

THIRD COPY

1

Working Paper No. 346

AD-A215 382

THE THEORY OF STRUCTURED MODELING

by

ARTHUR M. GEOFFRION

DTIC
SELECTE
DEC 07 1989
S D

May 1987

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

WESTERN MANAGEMENT SCIENCE INSTITUTE
University of California, Los Angeles

WESTERN MANAGEMENT SCIENCE INSTITUTE
University of California, Los Angeles

Working Paper No. 346

May, 1987

THE THEORY OF STRUCTURED MODELING

by

Arthur M. Geoffrion

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By <i>par C2</i>	
Distribution	
Availability Codes	
Dist	Avail and/or Special
<i>A-1</i>	

Abstract

This paper presents a formal development of the definitions and theory of structured modeling. It is a companion to the author's recently published paper "An Introduction to Structured Modeling".



Acknowledgments

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, A. Jain, S. Jain, C. Jones, and M. Shimony. 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 partially supported by the National Science Foundation, the Office of Naval Research, and the Navy Personnel R&D Center. The views contained in this report are those of the author and not of the sponsoring agencies.

THE THEORY OF STRUCTURED MODELING

The author's basic paper "An Introduction to Structured Modeling" (Geoffrion <1987>) is an informal, example-based exposition. The present paper, in contrast, presents a formal development of the definitions and theory of structured modeling. It is intended for a technical audience rather than for prospective practitioners of the structured modeling approach.

We assume that the reader is familiar with the first three sections of Geoffrion <1987> in order to eliminate having to furnish here a detailed motivation for and introduction to structured modeling, its uses, and its connections to other fields.

The first two sections present the basic definitions of structured modeling. The third develops related theoretical results, and the final section gives a brief conclusion. An extended example illustrating all of the definitions and theoretical properties is given in Appendix 1 for easy reference.

An important topic not covered in this paper is the matter of a detailed notational system for expressing structured models. One such, the one that is the basis for the FW/SM experimental prototype structured modeling system now being constructed, will be detailed in a forthcoming report.

The following excerpt from Geoffrion <1987> helps to set the stage for the technical development that follows.

The formal framework of structured modeling is based on discrete mathematics. It uses a hierarchically organized, partitioned, and attributed acyclic graph to represent a model or a model class. Particular attention is given to representing semantic as well as mathematical structure, and to compatibility with four of the most fundamental manipulations applied to models: retrieval, expression evaluation, solving a simultaneous system, and optimization.

At the core of structured modeling is the notion of a definitional system, that is, a system of definitions of all of the elements constituting a "model". The definitions have some special properties: they are typed (there are five types), correlated (interdependencies are explicit), and certain of the types are value-bearing. Moreover, the definitions are grouped

by definitional similarity, the resulting groups are organized hierarchically by conceptual similarity, and the whole system must be free of circularity.

This kind of definitional system turns out to be widely applicable within model-oriented fields such as MS/OR/DSS (for finance, logistics, marketing, production, and other application areas), information systems, economics, and engineering. Thus structured modeling ideas have the potential for wide adoption.

This kind of definitional system also turns out to have deep connections to formalisms used in artificial intelligence, database management, programming language design, and software engineering. These connections invite cross-fertilization among these fields from the modeling perspective.

The mathematical prerequisites of this paper are modest. Elementary directed graph theory is the main area requiring some prior familiarity. The terminology used is fairly 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 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 descendents of any given node are called **siblings**. Every node in a rooted tree has a unique **rootpath** from the node to the root (the rootpath includes the given node). 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 **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. There are very simple and efficient algorithms for performing topological sorts (see, e.g., Knuth <1973>, p. 258ff.).

A **tuple** is a finite nonempty ordered collection of components. A tuple is **segmented** when its components are partitioned in a contiguous way with non-empty 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. THE CORE CONCEPTS OF STRUCTURED MODELING

This section presents the core concepts of the structured modeling framework as a collection of formal definitions. Concrete examples are needed for these definitions to be understood. To preserve the utility of this section and the next as convenient references, however, the examples are deferred to Appendix 1. Hence it would be best for the reader to read Appendix 1 in parallel with this section.

Commentary is interspersed to explain the intent of the definitions as an abstraction of the notion of a "model" in management science and related fields. This commentary does not amend or augment the formal framework in any way.

Models are viewed in terms of **elements**. There are five **element types**; these are the subject of the first five definitions.

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

This represents a primitive definition concerning a distinctly identifiable thing or concept. Every model must have at least one primitive entity element. Each is introduced at the discretion and convenience of the modeler without, however, any 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).

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 other entities already defined, and that does not require a "value". A compound entity element can represent a new entity or it can represent a relationship or association among extant entities. It can represent a set, and also a relation in the sense of discrete mathematics.

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

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

Most of the data "coefficients" and decision "variables" of conventional models are represented 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 is unstable in most model-based studies owing to statistical estimation options, sensitivity analysis, "what if" analysis, and other needs. 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 unique value in some range to this tuple -- more precisely, in the case of non-entity 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 non-entity elements involved. No presumption is made concerning the mathematical structure of the domain or range spaces.

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 such constraint to indicate whether or not it is satisfied.

