REUSING STRUCTURED MODELS VIA MODEL INTEGRATION

Arthur M. Geoffrion

John E. Anderson Graduate School of Management
University of California, Los Angeles
405 Hilgard Avenue
Los Angeles, California 90024

ABSTRACT

This paper begins with a review of reusability and modularity ideas from the software engineering literature, most of which are applicable to the modeling context. Many features of structured modeling support reusability and modularity, and these are noted. The main focus of the paper, however, is on achieving reusability and modularity via the integration of two or more model schemas. A 5-step approach for integrating schemas written in SML (Structured Modeling Language) is proposed for this purpose. Examples are given to illustrate this approach, and the pros and cons of structured modeling for reuse are discussed at some length.

INTRODUCTION

"Reusability" and "modularization" concepts play a key role in the part of software engineering that is concerned with improving software development productivity. The analogies between programming and modeling suggest that some of what software engineers have learned about these concepts might be adapted to improve model development productivity.

Reusability need not involve model integration, but integration is an important way in which reuse can occur in the context of modeling. Reuse via model integration is the central focus of this paper.

In particular, we treat this topic from the viewpoint of *structured modeling* (Geoffrion [9] [11]). It would be impossible to give an adequate introduction to structured modeling within the confines of this paper, and so we must presume prior knowledge on the part of the reader. In a nutshell, however, one can say that structured modeling aims to provide a formal conceptual framework of considerable generality for modeling, within the broader task of laying the foundation for a new generation of modeling environments. The framework uses a hierarchically organized, partitioned, and attributed acyclic graph to represent the semantic as well as mathematical structure of a model. A language called SML has been designed and implemented to support this framework (Geoffrion [12]).

The organization of this paper as follows. Section 1 reviews reusability and modularity ideas from the software engineering literature for the benefit of readers who may not be familiar with these ideas. Section 2 presents a procedure for integrating the general structure of two distinct models written in SML. Section 3 illustrates this approach in a simple but instructive setting. The last section discusses some of the issues that arise when one considers structured modeling in the context of the ideas presented in the prior sections.

Papers by other authors on model integration include Blanning [3], Bradley and Clemence [6], Kottemann and Dolk [17], Liang [18], Muhanna and Pick [22], Murphy et. al. [23], Tsai [27], and Zeigler [30] (Chaps. 4,5). Each adds a dimension not treated here. For example, Kottemann and Dolk [17] addresses the integration of model manipulation operations, whereas we limit consideration to model definition alone. For a more general discussion of model integration, see Geoffrion [10]. That paper distinguishes several different kinds of model integration and assesses the strengths and weaknesses of structured modeling in support of each.

1. SELECTED IDEAS FROM SOFTWARE ENGINEERING

Reusability is a prominent topic in software engineering because being able to use previously created and (presumably) debugged code saves work that is expensive and for which skilled personnel are in short supply. Advantages accrue not only to the initial development activity, but also to the very important activities of maintenance and modification. Moreover, code is not the only thing that can be reused; it can be advantageous also to reuse data, program and system designs and architecture, specifications, test plans, and so on. See Freeman [8] for an excellent collection of readings on reusability.

As might be expected, different authors focus on different aspects of reusability. Here is a selected sample of points and opinions.

One recent article by a leading expert on software engineering cites "reusing components" as one of six primary options for improving software productivity (Boehm [5]). To Boehm, reusability is most closely related to two things: libraries of software components (like mathematical and statistical routines) and 4GL/application generators (like Focus and Nomad). The justification for the second item appears to be that various of the software components constituting a 4GL processor or application generator are reused at each application in place of writing a program in a traditional language. Reuse is by reconfiguring the pre-existing components into a relatively special purpose software system, or by using them in lieu of a special purpose software system by "executing" a high level specification of functionality.

0073-1129/89/0000/0601$01.00 © 1989 IEEE

Near the end of the article, Boehm offers this opinion:

"Thus, for the 1995-2000 time frame, we can see that two major classes of opportunities for improving software productivity exist: providing better support systems for broad-domain applications, involving fully integrated methods, environments, and modern programming languages such as Ada; and extending the number and size of the domains for which we can use domain-specific fourth-generation languages and application generators."

If transposed from the context of programming to the context of modeling, this view is very much in the spirit of structured modeling.

Another view of reusability is provided by Balzer, Cheatham, and Green [1]. Their basic approach is that software should be produced by end users who write formal specifications that can be transformed more or less automatically by computer into the desired software. Maintenance would not be done on source code, as is the usual case today, but rather on the formal specification; revised code would be regenerated as needed. This automation-based paradigm for software development admittedly is idealistic: the necessary technology is not yet adequate for many applications. Thus the authors accept skilled intervention with respect to strategic implementation decisions. However, they argue that both the documentation of these decisions (they call this the "development") and their execution should be automated. The authors claim important advantages for the automation-based paradigm, among which is software reusability. To quote:

"Software libraries - with the exception of mathematical subroutine libraries - have failed because the wrong things are in those libraries. They are filled with implementations, which necessarily contain many arbitrary decisions. The chance that such an implementation is precisely right for some later application is exceedingly small.

Instead, we should be placing specifications and their recorded development in libraries. Then, when a module is to be reused, the specification can be modified appropriately - that is, maintained - and its development changed accordingly. Thus, maintenance of both the specification and its development becomes the basis of reuse, rather than a fortuitous exact match with a previously stored implementation."

