEM}
{LINK, FLOW, COST} {$} {T:SUP} {T:DEM}.

It is easy to see that the master view is acyclicity-preserving by looking at the modular outline. One genus can call another genus only if the second genus is higher on the list; this obviously precludes a calling cycle. A similar argument suffices to show that all other views are also acyclicity-preserving. Since all views are acyclicity-preserving, the modular structure itself (Figure 4) is acyclicity-preserving.

Of course, in practice one does not have to enumerate all possible views in order to prove that a modular structure is acyclicity-preserving. Instead, one devises a monotone ordering and applies Proposition 7 of Section 4.

The remaining definitions focus on graphs and matrices that, in the main, represent the definitional dependencies in different ways.

24. *The **element graph** of an elemental structure is an attributed directed graph with a node for every element and an arc from element B to element A if element A calls element B (more precisely, there is an arc for every such call). Every node has an attribute denoting its type (primitive entity, compound entity, attribute, function, or test). Every non-entity node has another attribute giving its value, every attribute node has another attribute giving its range, and every function and test node has an attribute giving its rule. Every arc has two attributes; the first identifies the calling sequence segment to which it corresponds, and the second identifies its position within the segment.*

The element graph vividly portrays the cross-references among elements. More than that, it is a precisely equivalent representation of an elemental structure. Since the collection of elements is acyclic, it follows, of course, that the element graph is acyclic in the usual graph theoretic sense.

Figure 10 shows the element graph for our example. Notice that it is acyclic. The diagram in Section 1.1 also is an element graph.

25. *The **genus graph** of a generic structure satisfying similarity is a directed graph with a node for every genus and an arc for every segment of every genus (primitive entity genera excepted) directed from the genus being called to the calling genus.*

The genus graph portrays cross-references among genera. It is a far more manageable portrayal than the element graph for most purposes. Figure 11 shows the genus graph for the example.

26. *The **module graph** corresponding to a view of a modular structure is a directed graph with a node for every cell of the associated genus partition and an arc from cell A to cell B (where A and B are distinct) if and only if some genus of cell B calls some genus of cell A.*

The module graph portrays cross-references among the smallest conceptual units of a view. It takes the place of the genus graph for presentations of a structured model based on views other than the one provided by the default modular structure. The module graph corresponding to the master view of any modular structure coincides with the genus graph with multiple arcs removed (i.e., at most one arc is allowed between each pair of nodes). Figure 12 shows a module graph.

Element, genus, and module graphs are related to one another through the notion of **condensation**. If G is a directed graph and P is a partition of the nodes of G, then the condensation of G with respect to P is a graph having a node for every cell of P and an arc from cell i to distinct cell j if and only if G has an arc from some node of cell i to some node of cell j (see, e.g., Harary, Norman, and Cartwright 1965). It is evident that a genus graph with multiple arcs eliminated is always the condensation of an element graph, without attributes, with respect to the generic structure. Similarly, a module graph is always the condensation of a genus graph with respect to the genus

partition associated with the corresponding view. Clearly, Figure 11 is a condensation of Figure 10, and Figure 12 is a condensation of Figure 11.

The last two definitions are based on standard concepts from graph theory.

27. *The **adjacency matrix** corresponding to an element graph, a genus graph, or a module graph is a square matrix with a row and column for every node of the graph and a "1" in row i and column j if there is an arc from the node of row i to the node of column j; all other entries are zero. (Informally: "column calls row.")*

An adjacency matrix is an alternative representation of a graph that is easier to produce typographically; however, it ignores the attributes of an element graph and the possibility of multiple arcs for element and genus graphs. It is well suited to tracing the references to or from any given node. Choosing (if possible) the row/column order so that a triangular matrix results usually is advantageous. When a monotone ordering of the modular structure is available, using the corresponding preorder traversal sequence necessarily results in an upper triangular matrix for the element graph, genus graph, and all module graphs.

The adjacency matrix for the module graph corresponding to a given view is easy to determine once the adjacency matrix for the genus graph is available. Any view can be specified by listing the modules that are terminal nodes in the view's subtree. The associated genus partition is easy to identify from this list of modules thanks to the indentation structure of the modular outline. The genera of each cell of the genus partition are all contiguous in the modular outline and will correspond, therefore, to adjacent rows and columns of the genus adjacency matrix (assuming that the preorder traversal sequence corresponding to the ordering is used as usual to establish row/column order). One sees easily that the rows and columns of the genus graph's adjacency matrix can be aggregated by Boolean summation to obtain the desired module graph adjacency matrix (a final step is necessary: zero out any 1's on the diagonal).

The example's element graph adjacency matrix is as shown in Figure 13 when the rows and columns are ordered in the same sequence as the original elements in Figure 1. Each column indicates which elements the column element calls, and each row indicates which elements call the row element. Row and column spaces have been introduced to preserve the genus-grouping of the original elements. Observe that this matrix is upper triangular; this verifies acyclicity.