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 cross-references among the various elements of a model are a central focus of structured modeling. The calling sequence is the principal abstraction of these cross-references. 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 which play a similar role normally are put in the same segment. Note that primitive entity elements do not have calling sequences.

Two conventions should be followed in order for a collection of elements to represent properly the system being modeled. First, if the interpretation of a given element refers to 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. Second, no calling sequence should contain an element that is patently irrelevant to the calling element's 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 necessary for all cross-references to be defined within the formal framework.

8. A closed collection of elements is *acyclic* if there is no sequence $\{E_1, \dots, E_n\}$ such that E_1 calls $E_2, \dots, E_{n-1}$ calls E_n , where $n > 1$ and $E_n = E_1$.

Our concern is exclusively with systems of elements whose interpretations 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 call itself, it is neither possible nor desirable to exclude self-reference from the intended meaning of an element. In particular, it is permissible for an element to represent a directly (self-) recursive definition.

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

This is the first 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 could probably be made that would be satisfactory for most purposes. The rationale for closedness and acyclicity has already been given.

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 resulting mutually disjoint and exhaustive element sets are called *genera* (plural of *genus*).

Generic structure is intended to represent the phenomenon of "parallel structure" so commonly observed in real models. All of the elements of a given genus are supposed to be "alike" except for details; it should be meaningful and natural to speak of a "generic" element.

Models become large in practice primarily because of parallel structure. One indication of this is the wide use of index variables, which are used in conventional models to simplify notation through the exploitation of such structure.

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.

It turns out that, in most applications, some possible generic structures are inconsistent with any reasonable interpretation of "parallel structure", or they may have undesirable mathematical properties. We therefore limit consideration for the most part to generic structures satisfying the next property. It says, roughly, that the elements in a genus shouldn't be too different in terms of what other elements they depend upon. This acts to limit the allowable coarseness of generic structure.

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

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

The next concept 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 way, 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 non-terminal 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.

The next definition orders the modular structure tree in a way that ties modular structure closely to the underlying calling relationships among genera. The full significance of this ordering will not be apparent until Definition 23 of Section 2 and Propositions 5-7 of Section 3.

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 partial order is *monotone* in the following sense: if genus B calls genus A and A and B are descendents of distinct sibling nodes #1 and #2 respectively (A=#1 and/or B=#2 permitted), then #1 comes "before" #2 in their sibling order.

The definition of the "usual extension" of sibling set orders 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 all descendents of N1 come "before" all descendents of N2.

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. This possibility is recognized in the next two definitions.

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), a generic structure satisfying sim-

larity, 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 attribute elements*; these can 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.

The typical practical model is very 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 the case of 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 3 and the subsequent discussion. Alternatively, one-pass evaluation can be guided by the modular outline defined in the next section.

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. For example, an attribute value may not fall within the domain of definition of a function element's rule. We wish to preclude this possibility by assuming hereafter the following property unless it is explicitly dropped.

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

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.

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) adjacency is preserved in the modular structure trees, and (b) corresponding genera have the same number of calling sequence segments and call corresponding genera from each segment.

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 or arbitrary reasons.

As mentioned earlier, all of the above definitions are illustrated in Appendix 1.

2. ASSOCIATED CONCEPTS AND CONSTRUCTS

The previous section gave 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.

As for the previous section, the reader is strongly urged to read Appendix 1 in parallel with this section. It illustrates all of the definitions given here.

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. In this way, conceptual structures can be tailored so that they contain just the right level of detail for the intended target audience or conceptual analysis.

It should be noted that the term "view" has another meaning in the literature on database management systems (e.g., Date <1981> or Ullman <1982>).

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 $\{C_1, \dots, C_n\}$ such that some genus in C_1 calls some genus in C_2 , . . . , some genus in C_{n-1} calls some genus in C_n , where $n > 1$ and $C_n = C_1$. The modular structure itself is said to be **acyclicity-preserving** if every possible view is acyclicity-preserving.

Next we give 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 3).

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). What this means in simple terms is 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, these representations appear less natural for present purposes than the one based on preorder traversal.

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 portrays vividly 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.

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.

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).

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 sans 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.

The next 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. Usually it is best to choose (if possible) the row/column order so that an upper triangular matrix results. 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 therefore correspond 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). It is easy to see 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 (one final step is necessary: zero out any 1's on the diagonal).

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: (a) by columns to determine all of the elements referenced directly or indirectly by the column element, and (b) by rows to determine all of 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; see the algorithm given in Appendix 2.

3. 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 genus 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 the acyclicity of the elemental structure. Thus the supposition must be false.

QED

The second result shows that genus graphs inherit the acyclicity of element graphs.

Proposition 2. *Genus graphs are always acyclic.*

Proof. Recall that genus graphs are defined only when the generic structure satisfies generic similarity. Suppose that the genus graph contains a directed cycle, say $\{G_1, G_2, \dots, G_n, G_1\}$ where G_1 calls G_2 , which calls G_3 , ..., which calls $G_n = G_1$. 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 G_1 and trace a directed chain through elements in G_2 , G_3 , ..., and so on back to $G_n = G_1$ again. This may be done by supposition and the generic similarity property. If the terminal element in G_1 is the starting one, then 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 G_1 . 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.

QED

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."