There seems to be a strong connection between the automation-based paradigm and the 4GL/application generators advocated by Boehm: the latter can be viewed as domain-specific examples of the former.

A third article discussing reusability, Kartashev and Kartashev [16], takes the position that the reuse of software components is one of only two main options for reducing software costs. Within this option, two approaches are noted:

"... The *composition approach* provides for some abstract initial representation of reused program modules, then the use of a composition technique such as parameterized programming, the UNIX pipe mechanism, or special specification techniques to construct a complex program from reusable modules with the interfaces specified by the composition technique.

The second approach involves the creation of libraries of reused components that do not need preliminary specifications to be connected to each other."

Goguen [15] develops the first approach in detail. He gives this brief and tantalizing synopsis at the outset:

"...The goal is to make programming significantly easier, more reliable, and cost effective by *reusing* previous code and programming experience to the greatest extent possible. Suggestions given here include: systematic (but limited) use of semantics, by explicitly attaching *theories* (which give semantics, either formal or informal) to software components with *views* (which describe semantically correct interconnections at component interfaces); use of *generic* entities, to maximize reusability; a distinction between *horizontal* and *vertical* composition; use of a *library interconnection language*, called LIL, to assemble large programs from existing entities; support for different levels of formality in both documentation and validation; and facilitation of program understanding by animating abstract data types and module interfaces."

Several of these ideas have analogues in structured modeling.

Goguen's most intriguing ideas, and his main focus, are those relating to LIL. The structured modeling analogue of LIL would be a language for integrating models (schemas or even fully specified structured models). No such language presently exists for structured modeling. In the modeling literature more generally, the only efforts I know of along these lines are the recent contributions by Bradley and Clemence [6] and by Muhanna and Pick [22]. Since such a language could greatly facilitate model reuse through integration, this is a potentially fruitful area for research. Some inspiration may be found in the literature on module interconnection languages, which has been surveyed by Prieto-Diaz and Neighbors [25].

A topic in software engineering closely related to reusability is *modularization*. The connection is that well modularized code is easier to reuse than code that is poorly modularized. Presumably, the same statement is true of models. See, for example, the classic paper by Parnas [24], which cites these benefits for a modular style of programming:

* <u>shorter development time</u> is possible because different people can work simultaneously on separate program modules;

* <u>evolutionary flexibility</u> is improved because drastic changes can be made to one module without impacting the others;

* <u>understandability</u> is improved because it should be possible to study the system one module at a time.

That paper champions <u>information hiding</u> as a criterion that often leads to better modularization than more traditional criteria. That is, each module "hides some design decision from the rest of the system", with its interfaces crafted "to reveal as little as possible about its inner working." The design decisions worth modularizing are those which are difficult or likely to change.

Modularization ideas play a major role in the thinking of many leading authors in the area of program and system design. For example, Yourdon and Constantine [29] describe the central focus of their book as follows:

> "Our concern is with the architecture of programs and systems. How should a large system be broken into modules? *Which* modules? Which ones should be subordinate to which? How do we know when we have a 'good' choice of modules? and, more important, How do we know when we have a 'bad' one? What information should be passed between modules? Should a module be able to access data other than that which it needs to know in order to accomplish its task? How should the modules be 'packaged' into efficient executable units in a typical computer?" [page xvi]

Much of the book's discussion of such questions can be translated easily into provocative positions about analogous questions in modeling. These positions deserve to be studied.

One of the most fashionable manifestations of modularization and related ideas in the service of reusability (and other popular goals) is *object-oriented programming*. A particularly vigorous presentation from this point of view is found in the book by Cox [7]. He argues that it is possible to develop software counterparts of integrated circuits; counterparts in that they can be reused readily by programmers much as integrated circuits are reused readily by hardware designers. Here are two intriguing passages to provide a taste of Cox's viewpoint:

> "Objects are tightly encapsulated and thus relatively independent of their environment. They can be designed, developed, tested, and documented as stand-alone units without knowledge of any particular application, and then stockpiled for distribution and reuse in many different circumstances. A reusable module acquires value unto itself, while the value of application dependent code has no value beyond that of the application as a whole. It can be tested and documented to much higher standards than has been done in the past." [page 26]

> "These concepts, working together, put reusability at the center of the programming process. Encapsulation raises firewalls around the objects within a system, so that if they change, other parts of the system can remain unchanged. Inheritance does the converse, by spreading decisions implemented in generic classes automatically to each of their subclasses. The net effect is making libraries of reusable code, called Software-IC libraries, a central feature of the system-building shop." [page 91]

This completes the review of reusability and modularization ideas from the software engineering literature.

We turn now to a step-by-step approach for integrating two structured modeling schemas. This is one way in which reuse can occur in modeling. Each participating schema can be viewed as a module.

2. A PROCEDURE FOR MODEL INTEGRATION

This section explains a procedure for integrating the general structure of two models. Let two SML model schemas be given.

By necessity, we presume that the reader is acquainted with the basic technical terms of SML (Geoffrion [12]). For example, the reader should know that an SML *schema* represents a model's general structure; that *Schema Properties* are context-sensitive syntax rules associated with the schema; and that it falls to SML's *elemental detail tables* to instantiate a schema to a specific model instance. In what follows, a "structural change" is any change other than simple renaming or changing an interpretation.