Figure 14 gives the genus graph's adjacency matrix. Figure 15 gives the adjacency matrix for the module graph corresponding to the second view; it can be obtained either by inspection of Figure 12 or from the Boolean summation procedure mentioned before.

28. *The **reachability matrix** corresponding to an element graph, a genus graph, or a module graph is a square matrix with a row and column for every node of the graph and a "1" in row i and column j if there is a directed path from the node of row i to the node of column j; all other entries are zero. By convention, diagonal entries are taken to be unity.*

The reachability matrix of an element graph can be read two ways: 1) by columns to determine all the elements referenced directly or indirectly by the column element, and 2) by rows to determine all the elements that directly or indirectly reference the row element. Similar statements can be made about the reachability matrices of genus and module graphs.

A reachability matrix is easy to calculate from the corresponding adjacency matrix using Warshall's procedure (e.g., page 132 of Berztiss 1975) or variants that are still more efficient.

The example's element graph reachability matrix is shown in Figure 16 when the rows and columns are ordered in the same sequence as for Figure 13. Each column tells which elements are called directly or indirectly by the column element; for instance, element 21 directly or indirectly calls elements 1, 5, and 11. Each row tells which elements directly or indirectly call the row element; for instance, element 1 (representing the plant in Dallas) is called directly or indirectly by 15 other elements.

Figure 17 gives the genus graph reachability matrix, and Figure 18 gives the module graph reachability matrix corresponding to Figure 12.

4. Some Theoretical Results

This section develops some elementary but useful results about the definitions of the previous two sections. The first result confirms a rather obvious fact.

Proposition 1. *In an elemental structure with a generic structure satisfying similarity, no element calls another element in the same genus.*

Proof. Suppose, to the contrary, that some element in genus G calls another element in G. Then, by generic similarity, every element in G calls another element in G. By the finiteness of the number of elements of G, this implies that there is a cycle

of elemental calls among the elements of G, which violates elemental structure acyclicity. Thus, the supposition must be false.

In structured modeling, elemental structure is always acyclic. This acyclicity propagates to generic structure so long as generic similarity holds, and to modular structure so long as a monotone ordering is used. The next result and Proposition 5 establish this claim.

Proposition 2. *Genus graphs are always acyclic.*

Proof. Recall that a genus graph is defined only when there is a generic structure satisfying generic similarity, and that a generic structure, in turn, presumes an elemental structure. Suppose that the genus graph contains a directed cycle, say $\{G1, G2, \ldots, Gn-1, G1\}$ where $G1$ calls $G2$, which calls $G3, \ldots$, which calls $Gn = G1$. We wish to show that an elemental cycle must then exist, for this would contradict the acyclicity of the elemental structure and, thereby, demonstrate that the supposition must be false. Take any element in $G1$ and trace a directed chain through the elements in $G2, G3, \ldots$, and so on, back to $Gn = G1$ again. This may be done by supposition and the generic similarity property. If the terminal element in $G1$ is the starting one, an elemental cycle has been found. If the terminal element is not the starting one, then by the similarity property, another directed elemental chain can be constructed that starts at the terminal element just found and ends in $G1$. If any element of the new chain coincides with any element of the earlier chain, then an elemental cycle has been found. If not, the chain can be continued in a like manner. Eventually, since there is but a finite number of elements, some element must be encountered again, thereby establishing an elemental cycle.

One can view Proposition 2 as demonstrating that the calls among genera induce a strict partial order over all genera. Thus, the last sentence of Definition 13 can be rephrased informally as: "This partial order is **monotone** in that it is consistent with the partial order induced by calls among genera."

The genus graph of Figure 11 is acyclic, in accordance with Proposition 2.

A well known and important property of acyclic directed graphs is that their nodes can be classified into **ranks** such that nodes of rank r ($r > 1$) have incoming arcs only from nodes of lower rank including at least one node of rank $r - 1$. The classification is unique.

Element and genus graphs can be ranked, for both are acyclic. The next result shows that these rankings are consistent when viewed in terms of elements.

Proposition 3. *Consider an elemental structure together with a generic structure satisfying similarity. The rank of any element based on the element graph is identical to the rank of the element's genus based on the genus graph.*

Proof. It suffices to show that the genus-based ranking of all elements coincides with the element-based ranking for all $r > 0$, where r denotes genus rank. Clearly this is true for $r = 1$, for which the genera simply partition the primitive entity elements (which all have element rank 1). Suppose it is true for all r less than or equal to R, where R is a fixed positive integer. To see that it is true for $R + 1$, consider any element of genus rank $R + 1$. This element must call some element whose genus rank is R, and all of its other calls must be to elements of genus rank R or less. Hence, this element must call some element whose element rank is R, and all of its other calls must be to elements of element rank R or less. It must, therefore, be of element rank $R + 1$.

Corollary 1. *Consider an elemental structure together with a generic structure satisfying similarity. For every genus, all of its elements must be of the same type and elemental rank.*

Proof. Consider any two elements in any given genus, say $e1$ and $e2$. The definition of generic structure guarantees they are of the same type. Proposition 3 guarantees they both have the same elemental rank as genus rank. Since both elements are in the same genus, it follows that both elemental ranks must be the same.

