Modelling and Decision Support

Toward a formal theory of model integration

Marco Gagliardi
Department of Quantitative Methods, University of Siena, Piazza S. Francesco 17, I-53100 Siena, Italy

Cosimo Spera*
Department of Industrial and Operations Engineering, University of Michigan, 1205 Beal Avenue, Ann Arbor, MI 48105, USA
E-mail: spera@sivax.unisi.it.

The aim of this paper is to provide the first steps toward a formal theory of model integration. This is supported at least by three arguments: (a) increasing the productivity of the modeling work; (b) decreasing errors; (c) saving time and money. Of course, any formal theory has to be based on a given framework; in our case, we consider only models which satisfy the core concepts of Structured Modeling. The outline of the paper is as follows. After the motivations are pointed out, some preliminary results are given in section 2. Section 3 defines the levels of integration, while in sections 4 and 5 some examples are presented. Remarks and future extensions conclude the paper.

Keywords: Model integration, Structured Modeling.

1. Motivations for a formal theory

The definition of a specific model is conceived as a work which has to be done from scratch. Ideally, the model builder would like to construct his model by assembling, when it is possible, models previously defined, or by using models defined and tested by other people. There are two cases to consider:

- all the models to be assembled are expressed in the same definitional framework;
- the models to be assembled derive from different frameworks.

These two cases bring to different types of integration: "*deep*" integration and "*functional*" integration. This distinction is due to Geoffrion.

Muhanna and Pick have called it *structural and composition* integration [13], while Dolk and Kottemann have called it *definitional and procedural* integration. They

*On leave from the University of Siena on a fellowship from CNR, Italy. Permanent address: Department of Quantitative Methods, Piazza S. Francesco 17, I-53100 Siena, Italy (contact author).

have also proposed a "*model interconnection language*" (Dolk and Kottemann [12]), and a "*model description language*" (Muhanna and Pick [13]), to treat the functional integration.

Deep integration has been treated by Geoffrion in [10]. Here, we will deal with deep integration and try to show that models can be defined by assembling pieces of correlated sub-models. This process, to be effective, has to minimize the errors in specifying the model and has to include as much as possible automated procedures. It has to be carried on within a formal framework. We have chosen the Structured Modeling framework as defined by Geoffrion [7,9]. The main features, with respect to the assembling process, derived by Structured Modeling is *modularity*: this greatly influences the productivity of the work.

In this paper, we give some formal results and a few examples of integration among models. Our effort is to define automated procedures, which can be used to replace genera, modify definitional dependencies, set new dependencies among sub-models, etc.

These procedures check in an automated way if some of the Structured Modeling principles are violated at the end of the integration process.

We want to point out that we try to formalize this integration theory outside any model definition language. Nevertheless, our examples are given using an object-oriented language, but the obtained results hold in general.

2. Preliminary results

In the remainder of this paper, we assume that the reader is familiar with the formal theory of Structured Modeling.

Given a Structured Model M_i, let $G_i = \{g_j, j \neq 1,\dots,k_i\}$ be the set of all the genera; this can be partitioned into three disjoint sets, PC_i, A_i and FT_i, such that

$$PC_i = \{g_j \in G_i\text{: } g_j \text{ is a primitive or a compound entity genus}\},$$

$$A_i = \{g_j \in G_i\text{: } g_j \text{ is an attribute genus}\},$$

$$FT_i = \{g_j \in G_i\text{: } g_j \text{ is a function or a test genus}\}.$$

LEMMA 1

Any genus $g_j \in PC_i$ does not have references to any other genera $g_k \in (A_i \vee FT_i)$.

Proof

Primitive entity elements, by definition, have no calling sequence, therefore they do not have references to any other elements; compound entity elements, by definition, are constructed only on primitive entity and other compound entity elements. □

LEMMA 2

Any genus $g_j \in A_i$ has only references to other genera $g_k \in PC_i$.

Proof

Attribute elements, by definition, characterize only primitive and compound elements. □

Our formal theory to integrate models is developed at the level of generic structure. The following proposition proves that if the integrated graph of genera G satisfies the Structured Modeling principles, so does the elemental structure E.

PROPOSITION 1

Let E be a non-empty and finite set of elements, and let G be a set of partitions constructed on E, one for each of the five types. E is an Elemental Structure if:

(1) G satisfies generic similarity;

(2) G is a closed set;

(3) G is an acyclic set.