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 $R+1$.

QED

Corollary 3.1. *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 e_1 and e_2 . The definition of generic structure guarantees that they are both of the same type. Proposition 3 guarantees that they both

have the same elemental rank as their genus rank. Since both elements are in the same genus, it follows that both elemental ranks must be the same.

QED

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) *for $r > 1$, 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 $r-1$.*

It is a simple matter to classify genera by rank. 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. 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).

We turn now to the acyclicity of module graphs. 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.

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 $\{C_1, \dots, C_n\}$ of the associated genus partition such that some genus in C_1 calls some genus in C_2 , ..., some genus in C_{n-1} calls some genus in C_n , where $n > 1$ and $C_n = C_1$. Consider the image

of this cycle in terms of 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.

QED

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.

QED

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 with 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, and 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 $\{S_1, \dots, S_n\}$ such that some genus descendent of S_1 calls some genus descendent of S_2 , ..., some genus descendent of S_{n-1} calls some genus descendent of S_n , where $n > 1$ and $S_n = S_1$.

Proof. To see that (i) $\implies$ (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) $\implies$ (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) $\implies$ (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 $\{S_1, S_2, \dots\}$ such that some genus descendent of S_i calls some genus descendent of S_j (i and j distinct) only if $j < i$. Obviously, this constructs a monotone ordering.

QED

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).

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 genera 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 devising a monotone-ordered modular structure can be viewed as an exercise in arranging all genera in outline form without any 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.

QED

4. CONCLUSION

The basic concepts of structured modeling, introduced informally and used in Geoffrion <1987>, have all been developed here formally. In addition, we have obtained some associated theoretical results for possible use in future work. An extensive example appears in Appendix 1.

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

This paper is part of a series. The next in this series will present formally a complete notational system for expressing structured models and model schemata. That notational system is used by an experimental prototype now under development that will be the subject of another paper.

REFERENCES

- AHO, A.V., J.E. HOPCROFT and J.D. ULLMAN <1983>. *Data Structures and Algorithms*, Addison-Wesley, Reading, MA.
- BERZTISS, A.T. <1975>. *Data Structures: Theory & Practice*, Second Edition, Academic Press, New York.
- DATE, C.J. <1981>. *An Introduction to Database Systems*, Volume 1, Third Edition, Addison-Wesley, Reading, MA.
- GEOFFRION, A. <1987>. "An Introduction to Structured Modeling," *Management Science*, 33:5 (May), 547-588. A version that includes a section on implementation will also appear in *Proceedings of the Conference on Integrated Modeling Systems* (held at the University of Texas, Austin, October 1986). The latter version is available as Working Paper No. 338, Graduate School of Management, UCLA, revised February 1987.
- HARARY, F., R.Z. NORMAN and D. CARTWRIGHT <1965>. *Structural Models: An Introduction to the Theory of Directed Graphs*, John Wiley, New York.
- KNUTH, D.E. <1973>. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Second Edition, Addison-Wesley, Reading, MA.
- SOWA, J.F. <1984>. *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA.
- ULLMAN, J.D. <1982>. *Principles of Database Systems*, Second Edition, Computer Science Press, Rockville, MD.

APPENDIX 1

AN ILLUSTRATIVE EXAMPLE: THE TRANSPORTATION PROBLEM

This appendix is designed to be read in parallel with Sections 1-3. It illustrates all of the definitions and most of the propositions thereof using a very simple model familiar to anyone who has taken a first course in management science: the ordinary Hitchcock-Koopmans transportation model.

Consider two plants producing a single product for shipment to three customers. Every eligible plant/customer transportation link has an associated unit transportation cost. Every plant has a maximum supply capacity and every customer has an exact demand requirement. We are interested in alternative patterns of transportation flows that honor production capacities and demand requirements, and we are also interested in total transportation cost.

This model is presented first informally in natural language. It is then developed formally within the structured modeling framework.

1. There is a plant in Dallas called DAL.
2. There is a plant in Chicago called CHI.
3. DAL has a supply capacity of 20,000 tons.
4. CHI has a supply capacity of 42,000 tons.
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 of 25,000 tons.
9. ATL has a demand of 15,000 tons.
10. CLEV has a demand of 22,000 tons.
11. There is a transportation link from DAL to PITTS.
12. There is a transportation link from DAL to ATL.
13. There is a transportation link from DAL to CLEV.
14. There is a transportation link from CHI to PITTS.
15. There is a transportation link from CHI to CLEV.
16. There can be a nonnegative transportation flow (in tons) over any transportation link.
17. The cost rate of using the link from DAL to PITTS is \$23.50 per ton.
18. The cost rate of using the link from DAL to ATL is \$17.75 per ton.

19. The cost rate of using the link from DAL to CLEV is \$32.45 per ton.
20. The cost rate of using the link from CHI to PITTS is \$7.60 per ton.
21. The cost rate of using the link from CHI to CLEV is \$25.75 per ton.
22. There is a TOTAL COST associated with all transportation flows equal to the sum over all links of the transportation flow times the cost rate.
23. The total transportation flow leaving DAL either does or does not pass the test of falling within its supply capacity.
24. The total transportation flow leaving CHI either does or does not pass the test of falling within its supply capacity.
25. The total transportation flow arriving at PITTS either does or does not pass the test of being exactly equal to the PITTS demand.
26. The total transportation flow arriving at ATL either does or does not pass the test of being exactly equal to the ATL demand.
27. The total transportation flow arriving at CLEV either does or does not pass the test of being exactly equal to the CLEV demand.