It is convenient to record here the rankability of genus graphs.

Proposition 4. *When generic structure satisfies similarity, all genera can be classified uniquely into ranks in such a manner that*

(i) *rank one genera call no other genera, and*
(ii) $r > 1$, *every genus of rank r calls at least one genus of rank* $r - 1$, *possibly other genera of rank less than* $r - 1$, *but no genus of rank greater than* $r - 1$.

Classifying genera by rank is a simple matter. Rank 1 consists of all primitive entity genera. Rank 2 consists of all genera which call only primitive entity genera. Rank 3 consists of all remaining genera which

only call genera of ranks 1 and 2. In general, rank r consists of all remaining genera which call only previously classified genera.

Evaluation can be done in one pass rank by rank, in ascending order. This is a fact of considerable computational significance.

Classifying nodes by rank is really just a kind of topological sort, namely the one which uses the fewest possible distinct node labels. (A **topological sort** of an acyclic directed graph produces a node sequence such that if there is an arc from node A to node B, then node A comes before node B in the sequence; see, e.g., Knuth 1973, p. 258ff.) While any topological sort enables evaluation to be done sequentially in a single pass, this particular sort maximizes the opportunities for parallel computation (an opportunity that future computers are likely to be able to exploit).

These ideas can be illustrated with the example of the Appendix. The genus ranks are easy to see from the genus graph (Figure 11). The topological sort first identifies PLANT and CUST, as these have no incoming arcs. After "dropping" the outgoing arcs of these two genera, the topological sort then identifies SUP, LINK, and DEM as having no incoming arcs. And so on. The result is as follows.

Rank	*Genera*
1	PLANT, CUST
2	SUP, LINK, DEM
3	COST, FLOW
4	$, T:SUP, T:DEM

We turn now to the topic of module graph acyclicity. Recall Definitions 22 and 26. Consider any view of a modular structure defined on a generic structure satisfying similarity. Clearly the corresponding *module graph will be acyclic if and only if the view is acyclicity-preserving.* The next result shows that acyclicity always obtains for a structured model. Thus, the acyclicity of Figure 12 is no accident.

Proposition 5. *Module graphs for structured models are always acyclic.*

Proof. Consider any view of any structured model. Suppose that this view is not acyclicity-preserving. Then there is a sequence of cells $\{C1, \ldots, Cn\}$ of the associated genus partition such that some genus in $C1$ calls some genus in $C2, \ldots,$ some genus in $Cn-1$ calls some genus in Cn, where $n > 1$ and $Cn = C1$. This cycle has an image in the modular outline. Because the genera of each cell of the genus partition occupy consecutive positions in the list, the image of the cycle clearly is inconsistent with the no-forward-reference property of the outline. Thus, the supposition must be erroneous and the desired result is at hand.

The property used at the end of the last proof deserves to be formalized.

Proposition 6. *If genus B calls genus A in a structured model, then A comes before B in the modular outline.*

Proof. Consider any structured model. Suppose that genus B calls genus A. Then, by the definition of monotone ordering, the node corresponding to genus A must come "before" the node of genus B in the usual partial order extension of the sibling orders. But it is well known that a preorder traversal of the nodes of an ordered rooted tree preserves the usual partial order extension of the sibling nodes, that is, if node $N1$ comes before node $N2$ in the usual partial order extension, then $N1$ comes before $N2$ in the preorder traversal sequence. Thus, genus A comes before genus B in the indented list representation corresponding to the preorder traversal.

The acyclicity of Figure 12 is consistent with Proposition 5, and Figure 7 is consistent with Proposition 6.

Consider an elemental structure, together with a generic structure satisfying similarity and a modular structure. It is natural to wonder about the existence of a monotone ordering and how to construct one, for without a monotone ordering there can be no structured model.

Existence is in doubt because it is easy to find situations in which no monotone order exists. One can verify that such a situation is the following: let there be three genera, with genus C calling genus B and genus B calling genus A; and let A and C (but not A and B and C) be siblings in the modular structure.

The following result gives two characterizations of when a monotone ordering exists. One is primarily of theoretical interest, while the other is simple and constructive.

Proposition 7. *Consider an elemental structure, together with a generic structure satisfying similarity and a modular structure. The following are equivalent:*

(i) *a monotone ordering exists;*
(ii) *the modular structure is acyclicity-preserving;*
(iii) *for every sibling set of the modular structure tree, no subset of the siblings can be arranged in a sequence $\{S1, \ldots, Sn\}$ such that some genus*

descendant of S1 calls some genus descendant of S2, . . . , some genus descendant of Sn−1 calls some genus descendant of Sn, where n > 1 and Sn = S1.