Proof

(a) E is not empty and finite by hypothesis.

(b) Closure (by contradiction). Suppose $e_i \in E$ has a reference in its calling sequence to $e_j \notin E$. Let e_j be an element of the genus g_j, and e_i be an element of the genus g_i. By the generic similarity property, g_i has in its calling sequence a reference to a genus g_j; but by construction $g_j \notin G$; this violates (2).

(c) Acyclicity (by contradiction). Let $S = \{e_1,\ldots,e_i\}$ be a cyclic sequence of elements belonging to E. If each e_i belongs to a different genus g_i, then the generic similarity property implies that G is cyclic. If there are two elements $e_k, e_h, k, h : 1,\ldots,i,\ k \neq h,\ e_k, e_h \in g_i$, then let us consider the sub-sequence $S_j \subset S$, $S_j = \{e_k,\ldots,e_k\}$. By the generic similarity property, there exists a sequence of genera $g_i,\ldots,g_i$, which is cyclic. This violates (3). □

Within the Structured Modeling framework, genera are grouped into modules. In the following sections, we often use particular Structured Modeling modules, which allow to identify sub-models.

DEFINITION 1: Connected module

A Structured Modeling module is *connected* if its genera and their calling sequences form a connected sub-graph.

DEFINITION 2: Sub-model

A *sub-model* is a connected module with at least one primitive entity genus.

The next definition individualizes sets of operations, which will not modify the output values of the model. These sets of operations do not modify the input data.

DEFINITION 3: Neutral set of operation

Given a model $M_i \in SM$, where SM is a set of Structured Models, we define the set T of operations to be *neutral* if the resulting model $T(M_i)$ returns the same output values when instanced with the same data of M_i.

DEFINITION 4: Neutral set of operation with respect to g_i

Given a sub-model $SubM_i \in SM$, where SM is the set of Structured Models, and a genus $g_i \in SubM_i$, we define the set T of operations to be *neutral with respect to* g_i if the resulting model $T(SubM_i)$ returns the same output values given by g_i when it is instanced with the same data of $SubM_i$ for the genera called directly or indirectly by g_i.

DEFINITION 5: Normal model

A model is called *normal* if the following conditions are satisfied:

(a) there is a 1:1 correspondence between attribute and compound genera;

(b) given a pair of matching genera, there is a 1:1 correspondence between their elements.

The reason to define a normal model is that the attribute genus index can be known through the compound genus index. This point will be much clearer when some of the integration procedures are illustrated. The graph of the elements of a normal model is shown in figure 1; dotted rectangles identify genera.

PROPOSITION 2

Given a Structured model M_i, it is always possible to construct a normal model $N(M_i)$ using the neutral set of operations N.

Proof

Consider any attribute genus $g_j \in A_i \subset M_i$. It is always possible to define a new compound entity genus, $c_k \in PC_i$, with the same calling sequence as g_j. Lemmas 1 and 2 ensure that genera which are called by an attribute genus can be called by a compound entity genus too. An isomorphic relation can be set among the elements

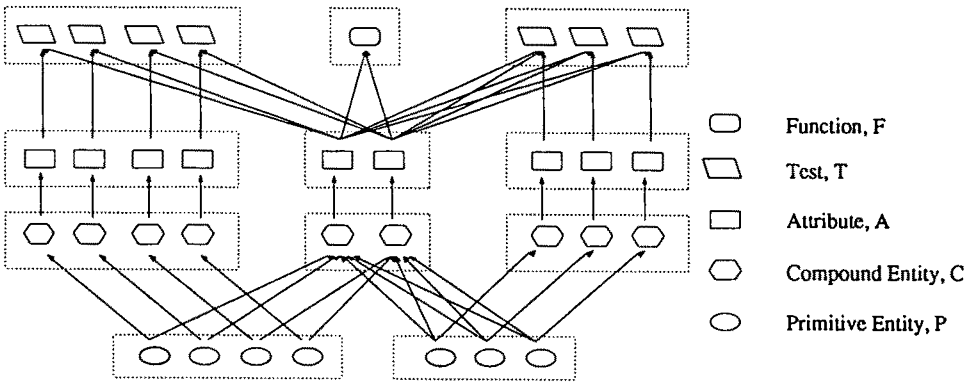


Figure 1. Graph of elements of a normal model.

of g_j and c_k: the first elements of g_j calls the first element of c_k, etc. This process is repeated for every attribute genus of M_i. The attribute genera so constructed have exactly the same number of elements and the same structure as before the structural changes; therefore, they can be instanced using the same data as previously.