The standard optimization problem associated with this model is to find values for all transportation flows so as to minimize TOTAL COST subject to a positive outcome for all of the tests defined in 23-27.

Items 1-2 and 5-7 will be represented as primitive entity elements.

Items 11-15 will be represented as compound entity elements. Using an obvious notation, the (segmented) tuple for 11 can be written (1;5), the tuple for 12 as (1;6), and so on. Note that a semicolon is used to delimit segments.

Items 3-4, 8-10, 16, and 17-21 will be represented as attribute elements. The tuple for 3 is (1), the tuple for 8 is (5), the tuple for 17 is (11), and so on. All values have the real numbers as their range. A comment is in order concerning 17: why is its tuple not (1;5)? The answer is that, although 17 does indeed refer to DAL and to PITTS, the purpose of 17 is to describe a property associated with the transportation link from DAL to PITTS; item 11 is therefore the proper reference. Of course, 11 refers to 1 and to 5, so 17 refers *indirectly* to 1 and to 5.

Note that item 16 actually stands for a collection of five items, each one pertaining to a specific link. They will be labeled 16a, 16b, ..., 16e (corresponding, respectively, to items

11-15). These items would all be treated as variable attribute elements in the context of the standard optimization problem mentioned above.

Item 22 will be represented as a function element. Its tuple is (17,18,19,20,21; 16a,16b,16c,16d,16e). Note that the first segment is for cost rates, and that the second is for flows. The rule is a linear function. The range of the value is the real numbers. The domain is a pair of real vectors of identical dimension.

Items 23-27 will be represented as test elements. The tuple for 23 is (16a,16b,16c; 3), the tuple for 25 is (16a,16d; 8), and so on. The rules for 23 and 24 correspond to linear inequalities viewed as logical expressions, while the rules for 25-27 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 said constraints hold for a given set of numerical transportation flows.

The calling sequences have been noted above as segmented tuples with semicolon delimiters. One would say, for example, that the element corresponding to 23 calls the elements corresponding to 16a, 16b, 16c in its first calling sequence segment, and 3 in its second segment.

Given these calling sequences, the elements clearly constitute a closed collection with 31 members (remember that 16 must be counted five times).

This collection clearly is acyclic because every call is to some element that appears above the calling element in the list of items as written above.

Since the collection of elements is nonempty, finite, closed, and acyclic, it constitutes an elemental structure. The element graph (sans attributes) of this elemental structure is shown in Figure 1. Clearly it is acyclic.