Proof. To see that i⇒ii, consider any view. As observed just prior to Proposition 5, it is acyclicity-preserving if its module graph is acyclic. Acyclicity of the module graph follows from Proposition 5, which applies because of i. To see that ii⇒iii, consider any sibling set of the modular structure tree and any view whose terminal nodes include this sibling set. Since the modular structure is acyclicity-preserving, this view is also acyclicity-preserving, which implies iii by definition because the siblings are among the view's terminal nodes. Finally, to see that iii⇒i one observes that iii implies a topological sort is possible for each of the sibling sets of the modular structure tree; that is, it is possible to arrange each sibling set in a sequence $\{S1, S2, \ldots\}$ such that some genus descendant of Si calls some genus descendant of Sj (i and j distinct) only if $j < i$. Obviously, this constructs a monotone ordering.

The constructive procedure used in the last part of the proof is important because it gives a simple way to construct monotone orderings when they exist for a given modular structure: just attempt a topological sort of each sibling set. If this succeeds for all sibling sets, the resulting **topological labeling** yields one or more monotone orderings (but not necessarily all possible monotone orderings). If the attempt fails for some sibling set, then no monotone ordering exists. This procedure can be implemented efficiently using only the adjacency matrix of the genus graph and the modular outline (the order need not be monotone).

We now apply this procedure to the example. The construction considers the sibling sets one by one. These sets are

1. &TRANS
2. &SDATA, &CDATA, &TDATA, $, T:SUP, T:DEM
3. PLANT, SUP
4. CUST, DEM
5. LINK, FLOW, COST.

The topological sort of the first sibling set is trivial, and yields the label 1 for &TRANS.

A topological sort of the second sibling set yields

1 &SDATA, &CDATA
2 &TDATA
3 $, T:SUP, T:DEM

because the genera of &SDATA and &CDATA call no other genera outside their modules; the genera of &TDATA call only genera in &SDATA and &CDATA; and the genera of $, T:SUP, and T:DEM call genera in &TDATA.

A topological sort of the third sibling set yields

1 PLANT
2 SUP.

Similarly, a topological sort of the fourth sibling set yields

1 CUST
2 DEM

and of the fifth sibling set yields

1 LINK
2 COST, FLOW.

The results of these topological sorts are summarized in Figure 19. This topological labeling makes it clear that a monotone order will result if

- &SDATA and &CDATA precede &TDATA
- &TDATA precedes $, T:SUP, and T:DEM
- PLANT precedes SUP
- CUST precedes DEM
- LINK precedes FLOW and COST.

However, three multiple choice options remain:

- pick an order for &SDATA and &CDATA
- pick an order for $, T:SUP, and T:DEM
- pick an order for FLOW and COST.

These options generate all $2 \times 6 \times 2 = 24$ monotone orderings.

Although explicit topological labeling can be useful, experience shows that monotone orderings of sensible modular structures are not difficult to devise. Suppose that one has an elemental structure together with a generic structure satisfying similarity. It is no problem at all to write down a plausible modular structure tree that is ordered, and this can always be done in outline form—that is, using an indented list representation based on the preorder traversal sequence. If there are no forward references among general in the modular outline, then the order used for the modular structure tree is monotone. The converse is also true. Thus, *the practical task of designing a monotone-ordered modular structure can be viewed as an exercise in sensibly arranging all genera in outline form in such a way that there are no forward references among them.*

The final result is a simple one but, in view of Proposition 7, it does settle (in the affirmative) the question of the existence of a monotone-ordered

modular structure for any given elemental structure together with a generic structure satisfying similarity.

Proposition 8. *For any elemental structure together with a generic structure satisfying similarity, the default modular structure is necessarily acyclicity-preserving.*

Proof. Only one view is possible for the default modular structure. The definition of "acyclicity-preserving" reduces, in this case, to the requirement that the genus graph must be acyclic. It is, by Proposition 2.

5. Conclusion

The basic concepts of structured modeling, introduced informally and used in Geoffrion (1987a), have all been developed here formally after being motivated in terms of definitional systems. In addition, we have obtained some associated theoretical results for use in future work.

Various extensions of the formal modeling framework given here may be achievable. Some possibilities are suggested in the last section of Geoffrion (1987a).

This paper is part of a series. Readers who find this conceptual modeling framework of interest may wish to examine the complete notational system in Geoffrion (1988) for expressing structured models and model schemata. That notational system is used by a research prototype implementation that will be the subject of another paper.

APPENDIX

Exhibits for an Illustrative Transportation Model

A 2 × 3 Hitchcock-Koopmans transportation model is represented by the definitional system shown in Figure 1, which uses the ad hoc notational conventions explained in Section 1.1. In addition, grouping and indentation are used for organizational purposes. The identical model appears in Geoffrion (1987a).

Supply Data

1. There is a plant in Dallas called DAL.
2. There is a plant in Chicago called CHI.
3. DAL has a supply capacity (DAL_SUP) of 20,000 tons.
4. CHI has a supply capacity (CHI_SUP) of 42,000 tons.

Customer Data

5. There is a customer in Pittsburgh called PITTS.
6. There is a customer in Atlanta called ATL.
7. There is a customer in Cleveland called CLEV.
8. PITTS has a demand (PITTS_DEM) of 25,000 tons.
9. ATL has a demand (ATL_DEM) of 15,000 tons.
10. CLEV has a demand (CLEV_DEM) of 22,000 tons.