Step 1. Customize the two schemas to the roles that they will play in the integrated model, but do not violate the syntax or Schema Properties of SML. For example, achieve consistency in units of measurement.

a. Make any appropriate structural changes.

b. Customize the genus names, module names (including the root module name), key phrases, and interpretations of each schema as appropriate. Often this helps to reduce the amount of work that must be done at Step 3a.

Step 2. Make technical preparations for joining the two schemas, but do not violate the syntax or Schema Properties of SML.

a. Anticipate duplicate genus names, module names, key phrases, indices, functional and multi-valued dependency names, and symbolic parameter stems, that will occur at Step 3, and eliminate all conflicts.

b. Usually there is at least one pair of genera, one from each schema, that needs to be "merged". Identify all such pairs. Make ready for the merges by making both genus paragraphs as similar as possible for each merging genus pair. Sometimes they can be made identical, but not always. Structural changes are allowed.

c. Optionally, put the "inputs" and "outputs" of each schema in special modules. This may not be possible to do in a fully satisfactory way. Structural changes are allowed.

Step 3. Join the two schemas. This step can cause Schema Property violations.

a. Concatenate the two schemas. Choose an order that preserves monotonicity of the modular structure if possible.

b. Continue the work begun in Step 2b, if necessary, until each merging genus pair is identical. Drop the second genus replicate for each merging genus pair.

c. Make any other structural changes necessary for proper joining.

Step 4. Revise the integrated schema as necessary, structurally or non-structurally, to restore satisfaction of all Schema Properties. For example, it may be necessary to revise the modular structure so as to restore monotonicity.

Step 5. Optionally, make any desired final cosmetic or structural changes, being sure to observe the syntax and Schema Properties of SML.

The next section presents an illustrative application of this approach. Actually, the approach is carried out three times in a cumulative way that culminates in the integration of four distinct models.

Additional illustrative applications are given in the appendices of Geoffrion [14]. They apply the approach to three simple situations. The first integrates a forecasting model with the classical transportation model. The second integrates the transportation model with a multi-item economic order quantity model. The third integrates two transportation models to produce a two-echelon transshipment model.

All examples treat model schemas, not specific model instances. It would be worth examining these examples at the level of fully detailed instances, but that is not done here.

3. AN EXAMPLE

This section illustrates the approach of Section 2 with a small but richly instructive example that has been used by several authors. The example is due to Blanning, who discusses model integration at some length in several of his papers (e.g., Blanning [3], Blanning [4]). A recurring example in his papers is the "CORP" model, which has been discussed also by other authors (e.g., Muhanna [20]).

One of the interesting things about this model is that it can be viewed as four interconnected submodels. The development of this section explains how the appropriate interconnections can be accomplished by applying the integration procedure of Section 2 to SML representations of the component submodels.

The CORP model is discussed at length in Geoffrion [13] because although it appears to be cyclic, closer inspection reveals that it need not be. The Third Formulation therein repairs certain anomalies of the original model and presents it as a genuine SML schema. That is the version used here. It can be considered to be what the final integrated model should look like, except perhaps for its overly simple modular structure.

Original Component Models

There are four component models. The first estimates demand.

&MKT DEMAND ESTIMATION

PROD /pe/ There is a certain PRODUCT.

P (PROD) /a/ : Real+ The PRODUCT has a unit PRICE.

V (P) /f/ ; 800000 - 44000 * P The PRODUCT has a sales VOLUME given by a certain linear demand function in terms of PRICE.

The second model analyzes markup. It is the only component model that differs significantly from the corresponding one of Blanning. It does not calculate **P** from the values of **M, V,** and **E.** However, that value of P can be calculated by a solver for the satisfaction problem of finding **P** such that **M_VAR** is zero.

&MAR MARKUP ANALYSIS

PROD /pe/ There is a certain PRODUCT.

P (PROD) /va/ : Real+ The PRODUCT has a unit PRICE.

V (PROD) /a/ : Real+ The PRODUCT has a sales VOLUME.

E (PROD) /a/ : Real+ There is a total MANUFACTURING EXPENSE associated with the PRODUCT.

M (P,V,E) /f/ ; P * V / E The actual MARKUP for the PRODUCT is total revenue (PRICE times VOLUME) divided by total MANUFACTURING EXPENSE.

M_VAR (M) /f/ ; M - %M_TARGET The actual MARKUP less a certain target value is called the MARKUP VARIANCE.

The third component model predicts manufacturing cost.

&MFG MANUFACTURING

PROD /pe/ There is a certain PRODUCT.

U (PROD) /a/ : Real+ The PRODUCT has a UNIT COST of manufacture, exclusive of fixed manufacturing expenses.

V (PROD) /a/ : Real+ The PRODUCT has a sales VOLUME.

E (U,V) /f/ ; 1000000 + U * V The total MANUFACTURING EXPENSE for the PRODUCT is fixed manufacturing expenses plus UNIT COST times VOLUME.

The fourth model one calculates a key financial quantity.

&FIN FINANCIAL

PROD /pe/ There is a certain PRODUCT.

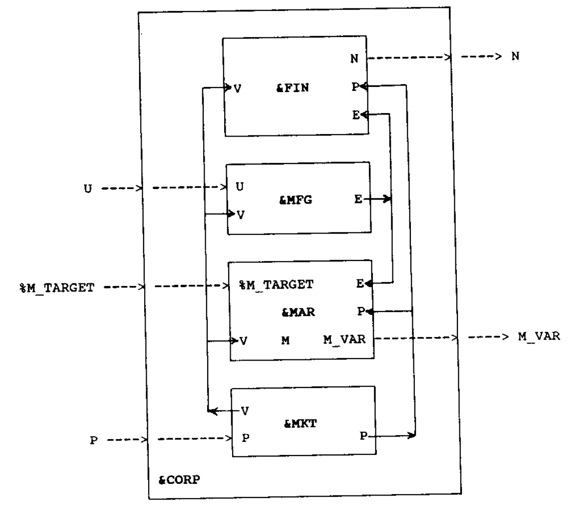
P (PROD) /a/ : Real+ The PRODUCT has a unit PRICE.