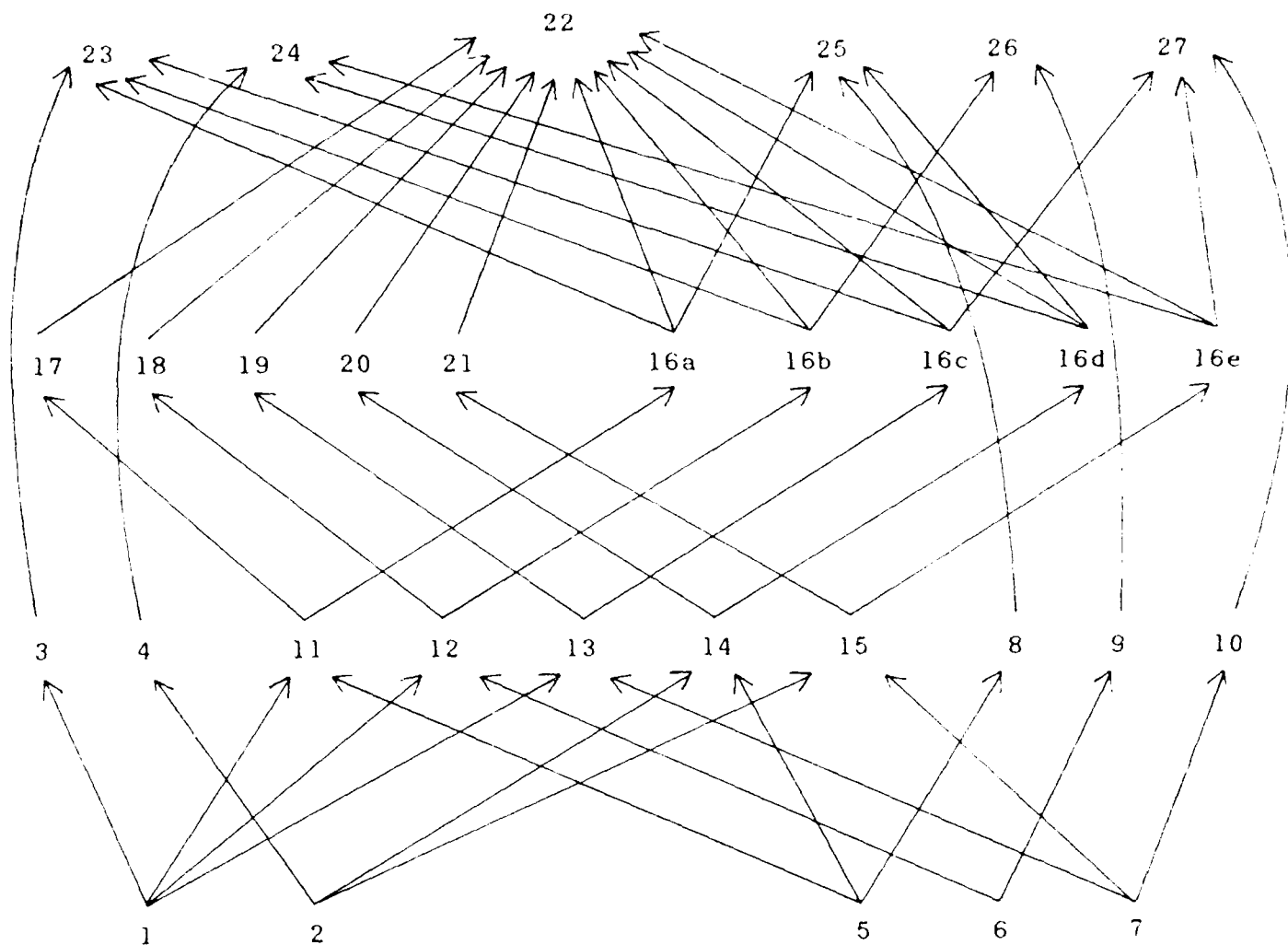


Fig. 1 ELEMENT GRAPH

The adjacency matrix corresponding to the element graph is as follows when the rows and columns are ordered in the same sequence as the original items. Row and column spaces have been introduced to preserve the grouping of the original items. Actually, we give a slight variant of the adjacency matrix that records the calling sequence segment for each call. Observe that this matrix is upper triangular; this verifies acyclicity.

Each column indicates which elements the column element calls. Each row indicates which elements call the row element.

[illegible]

[illegible]

Each column tells which elements are called directly or indirectly by the column element. For example, element 17 directly or indirectly calls elements 1, 5, and 11. Each row tells which elements directly or indirectly call the row element. For example, element 1 (representing the plant in Dallas) is called directly or indirectly by fifteen elements.

A natural generic structure is indicated by the grouping of the items as originally given. For example, the primitive entities are partitioned into plants and customers. Here is that generic structure, with each cell (genus) named for future reference.

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

Note that a partition can have but a single cell (e.g., the compound entity partition); and a cell can have as few as 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 the adjacency matrix variant detailed previously; its rows and columns are grouped by genus and it records the calling sequence segment for each call. 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 FLOW for both elements, and the second segment calls only SUP for both elements. Thus generic similarity holds for this genus. Similar checks can be carried out for the other genera.

The genus graph corresponding to this generic structure is shown in Figure 2. It is acyclic, as predicted by Proposition 2. Comparison with the element graph (sans attributes) shows that it is a condensation of it.

The genus ranks are easy to see from the genus graph. The topological sort first identifies PLANT and CUST, as these have no incoming arcs. After "erasing" 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 of the topological sort is as follows. This illustrates Proposition 4.

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

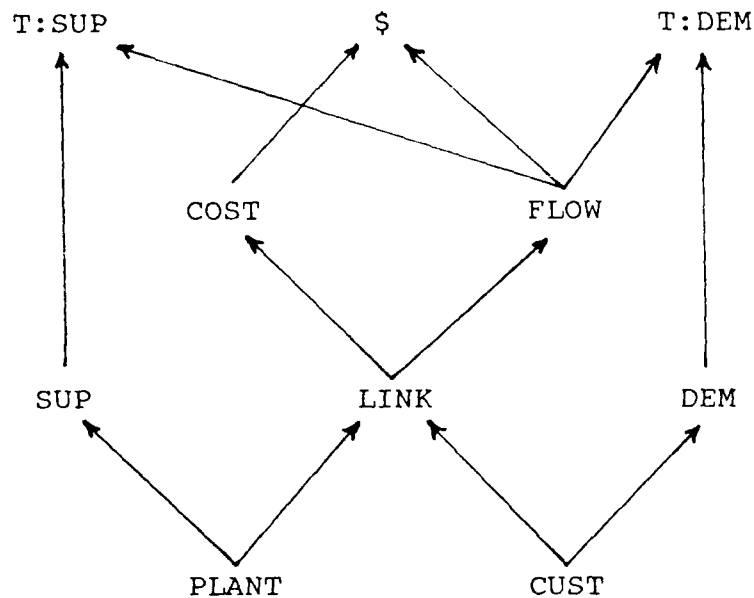


Fig. 2 GENUS GRAPH

The adjacency matrix corresponding to the genus graph is:

	P		C		L	F	C		T	T
	L	S	U	D	I	L	O		:	:
	A	U	S	E	N	O	S		S	D
	N	P	T	M	K	W	T	\$	U	E
	T	P	T	M	K	W	T	\$	P	M
PLANT	.	1	.	.	1	.	.	.	.	.
SUP	.	.	.	.	.	.	.	.	1	.
CUST	.	.	.	1	1	.	.	.	.	.
DEM	.	.	.	.	.	.	.	.	.	1
LINK	.	.	.	.	.	1	1	.	.	.
FLOW	.	.	.	.	.	.	.	1	1	1
COST	.	.	.	.	.	.	.	1	.	.
\$	.	.	.	.	.	.	.	.	.	.
T: SUP	.	.	.	.	.	.	.	.	.	.
T: DEM	.	.	.	.	.	.	.	.	.	.