Transportation Data

11. There is a transportation link (DAL_PITTS) from DAL to PITTS.
12. There is a transportation link (DAL_ATL) from DAL to ATL.
13. There is a transportation link (DAL_CLEV) from DAL to CLEV.
14. There is a transportation link (CHI_PITTS) from CHI to PITTS.
15. There is a transportation link (CHI_CLEV) from CHI to CLEV.
16. There can be a nonnegative transportation flow in tons (DAL_PITTS_FLOW) over DAL_PITTS.
17. There can be a nonnegative transportation flow in tons (DAL_ATL_FLOW) over DAL_ATL.
18. There can be a nonnegative transportation flow in tons (DAL_CLEV_FLOW) over DAL_CLEV.
19. There can be a nonnegative transportation flow in tons (CHI_PITTS_FLOW) over CHI_PITTS.
20. There can be a nonnegative transportation flow in tons (CHI_CLEV_FLOW) over CHI_CLEV.
21. The DAL_PITTS cost rate is $23.50 per ton (DAL_PITTS_COST).
22. The DAL_ATL cost rate is $17.75 per ton (DAL_ATL_COST).
23. The DAL_CLEV cost rate is $32.45 per ton (DAL_CLEV_COST).
24. The CHI_PITTS cost rate is $7.60 per ton (CHI_PITTS_COST).
25. The CHI_CLEV cost rate is $25.75 per ton (CHI_CLEV_COST).
26. There is a TOTAL_COST associated with all transportation flows equal to DAL_PITTS_FLOW × DAL_PITTS_COST + . . . + CHI_CLEV_FLOW × CHI_CLEV_COST.
27. The DALLAS_SUPPLY_TEST determines whether the total transportation flow leaving Dallas, namely DAL_PITTS_FLOW + DAL_ATL_FLOW + DAL_CLEV_FLOW, is less than or equal to DAL_SUP.
28. The CHICAGO_SUPPLY_TEST determines whether the total transportation flow leaving Chicago, namely CHI_PITTS_FLOW + CHI_CLEV_FLOW, is less than or equal to CHI_SUP.
29. The PITTSBURGH_DEMAND_TEST determines whether the total transportation flow arriving at Pittsburgh, namely DAL_PITTS_FLOW + CHI_PITTS_FLOW, exactly equals PITTS_DEM.
30. The ATLANTA_DEMAND_TEST determines whether the total transportation flow arriving at Atlanta, namely DAL_ATL_FLOW, exactly equals ATL_DEM.
31. The CLEVELAND_DEMAND_TEST determines whether the total transportation flow arriving at Cleveland, namely DAL_CLEV_FLOW + CHI_CLEV_FLOW, exactly equals CLEV_DEM.

Figure 1. Definitions for a simple transportation model.

Element	Calling Sequence
3	(1)
4	(2)
8	(5)
etc.	etc.
11	(1; 5)
etc.	etc.
16	(11)
etc.	etc.
21	(11)
etc.	etc.
26	(16, 17, 18, 19, 20; 21, 22, 23, 24, 25)
27	(3; 16, 17, 18)
28	(4; 19, 20)
29	(8; 16, 19)
etc.	etc.

Figure 2. Elements and their calling sequences (with semicolons separating segments).

		Genus
Partition of primitive entities:	{1, 2}	PLANT
	{5, 6, 7}	CUST
Partition of compound entities:	{11, 12, 13, 14, 15}	LINK
Partition of attribute elements:	{3, 4}	SUP
	{8, 9, 10}	DEM
	{16, 17, 18, 19, 20}	FLOW
	{21, 22, 23, 24, 25}	COST
Partition of function elements:	{26}	$
Partition of test elements:	{27, 28}	T:SUP
	{29, 30, 31}	T:DEM

Figure 3. Generic structure (satisfying generic similarity).

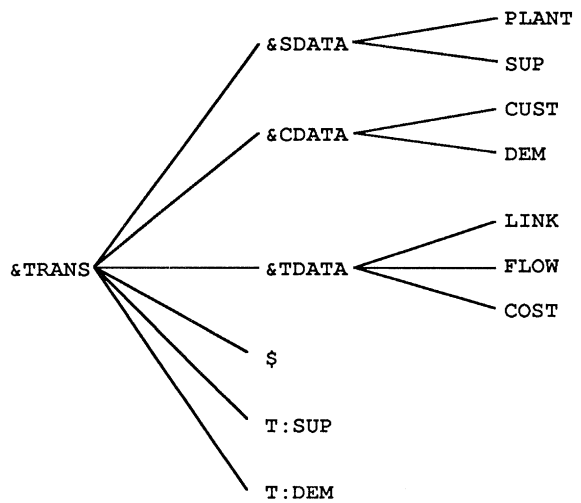

Figure 4. Modular structure.