V (PROD) /a/ : Real+ The PRODUCT has a sales VOLUME.

E (PROD) /a/ : Real+ There is a total MANUFACTURING EXPENSE associated with the PRODUCT.

N (P,V,E) /f/ ; P * V - E The NET INCOME obtained for the PRODUCT is total revenue (PRICE times VOLUME) less total MANUFACTURING EXPENSE.

It is straightforward to interconnect these four components into a single composite model provided there is sufficient compatibility among definitions that appear in more than one component model. This is a critical requirement whenever component models are to be interconnected. Care has been taken to ensure that it holds here. The following interconnection diagram is adapted from Muhanna and Pick [21].



Note that &MKT has the task of supplying the value of V to the other three models, &MFG the value of E to two of the other models, &FIN the value of N, and &MAR the value of M_VAR. All these are obvious roles because V, E, N, and M_VAR are computed only once in the four models. U is not computed at all, and so must be supplied externally. The same is true for the symbolic parameter %M_TARGET. In addition, &MKT passes on the externally supplied value of P, which is modeled as a variable attribute, to the two other models that need it. This is one of two significant differences between the above diagram and the Muhanna-Pick diagram on which it is based; their diagram would receive and distribute P from &MAR (which they call "PRI") instead of from &MKT. If our diagram did it this way, it would be cyclic rather than acyclic. The second significant difference is noted in the next paragraph.

Thus **&CORP** can be viewed as having three inputs, U, %M_TARGET, and **P**, and two outputs, **N** and **M_VAR**. If a value of **P** is desired such that **M_VAR** is zero, then finding such a value is the task of a solver external to **&CORP**. Please note that this viewpoint differs from that of Blanning and Muhanna, who both appear to view **&CORP** itself (as a model) as bearing the responsibility for finding such a value. I believe that such a viewpoint confuses the notion of model and solver.

There is nothing to prevent the four component models from being interconnected per the diagram by some mechanism external to the SML notational system. Perhaps a capability of this sort should be provided by a modeling environment based on structured modeling; a formal interconnection language would be one possibility. If so, however, some defenses are needed to detect and repair definitional incompatibilities when similar model elements appear in multiple component models. (This is not unlike the problems that arise in information systems when distinct databases are to be integrated with one another; see, e.g., Batini, Lenzerini, and Navathe [2]).

That is not the approach taken here. In what follows, we integrate the four models by stages into a single composite SML schema. First, **&MKT** and **&MAR** will be integrated. Then &MFG will be folded in. Finally, **&FIN** will be integrated with the result of the previous two integrations.

It is evident from the partial ordering of the component models (such an ordering is obvious from the interconnection diagram) that it would be wiser to interchange the integration order of **&MAR** and **&MFG**, for the former depends on the latter. Wiser in the sense that monotonicity would be easier to maintain. We choose the less favorable order to help dispel the reader's possible suspicion that choosing the order of integration requires divine guidance.

Integrate &MKT and &MAR

Step 1. Null.

Step 2a. Null.

Step 2b. The merging genus pairs are PROD, P, and V in &MKT and, respectively, PROD, P, and V in &MAR. The PROD pair already has identical genus paragraphs. The P paragraphs differ only in that the first is /a/ and the second is /va/; make the first identical to the second. The V paragraphs differ substantially; make the second identical to the first (since the second paragraph will be dropped at Step 3b, it is not truly necessary to make this change).

Step 2c. Omit.

Step 3a. Concatenate in the order &MKT, &MAR. Call the new module &MKT_MAR.

Step 3b. Drop the PROD, P, and V paragraphs from &MAR.

Step 3c. Null.

Step 4. Null.

Step 5. Omit.

The result is as follows.

&MKT_MAR

&MKT DEMAND ESTIMATION

PROD /pe/ There is a certain PRODUCT.

P (PROD) /va/ : Real+ The PRODUCT has a unit PRICE.

V (P) /f/ ; 800000 - 44000 * P The PRODUCT has a sales VOLUME given by a certain linear demand function in terms of PRICE.

&MAR MARKUP ANALYSIS

E (PROD) /a/ : Real+ There is a total MANUFACTURING EXPENSE associated with the PRODUCT.

M (P,V,E) /f/ ; P * V / E The actual MARKUP for the PRODUCT is total revenue (PRICE times VOLUME) divided by total MANUFACTURING EXPENSE.

M_VAR (M) /f/ ; M - %M_TARGET The actual MARKUP less a certain target value is called the MARKUP VARIANCE.

Integrate &MKT_MAR and &MFG

Step 1. Null.

Step 2a. Null.

Step 2b. The merging genus pairs are PROD, V, and E in &MKT_MAR and, respectively, PROD, V, and E in &MFG. The PROD paragraphs already are identical. The V paragraphs differ substantially; the second needs to be made like the first, but this cannot be accomplished until Step 3b because &MFG has no P genus. The E paragraphs also differ substantially; the first needs to be made like the second, but this cannot be accomplished until Step 3b because &MKT_MAR has no U genus. Thus no changes at all are made at this substep.