Formally, if $N(M_i)$ indicates the modified model, the set $B = \{c_k, g_j\} \subset N(M_i)$ $\Leftrightarrow g_j \in M_i$ for every genus $g_k \in FT_i \subset N(M_i)$. □

Based on proposition 2, the procedure Normal is constructed. Here, CS_i indicates the calling sequence of the ith genus, i.e. the finite list of the calling sequence segments.

procedure Normal (input: M_i; output $N(M_i)$);
begin
Create a LIST of $g_j \in A_i \subset M_i$;
  while not end of LIST **do**
    Select $g_j \in$ LIST;
    if (CS_j has more than a segment) **or** (g_j does not isomorphically call a compound entity genus)
    then
      Create a genus $dummy_i \in PC_i$;
      $CS_i = CS_j$;
      CS_j = ($dummy_i$: iso);*
      /* the calling sequence of g_j is set to call the $dummy_i$ genus in an isomorphic way */
    LIST := LIST – g_j;
  end while
end.

* This notation is taken from BLOOMS grammar and has the meaning of "set an isomorphic relation among the elements of g_j and the compound entity" [2,3].

Proposition 2 and procedure Normal ensure that for each Structured Model there is a neutral set of operations N which can construct a model that returns the same values when instanced with the same data. Moreover, this set of operations can be automated.

DEFINITION 6: Index basis

An *index basis* of a normal model $N(M_i)$ is a couple of genera $B_j = \{a_j, c_j\}$, where $a_j \in A_i \subset M_i$ is an attribute genus, and c_j is the compound entity genus called by a_j. The genus a_j is called the *value component* of B_j, while the genus c_j is called the *index component*.

DEFINITION 7: Index basis set

The sets $BS_i = \{B_j, j : 1,\ldots,h\}$ containing all the index basis of $N(M_i)$ is called the *index basis set*.

DEFINITION 8: Index function

Suppose L to be a language for the definition of Structured Models. An *index function* $i(g_j)$ is a rule which associates to every genus $g_j \in N(M_i)$, expressed using the language L, the cardinality of its generic index t-uple.*

As an example, given a genus g_i indexed by (j, k, l), its index function $i(g_i)$ returns as value 3.

Definitions 6, 7 and 8 are related to the indices' management; they are not language dependent.

3. Integration levels

As we pointed out, our integration theory will be developed working at the level of the graph of genera. In this section, we define three levels of integration and characterize some simple operations on the genera graph, which are called elementary operations. They form the basis to construct more complex procedures used to integrate models.

Level 1 All the procedures are automated. This means that the user selects the input models and the genera to be integrated, and the output integrated model is automatically produced.

*This concept is taken from Geoffrion's SML language. Nevertheless, it is a general concept which can be easily extended to every modeling language [6].

Level 2 The user selects the input models and the order of integration among the genera, and the output integrated model is automatically produced.

Level 3 The user selects the input models, the genera to be integrated and formulates the steps necessary to integrate. The output integrated model is not automatically produced. At this level, the user needs to create the integration procedures, which cannot have any generality since the integration steps can vary according to the situation.

The goal is to try to understand how many integration procedures can be on the first two levels, and to create for the third level an interface language which allows users to define ad hoc integration procedures.

This strategy, on the one hand, tries to take into account the need of automated procedures, which can be used in some context to increase the productivity of the model builders, and to decrease the number of possible errors; on the other hand, it gives a flexible tool to successfully deal with the variety of situations which occur in model integration.

3.1. ELEMENTARY OPERATIONS

Let us consider the set G. This set contains the graphs of genera $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ of all Structured Models. $\mathcal{V}_i$ is the set of typed nodes, $\mathcal{V}_i = (1,\dots,n_i)$, and $\mathcal{E}_i$ is the set of arcs (i, j), $i: 1,\dots,n_i, j: 1,\dots,n_i$, $i \neq j$, which represent the definitional dependencies between genera (nodes). Elementary operations can be defined both on arcs and nodes.