Module	Sibling Sequence of Module Children
&TRANS	&SDATA, &CDATA, &TDATA, $, T:SUP, T:DEM
&SDATA	PLANT, SUP
&CDATA	CUST, DEM
&TDATA	LINK, FLOW, COST

Figure 5. Sibling orders for the modular structure of Figure 4.

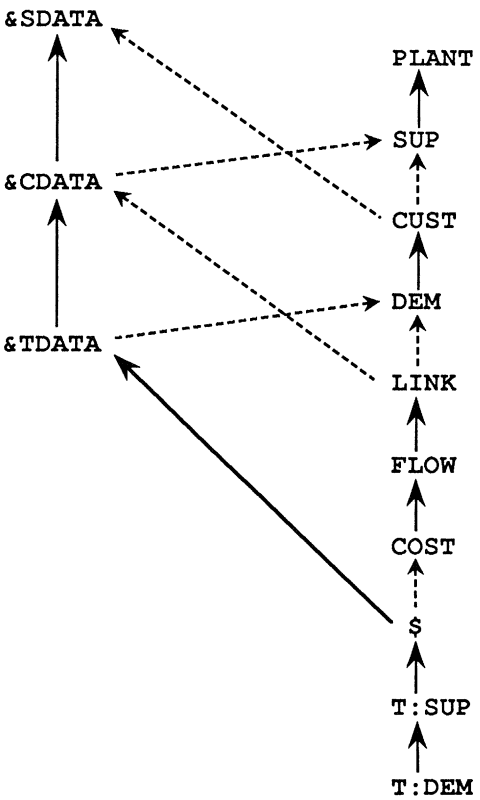

Figure 6. Sibling orders (solid lines only) and the extended order (solid and dashed lines) based on Figure 5.

```
&TRANS
  &SDATA
    PLANT
    SUP
  &CDATA
    CUST
    DEM
  &TDATA
    LINK
    FLOW
    COST
  $
  T:SUP
  T:DEM
```

Figure 7. Modular outline corresponding to Figure 5.

View 2	View 3	View 4	View 5
&TRANS	&TRANS	&TRANS	&TRANS
&SDATA	&SDATA	&SDATA	&SDATA
PLANT	PLANT	&CDATA	PLANT
SUP	SUP	CUST	SUP
&CDATA	&CDATA	DEM	&CDATA
CUST	&TDATA	&TDATA	&TDATA
DEM	LINK	LINK	$
&TDATA	FLOW	FLOW	T:SUP
$	COST	COST	T:DEM
T:SUP	$	$	
T:DEM	T:SUP	T:SUP	
	T:DEM	T:DEM	

View 6	View 7	View 8	View 9
&TRANS	&TRANS	&TRANS	&TRANS
&SDATA	&SDATA	&SDATA	
&CDATA	&CDATA	&CDATA	
CUST	&TDATA	&TDATA	
DEM	LINK	$	
&TDATA	FLOW	T:SUP	
$	COST	T:DEM	
T:SUP	$		
T:DEM	T:SUP		
	T:DEM		

Figure 8. The possible views associated with Figure 7 (excluding the master view).

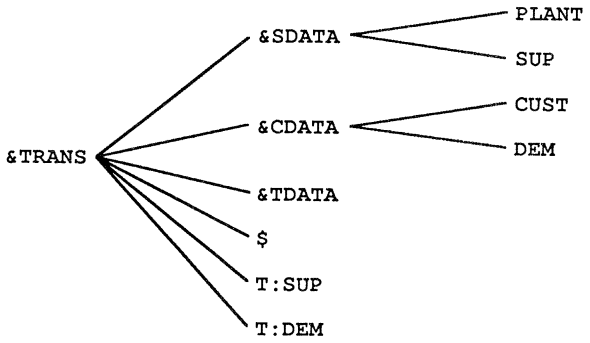

Figure 9. Modular subtree for View 2.