Step 2c. Omit.

Step 3a. Concatenate in the order &MKT_MAR, &MFG. Call the new module &MKT_MAR_MFG.

Step 3b. Drop the second PROD paragraph. Make the second V paragraph identical to the first, then drop the second (obviously, it is not truly necessary to change V before dropping it). Make the first E paragraph identical to the second, then drop the second one.

Step 3c. Null.

Step 4. There is only one Schema Property violation, the one caused by the forward reference to U by the E paragraph. Cure it by moving the &MFG module to the position between &MKT and &MAR.

Step 5. Move the E paragraph to the position immediately after the U paragraph. This is more in keeping with the scopes of &MAR and &MFG. Change the modular structure (here we do not bother to list the genera)

from	to
&MKT_MAR_MFG	&MKT_MFG_MAR
&MKT_MAR	&MKT
&MKT	&MFG
&MFG	&MAR.
&MAR	

The result is as follows.

&MKT_MFG_MAR

&MKT DEMAND ESTIMATION

PROD /pe/ There is a certain PRODUCT.

P (PROD) /va/ : Real+ The PRODUCT has a unit PRICE.

V (P) /f/ ; 800000 - 44000 * P The PRODUCT has a sales VOLUME given by a certain linear demand function in terms of PRICE.

&MFG MANUFACTURING

U (PROD) /a/ : Real+ The PRODUCT has a UNIT COST of manufacture, exclusive of fixed manufacturing expenses.

E (U,V) /f/ ; 1000000 + U * V The total MANUFACTURING EXPENSE for the PRODUCT is fixed manufacturing expenses plus UNIT COST times VOLUME.

&MAR MARKUP ANALYSIS

M (P,V,E) /f/ ; P * V / E The actual MARKUP for the PRODUCT is total revenue (PRICE times VOLUME) divided by total MANUFACTURING EXPENSE.

M_VAR (M) /f/ ; M - %M_TARGET The actual MARKUP less a certain target value is called the MARKUP VARIANCE.

Integrate &MKT_MFG_MAR and &FIN

Step 1. Null.

Step 2a. Null.

Step 2b. The merging genus pairs are PROD, P, V, and E in &MKT_MFG_MAR and, respectively, PROD, P, V, and E in &FIN. The PROD and P pairs already have identical genus paragraphs. The V paragraphs differ substantially; make the second identical to the first (since the second paragraph will be dropped at Step 3b, it is not truly necessary to make this change). The E paragraphs also differ substantially; the second needs to be made like the first, but this cannot be accomplished until Step 3b because &FIN has no U genus.

Step 2c. Omit.

Step 3a. Concatenate in the order &MKT_MFG_MAR, &FIN. Call the new module &CORP.

Step 3b. Drop the second PROD and P paragraphs. Drop the second V paragraph. Make the second E paragraph identical to the first, then drop the second one (obviously, it is not truly necessary to change E before dropping it).

Step 3c. Null.

Step 4. Null.

Step 5. Change the modular structure

```
from                    to
     &CORP                   &CORP
        &MKT_MFG_MAR            &MKT
            &MKT                &MFG
            &MFG                &MAR
            &MAR                &FIN.
        &FIN
```

The result is as follows. Clearly it is consistent with the interconnection diagram given earlier. In fact, the diagram gives an accurate view of **&CORP**.

&CORP

&MKT <u>DEMAND ESTIMATION</u>

PROD /pe/ There is a certain <u>PRODUCT</u>.

P (PROD) /va/ : Real+ The PRODUCT has a unit <u>PRICE</u>.

V (P) /f/ ; 800000 - 44000 * P The PRODUCT has a sales <u>VOLUME</u> given by a certain linear demand function in terms of PRICE.

&MFG <u>MANUFACTURING</u>

U (PROD) /a/ : Real+ The PRODUCT has a <u>UNIT COST</u> of manufacture, exclusive of fixed manufacturing expenses.

E (U,V) /f/ ; 1000000 + U * V The total <u>MANUFACTURING EXPENSE</u> for the PRODUCT is fixed manufacturing expenses plus UNIT COST times VOLUME.

&MAR <u>MARKUP ANALYSIS</u>

M (P,V,E) /f/ ; P * V / E The actual <u>MARKUP</u> for the PRODUCT is total revenue (PRICE times VOLUME) divided by total MANUFACTURING EXPENSE.

M_VAR (M) /f/ ; M - %M_TARGET The actual MARKUP less a certain target value is called the <u>MARKUP VARIANCE</u>.

&FIN <u>FINANCIAL</u>

N (P,V,E) /f/ ; P * V - E The <u>NET INCOME</u> obtained for the PRODUCT is total revenue (PRICE times VOLUME) less total MANUFACTURING EXPENSE.

Modular structure aside, **&CORP** is indeed exactly the same as the Third Formulation in Geoffrion [13], as predicted at the outset of this section.

This completes our application, thrice in succession, of the SML schema integration approach given in Section 2.

4. DISCUSSION

This section discusses reusability and related ideas sketched in Section 1 from the point of view of structured modeling and the examples of model integration provided in Section 3 and in Geoffrion [14].

<u>ISSUE</u>: How well does structured modeling support "information hiding" and "modularization"?