The reachability matrix corresponding to the genus graph is:

	P								T	T
	L		C		L	F	C		:	:
	A	S	U	D	I	L	O		S	D
	N	U	S	E	N	O	S		U	E
	<u>T</u>	<u>P</u>	<u>T</u>	<u>M</u>	<u>K</u>	<u>W</u>	<u>T</u>	<u>\$</u>	<u>P</u>	<u>M</u>
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

Consider the following modular structure. Note that each module is given a name beginning with an ampersand.

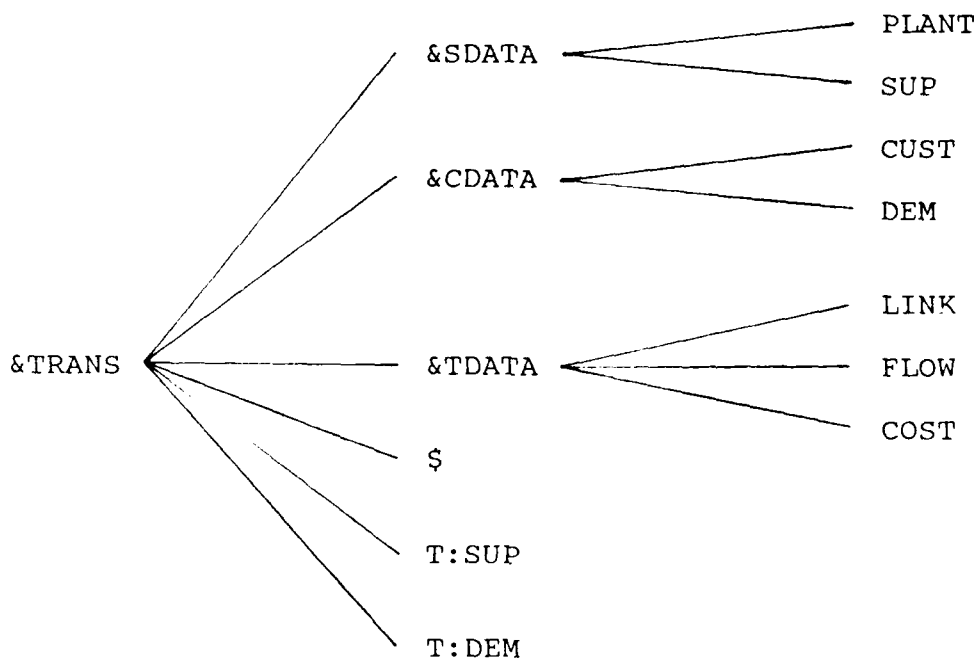


Fig. 3 A MODULAR STRUCTURE

The logic of this modular structure is that part of the model gives data concerning the sources of supply (namely PLANT and SUP), part gives data concerning the customers (namely CUST and DEM), and part gives data concerning transportation (namely LINK, FLOW, and COST). Each of these parts can be thought of as distinct conceptual units.

A monotone ordering of this modular structure is as follows:

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

Clearly this is an order for the modular structure tree. We need to verify that this order is monotone.

The preorder traversal of this ordered modular structure tree 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:

```

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

```

It is easy to see that there are no forward references among genera: if genus B calls genus A, then genus A is always above genus B in the list. Thus the order must be monotone. Clearly all this is consistent with Proposition 6.

In addition to the master view just described, these other views are possible (note that they all inherit a monotone ordering):

View 2

```

&TRANS
  &SDATA
    PLANT
    SUP
  &CDATA
    CUST
    DEM
  &TDATA
  $
  T:SUP
  T:DEM

```

View 3

```

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

```

View 4

```

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

```

View 5

```

&TRANS
  &SDATA
    PLANT
    SUP
  &CDATA
  &TDATA
  $
  T:SUP
  T:DEM

```

View 6

```

&TRANS
  &SDATA
  &CDATA
    CUST
    DEM
  &TDATA
  $
  T:SUP
  T:DEM

```

View 7

```

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

```

View 8

```

&TRANS
  &SDATA
  &CDATA
  &TDATA
  $
  T:SUP
  T:DEM

```

View 9

```

&TRANS

```

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 4.