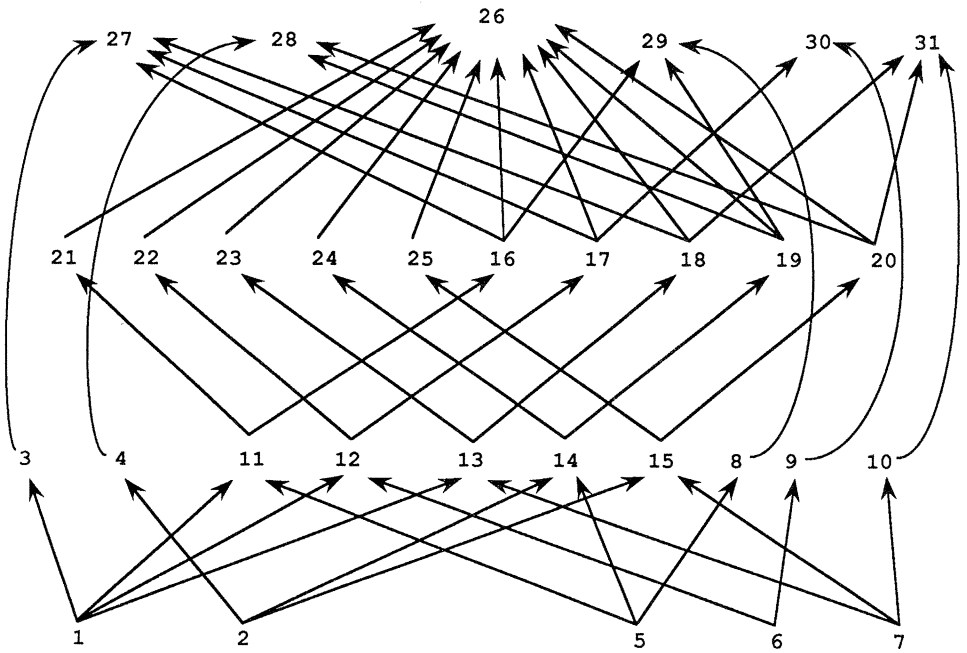

Figure 10. Element graph without attributes.

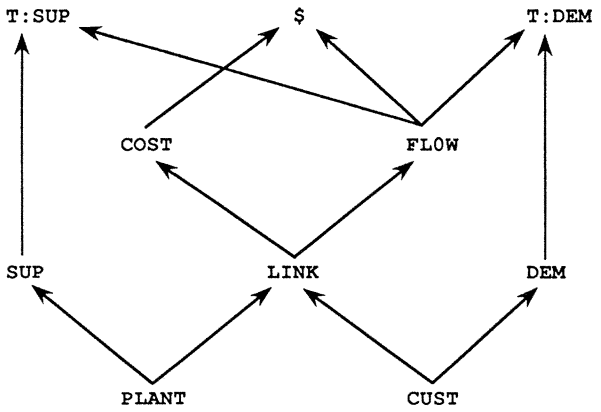

Figure 11. Genus graph.

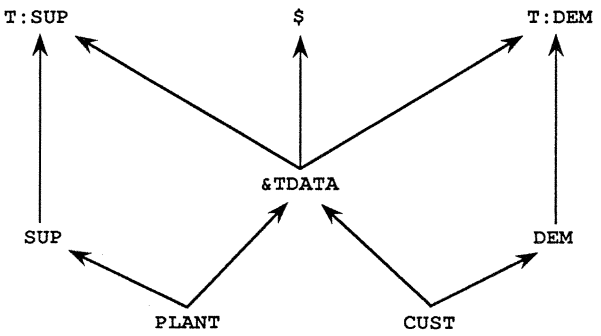

Figure 12. Modular graph for View 2 of Figure 8.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	.	.	1	.	.	.	.	.	.	.	1	1	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	.	1	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
5	.	.	.	.	.	.	.	1	.	.	1	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.	.	1	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	1	.	.	1	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.
9	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.
10	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
11	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.
12	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.
13	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	.	.	.	.	.
14	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	.	.	.	.
15	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	.	.	.
16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	1	.	1	.	.	.
17	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	1	.	.	1	.	.
18	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	1	.
19	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	1	1	.	.	.
20	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	1	.	.	1	.
21	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.
22	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.
23	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.
25	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.
26	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
27	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
28	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
29	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
31	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figure 13. Element graph adjacency matrix associated with Figure 10 ("·" replaces "0").

	PLANT	SUP	CUST	DEM	LINK	FLOW	COST	$	T:SUP	T:DEM
PLANT	·	1	·	·	1	·	·	·	·	·
SUP	·	·	·	·	·	·	·	·	1	·
CUST	·	·	·	1	1	·	·	·	·	·
DEM	·	·	·	·	·	·	·	·	·	1
LINK	·	·	·	·	·	1	1	·	·	·
FLOW	·	·	·	·	·	·	·	1	1	1
COST	·	·	·	·	·	·	·	1	·	·
$	·	·	·	·	·	·	·	·	·	·
T:SUP	·	·	·	·	·	·	·	·	·	·
T:DEM	·	·	·	·	·	·	·	·	·	·

Figure 14. Genus graph adjacency matrix associated with Figure 11.

	PLANT	SUP	CUST	DEM	&TDATA	$	T:SUP	T:DEM
PLANT	·	1	·	·	1	·	·	·
SUP	·	·	·	·	·	·	1	·
CUST	·	·	·	1	1	·	·	·
DEM	·	·	·	·	·	·	·	1
&TDATA	·	·	·	·	·	1	1	1
$	·	·	·	·	·	·	·	·
T:SUP	·	·	·	·	·	·	·	·
T:DEM	·	·	·	·	·	·	·	·