Several features of structured modeling promote the objectives of information hiding and modularization:

(1) the core concept of hierarchical *modular structure* (Definition 12 of Geoffrion [11]);

(2) the concepts of *views* and *modular outlines* (Definitions 20 and 23 of Geoffrion [11]), and the closely related notion of using outline processor functions to conceal and reveal different portions of modular structure depending on the intended audience (Section 4.3 of Geoffrion [9]) ;

(3) the sharp distinction between models and solvers (Section 2.3 of Geoffrion [9]) helps to keep the internal concerns of each away from the other;

(4) model and solver libraries (Sections 2.4 and 4.3 of Geoffrion [9]) achieve a kind of information hiding and modularization at the macroscopic level (although some of the models in a model library could be relatively small reusable fragments);

(5) the distinction in SML between the general structure of the schema and the detailed data of the elemental detail tables enables users to be insulated from an excessive volume of problem specifics when that is appropriate;

(6) the distinction in SML between the formal part and the interpretation of each genus and module paragraph helps to insulate technical users from excessive problem domain information, and non-technical users from excessive technical details; and

(7) each individual element in a structured model is, in effect, a module (e.g., the details of the rule of a function element are totally localized).

These features may not go as far toward the support of information hiding and modularization as some people might wish (cf. Muhanna and Pick [21]). Certainly they do not go as far as the notion of a "black box," taken here to mean a hiding or module so secure that special authority is required before one is permitted to look inside it. Black boxes don't make sense in the context of modeling if integration is to be an important kind of reuse. The reason is that <u>almost any part of a model can serve as the focus of integration with another model</u>. Since integration is difficult or impossible if the focal part is inaccessibly hidden, it follows that virtually no part of a model can be black boxed without limiting the possibilities for integration. Obviously, to limit the possibilities for integration is to limit the possibilities for reuse.

To illustrate that almost any part of a model can serve as the focus of integration, consider the case of the classical transportation model. The development of Appendix 1 of Geoffrion [14] illustrates that the demand portion can be the focus of integration (with a forecasting model). Appendix 2 of Geoffrion [14] illustrates that the transportation links can be the focus of integration (with an economic order quantity model). Appendix 3 of Geoffrion [14] illustrates that the origins and their supply capacities can be the focus of integration (with another transportation model on the front end), and that the destinations and their demands can be the focus of integration (with another transportation model on the back end). It would be easy to develop other examples in which the transportation cost rate is the focus of integration with a freight rate estimation model, or in which the supply capacities are integrated with a more detailed supply capacity estimation model. Thus every part of the classical transportation model can serve as the focus of integration.

The most extreme form of information hiding is close to the idea of a "black box." How does one know whether a black box contains something that is truly consistent with what it is connected to? This is a particularly difficult worry in the realm of modeling, where model elements often have subtle definitions. The only way to know for sure is for the modeler to inspect the contents of the box. The contents alone suffice to explain the complete function of the box. **Unit of measurement**? Look inside to find out whether it is pounds, hundredweight, tons, or some other unit. **Demand point**? Look inside to find out whether it is an individual customer, all customers of a given class of trade in a certain area, etc. **Product**? Look inside to find out whether tan and black products are grouped, whether standard attachments are included with the basic product itself, and so on. **Cost**? Only a look inside will tell you whether it is book, actual, incremental, fully or partially loaded, etc. It would be grossly redundant to attach all this information to the "inputs" and "outputs" of a black boxed model, which one would have to do to accommodate all possible kinds of reuse.

This establishes that information hiding and modularization ought not to be taken to the black box extreme in the context of modeling. However, I accept these as criteria to help guide the practical use of features (1)-(4) listed above.

It also establishes that visibility of all model parts and explicit representation of their interconnections are important objectives in support of reuse through integration.

Structured modeling comes close to making a fetish of these objectives. For example, the core concept of a *calling sequence* (Definition 6 of Geoffrion [11]) renders interconnections explicit, and the interpretation part of genus and module paragraphs in SML help to make all model parts visible to users who may find difficulty in reading the more formalistic parts of SML. "Input" modules or interfaces are not necessary because calling sequences in effect play this role. The generic calling sequence in SML does the same thing at the level of an entire genus. Nor are "output" modules or interfaces necessary in structured modeling, because any element or genus can be called by any other (subsequent) one that needs it as input.

The above comments have to do with how structured modeling relates to some of the ideas reviewed in Section 1. What about the model integration approach of Section 2? A brief answer is that Step 2c can increase the degree of modularization, but that Step 3b tends to decrease it by increasing the level of structural connectivity. It is not clear whether the latter effect is good or bad on balance, as its negative effects on maintainability and reusability are offset by positive effects on efficiency and avoidance of redundancy.

ISSUE: To what extent can the 5-step approach of Section 2 be automated?

The approach detailed in Section 2 and illustrated in Section 3 evidently is quite labor intensive. Certain standard capabilities of ordinary word processors, like the search/replace and block move features, have obvious uses for carrying out many of the substeps, but a higher degree of automation seems essential to facilitate model integration in general.

A tree-oriented editor, otherwise known as an "outliner", would be helpful. (We note that one is available in FW/SM, the prototype structured modeling implementation.) It would be much more helpful to have a SML syntax-directed (structure) editor along the lines proposed by Vicuña [28]: it could detect or even preclude the violations of SML syntax or Schema Properties prohibited in Steps 1, 2, and 5, and could detect the violations that must be fixed at Step 4.