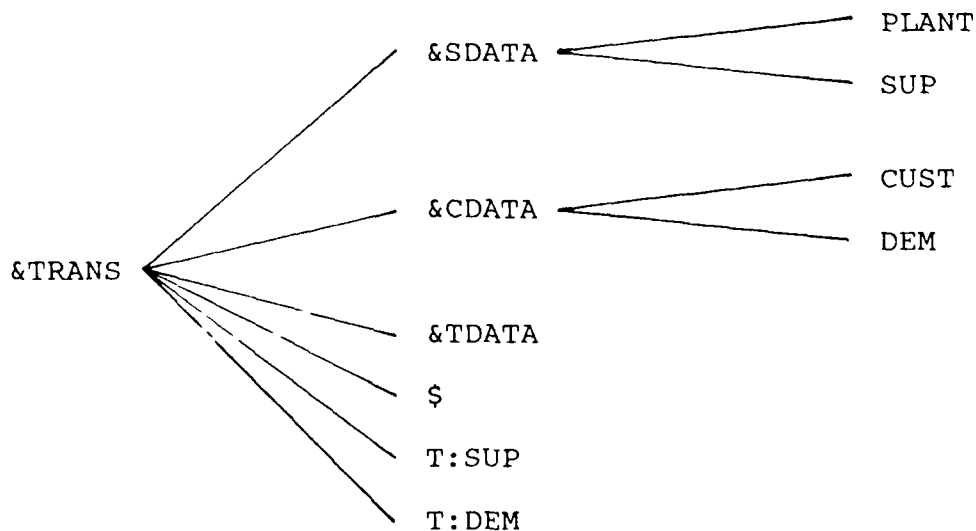


Fig. 4 MODULAR SUBTREE FOR VIEW 2

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 genus partition associated with View 2 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 up 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 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.

The module graph corresponding to View 2 is shown in Figure 5. It is acyclic as predicted by Proposition 5. Clearly, it is also a condensation of the genus graph.

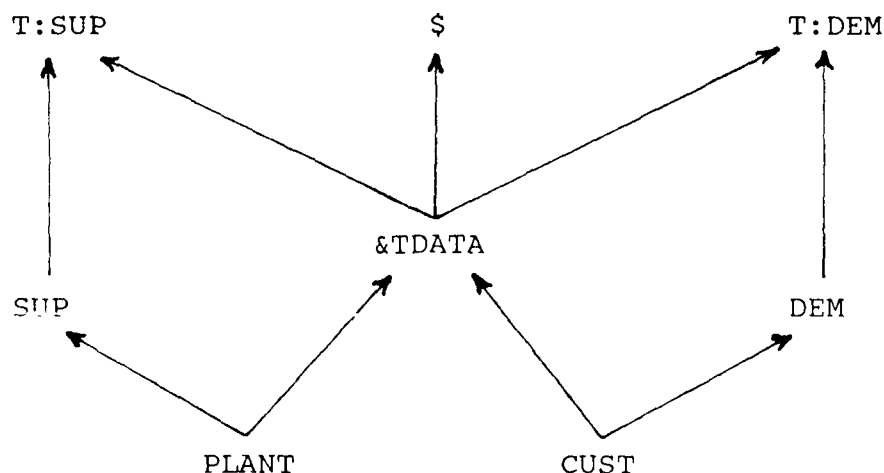


Fig. 5 MODULE GRAPH FOR VIEW 2

The adjacency matrix associated with View 2 is:

					&			
	P				T	T	T	
	L		C		D	:	:	
	A	S	U	D	A	S	D	
	N	U	S	E	T	U	E	
	T	P	T	M	A	\$	P	M
PLANT	.	1	.	.	1	.	.	.
SUP	.	.	.	.	.	.	1	.
CUST	.	.	.	1	1	.	.	.
DEM	.	.	.	.	.	.	.	1
&TDATA	.	.	.	.	.	1	1	1
\$	.	.	.	.	.	.	.	.
T: SUP	.	.	.	.	.	.	.	.
T: DEM	.	.	.	.	.	.	.	.

This can be seen either from the above graph or from the Boolean summation procedure mentioned near the end of Section 2.

The reachability matrix associated with View 2 is:

					&			
	P				T	T	T	
	L		C		D	:	:	
	A	S	U	D	A	S	D	
	N	U	S	E	T	U	E	
	T	P	T	M	A	\$	P	M
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

A structured model is at hand: the 31 elements corresponding to the natural language model given at the outset, together with the generic structure, modular structure, and monotone ordering given above. This model is incompletely specified owing to the absence of attribute element values, 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 well-defined because there is no way for the arguments of the rules of \$, T:SUP, and T:DEM to violate the domains of definition.

We wish now to illustrate the construction used in the proof of Proposition 7, which can be used to determine many possible monotone orderings of a modular structure. 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.

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

Now consider the second sibling set. A topological sort 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.

Consider now the third sibling set. A topological sort 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 can be summarized as follows. This is the topological labeling of the modular tree.

<u>Label</u>	<u>Module or Genus</u>
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

In all there are $1 \times 12 \times 1 \times 1 \times 2 = 24$ monotone orderings of the given modular structure, of which 12 are as follows.