Figure 15. Module graph adjacency matrix associated with Figure 12.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	.	1	.	.	.	.	.	.	.	1	1	1	.	.	1	1	1	.	.	1	1	1	.	.	1	1	.	1	1	1
2	.	1	.	1	.	.	.	.	.	.	.	.	.	1	1	.	.	.	1	1	.	.	.	1	1	1	.	1	1	.	1	1
3	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
4	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.
5	.	.	.	.	1	.	.	1	.	.	1	.	.	1	.	1	.	.	1	.	1	.	.	1	.	1	1	1	1	.	.
6	.	.	.	.	.	1	.	.	1	.	.	1	.	.	.	.	1	.	.	.	.	1	.	.	.	1	1	.	.	1	.
7	.	.	.	.	.	.	1	.	.	1	.	.	1	.	1	.	.	1	.	1	.	.	1	.	1	1	1	1	.	.	1
8	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.
9	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
10	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1
11	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	1	.	.	.	.	1	1	.	1	.	.
12	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	1	.	.	.	1	1	.	.	1	.
13	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	1	.	.	1	1	.	.	.	1
14	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	1	.	1	.	1	1	.	.
15	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	1	1	.	1	.	.	1
16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	1	1	.	1	.	.
17	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	1	.	.	1	.
18	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	1	.	.	.	1
19	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	.	1	1	.	.
20	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	1	.	1	.	.	1
21	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	.
22	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	1	.	.	.	.	.
23	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	1	.	.	.	.	.
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	1	.	.	.	.	.
25	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.
26	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.
27	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
28	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.
29	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.
30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
31	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1

Figure 16. Element graph reachability matrix associated with Figure 13 ("·" replaces "0").

	PLANT	SUP	CUST	DEM	LINK	FLOW	COST	$	T:SUP	T:DEM
PLANT	1	1	·	·	1	1	1	1	1	1
SUP	·	1	·	·	·	·	·	·	1	·
CUST	·	·	1	1	1	1	1	1	1	1
DEM	·	·	·	1	·	·	·	·	·	1
LINK	·	·	·	·	1	1	1	1	1	1
FLOW	·	·	·	·	·	1	·	1	1	1
COST	·	·	·	·	·	·	1	1	·	·
$	·	·	·	·	·	·	·	1	·	·
T:SUP	·	·	·	·	·	·	·	·	1	·
T:DEM	·	·	·	·	·	·	·	·	·	1

Figure 17. Genus graph reachability matrix associated with Figure 11.

	PLANT	SUP	CUST	DEM	&TDATA	$	T:SUP	T:DEM
PLANT	1	1	·	·	1	1	1	1
SUP	·	1	·	·	·	·	1	·
CUST	·	·	1	1	1	1	1	1
DEM	·	·	·	1	·	·	·	1
&TDATA	·	·	·	·	1	1	1	1
$	·	·	·	·	·	1	·	·
T:SUP	·	·	·	·	·	·	1	·
T:DEM	·	·	·	·	·	·	·	1

Figure 18. Module graph reachability matrix associated with Figure 12.

Label	Module or Genus
1	&TRANS
1	&SDATA
1	&CDATA
2	&TDATA
3	$
3	T:SUP
3	T:DEM
1	PLANT
2	SUP
1	CUST
2	DEM
1	LINK
2	FLOW
2	COST

Figure 19. Topological labeling of the modular structure of Figure 4.

Acknowledgment

The foundations of structured modeling emerged early in this decade from my efforts to develop a broadly applicable theory of model aggregation. During and since those early days, I received much valuable input from students and colleagues. The students (most of whom have since graduated) include E. Brehm, S. Chari, A. Dechter, C. K. Farn, V. Francis, S. Gokhale, A. Jain, S. Jain, C. Jones, M. Shimony, and G. Zall. The colleagues include G. Bradley, P. Chen, R. Dembo, G. Diehr, D. Dolk, H. Greenberg, J. Jackson, M. Lenard, J. Mamer, G. Wright, and P. Zipkin. All have my lasting appreciation.

This work was supported partially by the National Science Foundation, the Office of Naval Research, and the Navy Personnel Research and Development Center (San Diego).

References

AHO, A. V., J. E. HOPCROFT AND J. D. ULLMAN. 1983. *Data Structures and Algorithms*, Addison-Wesley, Reading, Mass.

BERZTISS, A. T. 1975. *Data Structures: Theory & Practice*, Ed. 2. Academic Press, New York.

GEOFFRION, A. 1987a. An Introduction to Structured Modeling, *Mgmt. Sci.* **33,** 547–588. [A version that includes a section on implementation is available as Working Paper No. 338, Western Management Science Institute, University of California, Los Angeles, revised March 1988.]

GEOFFRION, A. 1987b. Modeling Approaches and Systems Related to Structured Modeling. Working Paper 339, Western Management Science Institute, University of California, Los Angeles (February).

GEOFFRION, A. 1988. SML: A Model Definition Language for Structured Modeling. Working Paper 360, Western Management Science Institute, University of California, Los Angeles (May).

GEOFFRION, A. 1989. Computer-Based Modeling Environments. To appear in *Eur. J. Opntl. Res.*

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

JAMES, R. C., AND E. F. BECKENBACH (eds.). 1976. *James/James Mathematics Dictionary*, Ed. 4. Van Nostrand Reinhold, New York.

KNUTH, D. E. 1973. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Ed. 2. Addison-Wesley, Reading, Mass.

SOWA, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, Mass.

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