The diagnostic part of Step 2a lends itself particularly well to automation, as does -- although it is more difficult -- the diagnostic part of Step 4. Moreover, specialized search-and-replace capabilities for changing names, indices, and key phrases would be helpful for Steps 1b, 2a, and perhaps other steps.

A good implementation of all or even most of these possibilities would greatly facilitate the integration of structured model schemas. However, there will remain a certain amount of modeler's discretion that must be exercised at most of the steps. For example, only the modeler can decide at Step 1a exactly what structural changes are appropriate. Similar discretion is needed at the other steps, except for 3a, 2a (probably), and 3b (possibly). That discretion, incidentally, is what an integration language like that of Bradley and Clemence [6] must express.

The last-mentioned reference, and Murphy, Stohr, and Ma [23], are examples of recent work that offer intriguing prospects for automating some of what we have called "modeler's discretion" by more deeply exploiting the syntactic and semantic information available in (or attachable to) two SML schemas.

<u>ISSUE</u>: Within the context of SML, is it easier to reuse schemas per the integration approach of Section 2, or to build monolithic schemas from scratch?

I believe that reuse will be significantly more efficient than ground-up development in most cases. The amount of work that must be done with each approach in a particular application depends on the special facilities available in the modeling environment and the answer to the previous issue.

It should be possible to address this issue experimentally by counting operations (or actions, or deliberate decisions that must be made, or the amount of time consumed) for each approach for different modelers in a controlled setting. This is an attractive research topic.

There may be some modifications to SML that would reduce the total amount of work that must be done in reuse via model integration. For example, it can be a nuisance to have to avoid duplication of names for genera, modules, indices, functional and multi-valued dependencies, and symbolic parameter stems. It might be feasible to allow selected modules to limit the scope of the names defined within them, so that all external references would have to include the name of the defining module in order to gain access. Duplicate names could then be allowed because references could be unambiguous. Such an extension of SML would be well worth working out.

<u>ISSUE</u>: Within the context of FW/SM, the prototype structured modeling implementation, does structured modeling put the "right" things put in libraries in the sense of Balzer, Cheatham, and Green [1]?

Structured modeling puts model schemas, elemental detail tables, and solvers in libraries.

A schema is an analog of a "specification" in the sense of Balzer, Cheatham, and Green [1]. Putting a schema in a library seems appropriate to the extent that FW/SM facilitates revising it and integrating it with other schemas, provided that it is of possible future interest. See Sections 3.1 and 3.2 of Geoffrion [9] and the appendices of Geoffrion [14].

Putting not only a schema, but also associated elemental detail, in a library makes as much sense as putting just the schema in the library in some situations. One such occurs when the elemental detail is a commercial or private database with multiple potential applications, such as an air freight rate database that is of possible use in any logistics model involving air freight. Another such situation occurs when the aim is to set the stage for model integration within the context of the original application. Of course, there could not be many more than one reuse of elemental detail in this case.

There can be little question about the appropriateness of putting solvers in libraries, as the function they perform is useful over the entire class of models to which they are intended to apply.

While on the subject of libraries, it should be noted that a library is of little use if users cannot easily find out what is in it. This suggests that a structured model of the contents would be useful. More generally, it could be useful to have computer-based tools to support library management and access. One paper dealing specifically with libraries in the context of model management is Mannino, Greenberg, and Hong [19]. Related papers in the context of software, such as Prieto-Diaz and Freeman [26], may be inspirational.

<u>ISSUE</u>: In programming, the goals of reusability, maintainability, and modifiability usually are pursued by adhering to strict disciplines like structured programming, structured design, etc. that impose order on the otherwise chaotic process of programming. Similar goals for modeling would seem to imply the need for comparably strict disciplines that impose order on the otherwise chaotic process of modeling. Does structured modeling impose such discipline?

Yes, structured modeling does indeed impose a great deal of discipline on the modeler by comparison with what would be the case if, say, ordinary mathematics were used in the usual *ad hoc* fashion to express models. Consider:

* All definitional dependencies among model elements are supposed to be explicit (in the calling sequences).

* Circular definitional dependencies are supposed to be avoided.

* All values are supposed to have an explicit range.

* Modular structure is supposed to reflect the natural conceptual groupings of model elements.

* Model elements are supposed to be arranged in a sequence such that there are no forward definitional references.

* Model instances are supposed to be viewed in terms of a general class of model instances that are all isomorphic in a certain sense.

* Models are supposed to be sharply distinguished from problems posed on models, and both of these from solvers.

These conventions are imposed by the core concepts of structured modeling (Geoffrion [11]). SML either enforces or is consistent with all of these conventions, and lays down others as well (e.g., the strict separation of general structure from detailed data, and the provision for informal interpretation as an integral part of formal specification).

The overall intended effect of these and other conventions is to bring order to the modeling process in a way that will advance the goals of reusability, maintainability, and modifiability.

Acknowledgments

I gratefully acknowledge the helpful comments of Gordon Bradley, Dan Dolk, Sergio Maturana, Laurel Neustadter, Yao-Chuan Tsai, and Fernando Vicuña.

Supported partially by the National Science Foundation, the Office of Naval Research, and the Navy Personnel Research and Development Center (San Diego). The views contained in this report are those of the author and not of the sponsoring agencies.

REFERENCES

[1] BALZER, R., T.E. CHEATHAM, Jr., and C. GREEN 1983. "Software Technology in the 1990's: Using a New Paradigm," *Computer*, 16:11 (November), pp. 39-45.