<u>Ordering 1</u>	<u>Ordering 2</u>	<u>Ordering 3</u>	<u>Ordering 4</u>
&TRANS	&TRANS	&TRANS	&TRANS
&SDATA	&SDATA	&SDATA	&SDATA
PLANT	PLANT	PLANT	PLANT
SUP	SUP	SUP	SUP
&CDATA	&CDATA	&CDATA	&CDATA
CUST	CUST	CUST	CUST
DEM	DEM	DEM	DEM
&TDATA	&TDATA	&TDATA	&TDATA
LINK	LINK	LINK	LINK
FLOW	FLOW	FLOW	FLOW
COST	COST	COST	COST
\$	\$	T:SUP	T:SUP
T:SUP	T:DEM	\$	T:DEM
T:DEM	T:SUP	T:DEM	\$

<u>Ordering 5</u>	<u>Ordering 6</u>	<u>Ordering 7</u>	<u>Ordering 8</u>
&TRANS	&TRANS	&TRANS	&TRANS
&SDATA	&SDATA	&CDATA	&CDATA
PLANT	PLANT	CUST	CUST
SUP	SUP	DEM	DEM
&CDATA	&CDATA	&SDATA	&SDATA
CUST	CUST	PLANT	PLANT
DEM	DEM	SUP	SUP
&TDATA	&TDATA	&TDATA	&TDATA
LINK	LINK	LINK	LINK
FLOW	FLOW	FLOW	FLOW
COST	COST	COST	COST
T:DEM	T:DEM	\$	\$
\$	T:SUP	T:SUP	T:DEM
T:SUP	\$	T:DEM	T:SUP
<u>Ordering 9</u>	<u>Ordering 10</u>	<u>Ordering 11</u>	<u>Ordering 12</u>
&TRANS	&TRANS	&TRANS	&TRANS
&CDATA	&CDATA	&CDATA	&CDATA
CUST	CUST	CUST	CUST
DEM	DEM	DEM	DEM
&SDATA	&SDATA	&SDATA	&SDATA
PLANT	PLANT	PLANT	PLANT
SUP	SUP	SUP	SUP
&TDATA	&TDATA	&TDATA	&TDATA
LINK	LINK	LINK	LINK
FLOW	FLOW	FLOW	FLOW
COST	COST	COST	COST
T:SUP	T:SUP	T:DEM	T:DEM
\$	T:DEM	\$	T:SUP
T:DEM	\$	T:SUP	\$

The other 12 orderings are the same, but have FLOW and COST reversed in position.

Finally, we give some examples of a model schema. One such is the A-partially specified model just developed. 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 of the 24 monotone orderings just indicated for modular structure. 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 bona fide 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 a forthcoming paper.

APPENDIX 2

CALCULATING A REACHABILITY MATRIX FROM AN ADJACENCY MATRIX

Let G be an acyclic directed graph and let A be an adjacency matrix for it. Assume (a) that A is upper triangular, i.e., the nodes have been topologically sorted, and (b) that all entries are 0 or 1 (thus no segment identification data can be encoded as in the example of Appendix 1). The following algorithm calculates the reachability matrix R (associated with the same node ordering) columnwise in a single pass beginning with the first column. Assume that there are N nodes ($N > 1$). A superscript denotes an entire column of a matrix.

Step 1. Put $R^1 = A^1$. Set $n = 2$.

Step 2. Put $R^n = A^n + \sum_{h: A_h^n = 1} R^h$ where all addition is Boolean.

Step 3. If $n = N$, go to Step 4.
If $n < N$, increment n by 1 and return to Step 2.

Step 4. Add the identity matrix to R and STOP.

This algorithm makes it easy to calculate a reachability matrix for any element graph, genus graph, or module graph once the nodes have been ordered so that the adjacency matrix is upper triangular. Such an order is always available from a monotone ordering.

The validity of the algorithm is most easily shown by induction on the columns of R .

Proof. Let $S + I$ be the true reachability matrix. Let R be the result at the conclusion of Step 3. We wish to show that $R^j = S^j$ for $1 < j < N$. Certainly this is true for $j = 1$, for the algorithm puts $R^1 = A^1$; A^1 and S^1 must both be 0-vectors because, by assumption, the first node has no incoming arcs. Hypothesize that $R^j = S^j$ for $1 < j < n$ ($2 < n < N$). It remains to show that $R^n = S^n$. It suffices to show that $R_i^n = S_i^n$ for any typical component i .

Either $A_i^n = 1$ or $A_i^n = 0$. In the first case, the algorithm puts $R_i^n = 1$; as S_i^n evidently equals 1 also in this case, we

have the desired conclusion $R_1^n = S_1^n$. It suffices to consider the other case wherein $A_1^n = 0$.

Either $S_1^n = 1$ or $S_1^n = 0$. In the first case, there is a directed path $(j_1 = i, j_2, \dots, j_k = n)$ from i to n . This implies that there is a directed path from i to j_{k-1} , and so $S_i^{j_{k-1}} = 1$. Since $j_{k-1} < n$, we have by the inductive hypothesis that $R_i^{j_{k-1}} = 1$. But $A_{j_{k-1}}^n = 1$, and so the algorithm puts $R_1^n \geq S_i^{j_{k-1}}$. Thus we have the desired conclusion $R_1^n = 1 = S_1^n$.

Finally, consider the case $A_1^n = S_1^n = 0$. Then R_1^n must equal 0 also, for otherwise there would have to be an index h such that $A_{h1}^n = 1$ and $R_1^h = 1$; but this would contradict $S_1^n = 0$, for $R_1^h = S_1^h$ by the inductive hypothesis ($A_h^n = 1$ and $S_1^h = 1$ implies that there is a directed path from i to n , which would violate the case assumption).

QED

A slightly less efficient alternative is the Roy-Warshall algorithm described, for example, on page 132 of Berztiss <1975>.