[2] BATINI, C., M. LENZERINI, and S.B. NAVATHE 1986. "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, 18:4 (December), pp. 323-364.

[3] BLANNING, R.W. 1986. "A Relational Framework for Information Management," in E.R. McLean and H.G. Sol (eds.), *Decision Support Systems: A Decade in Perspective*, Elsevier Science Publishers B.V. (North-Holland), Amsterdam.

[4] BLANNING, R.W. 1987. "A Relational Theory of Model Management," in C.W. Holsapple and A.B. Whinston (eds.), *Decision Support Systems: Theory and Application*, Springer-Verlag.

[5] BOEHM, B.W. 1987. "Improving Software Productivity," *Computer*, 20:9 (September), pp. 43-57.

[6] BRADLEY, G.H. and R.D. CLEMENCE, Jr. 1988. "Model Integration with a Typed Executable Modeling Language," *Proceedings of the Twenty-First Hawaii International Conference on System Sciences, Vol. III*, January, pp. 403-410.

[7] COX, B.J., Jr. 1987. *Object-Oriented Programming*, Addison-Wesley, Reading, MA.

[8] FREEMAN, P. 1987. *Software Reusability*, IEEE Cat. No. EH0256-8, Computer Society Press, Washington, D.C.

[9] GEOFFRION, A. 1987. "An Introduction to Structured Modeling," *Management Science*, 33:5 (May), pp. 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, Western Management Science Institute, UCLA, 75 pages, 6/86. Last revised March, 1988. This is the version referenced here.

[10] GEOFFRION, A. 1987. "Integrated Modeling Systems," Working Paper 343, Western Management Science Institute, UCLA, 16 pages, 11/86. Last revised March, 1988. To appear in the *Proceedings of the Conference on Integrated Modeling Systems* (held at the University of Texas, Austin, October 1986).

[11] GEOFFRION, A. 1987. "The Formal Aspects of Structured Modeling," forthcoming in *Operations Research*. Available as Working Paper 346, Western Management Science Institute, UCLA, 39 pages, 5/87. Last revised August, 1988.

[12] GEOFFRION, A. 1988. "SML: A Model Definition Language for Structured Modeling," Working Paper 360, Western Management Science Institute, UCLA, 129 pages, May.

[13] GEOFFRION, A. 1988. "A 'Cyclic' Model Discussed by Blanning and Muhanna," Informal Note, UCLA, 10 pages, June 5. Revised August 4.

[14] GEOFFRION, A. 1988. "Reusing Structured Models via Model Integration," Working Paper 362, Western Management Science Institute, UCLA, 42 pages, September.

[15] GOGUEN, J.A. 1986. "Reusing and Interconnecting Software Components," *Computer*, 19:2 (February), pp. 16-28. Reprinted in [8].

[16] KARTASHEV, S. and S. KARTASHEV 1986. "Guest Editors' Introduction," *Computer*, 19:2 (February), pp. 9-13.

[17] KOTTEMANN, J.E. and D.R. DOLK 1988. "Process-Oriented Constructs for Model Integration," working paper, Naval Postgraduate School, 28 pages, March. Submitted for publication.

[18] LIANG, T.P. 1986. *Toward the Development of a Knowledge-Based Model Management System*, Ph.D. Dissertation, University of Pennsylvania.

[19] MANNINO, M., B.S. GREENBERG and S.N. HONG 1987. "Knowledge Representation for Model Libraries," Working Paper CBDA 143, Center for Business Decision Analysis, the University of Texas at Austin, August.

[20] MUHANNA, W. 1987. "Composite Models: Hierarchical Construction, Circularity, and Deadlocks," Working Paper, University of Wisconsin - Madison, October.

[21] MUHANNA, W.A. and R.A. PICK 1986. "A Systems Framework for Model Management," School of Business, Univ. of Wisconsin-Madison, December. Revised August, 1988.

[22] MUHANNA, W.A. and R.A. PICK 1988. "Composite Models in SYMMS," *Proceedings of the Twenty-First Hawaii International Conference on System Sciences, Vol. III*, January, pp. 418-427.

[23] MURPHY, F.H., E.A. STOHR and P. MA 1988. "Composition Rules for Building Linear Programming Models from Component Models," Working Paper, Information Systems Area, Graduate School of Business Administration, New York University, June 17.

[24] PARNAS, D.L. 1972. "On the Criteria To Be Used in Decomposing Systems into Modules," *Comm. ACM*, 15:12 (December), pp. 1053-1058.

[25] PRIETO-DIAZ, R. and J.M. NEIGHBORS 1986. "Module Interconnection Languages," *J. Systems and Software*, 6:4 (November), pp. 307-334. Reprinted in [8].

[26] PRIETO-DIAZ, R. and P. FREEMAN 1987. "Classifying Software for Reusability," *IEEE Software* (January), pp. 6-16. Reprinted in [8].

[27] TSAI, Y. 1987. "An Operational Approach to Model Integration Using a Structured Modeling Framework," Research Paper, Anderson Graduate School of Management, UCLA, 60 pages, December. Revised 2/88.

[28] VICUÑA, F. 1988. "A Modeling Environment for Operations Research," Ph.D. Research Proposal, UCLA, July.

[29] YOURDON, E. and L.L. CONSTANTINE 1979. *Structured Design*, Prentice-Hall, Englewood Cliffs, NJ.

[30] ZEIGLER, B. P. 1984. *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, London.