3.1.1. Operations on arcs

These operations influence the definitional dependencies among genera, both in the case where they are executed on a single graph and in the case where they are executed on two or more graphs. There are only two elementary operations on arcs:

(1) add;

(2) delete.

These operations are formalized in the following procedures. As before, CS_i indicates the calling sequence of the genus g_i.

```
procedure Add_Arc (Input: gi, gj; Output: (gi, gj));
/* Create a new direct arc from gi to gj. */
/* The symbol \\ means "append an element to the list" */
begin
   CSj := CSj \\ gi;
end.
```

```
procedure Delete_Arc (Input: gi, gj; Output: null);
/* Delete an existing arc from gi to gj.*/
begin
   CSj := CSj - gi;
end.
```

Add_Arc is an operation not always allowed. In fact, lemmas 1 and 2 establish the constraint for this procedure. Table 1 shows the allowed operations. (P, C, A, F and T indicate the types nodes of Structured Modeling.)

There are no limitations when a Delete_Arc operation is called. It is clear that these elementary operations are not closed on $\mathcal{G}$.

Table 1

Add arc.

g_j \ g_i	P	C	A	F	T
P					
C	×	×			
A	×	×			
F	×	×	×	×	×
T	×	×	×	×	×

A variety of procedures can be constructed combining these elementary operations Add_Arc and Delete_Arc. Let us show some of them.

Given three genera g_i, g_j and g_k, where g_i, $g_k \in M_1$ and $g_j \in (M_1 \vee M_2)$, two situations can arise:

(a) there is an arc (g_k, g_i) in M_1;

(b) there is an arc (g_i, g_k) in M_1.

Figures 2(a) and 2(b) illustrate these situations.

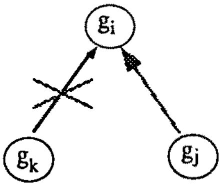


Figure 2(a). The added arc is (g_j, g_i), while the deleted arc is (g_k, g_i). This implies that the calling sequence segment of g_i having a references to g_k is modified to g_j.

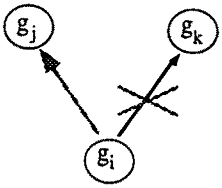


Figure 2(b). The added arc is (g_i, g_j), while the deleted arc is (g_i, g_k). This implies that the calling sequence segment, which calls g_i, of the genus g_j is deleted, and the calling sequence of g_k is set to have a reference to g_i.

The following procedures formalize the replacement operations.

```
procedure Replace_In_Arc (Input: gi, gk, gj; output: (gj, gi));
begin
   Add_Arc (gj, gi; (gj, gi));
   Delete_Arc (gk, gi; null);
end.

procedure Replace_Out_Arc (Input: gi, gk, gj; Output: (gi, gj));
begin
   Add_Arc (gi, gj, (gi, gj));
   Delete_Arc (gi, gk; null);
end.
```

Tables 2 and 3 show the feasible node replacement using the above procedures. Rows and columns indicate the types of g_k and g_j genera; a × at the interesection means that the replacement of the nodes is always possible, while capital letters indicate that the operation is possible only for specific types of the g_i nodes.

Table 2

Replace in-arc.

g_j \ g_k	P	C	A	F	T
P	×	×	×	×	×
C	×	×	×	×	×
A	×	×	×	×	×
F	F, T	F, T	×	×	×
T	F, T	F, T	×	×	×

Table 3

Replace out-arc.

g_j \ g_k	P	C	A	F	T
P					
C		×	×	×	×
A		×	×	×	×
F		P, C	P, C	×	×
T		P, C	P, C	×	×

The procedures described above are not closed on $\mathcal{G}$.

3.1.2. Operations on nodes

These operations allow to create new genera or delete existing ones. We analyze two elementary operations on nodes:

(a) add a node;

(b) delete a node.

The Add procedure is difficult to formalize, since genera contain the semantic information of the Structured Model and the operation of adding a genus (i.e. a node) has to be performed using a definition language. To add a genus g_i implies the definition of the arcs (g_k, g_i), where $g_k \in M_i$, $k: 1,\dots,n_i$, $k \neq i$, are the genera called by g_i. Therefore, the procedure which formalizes the (a) elementary operation on nodes uses the Add_Arc procedure.

When a genus $g_i \in M_i$ is deleted, the arcs (g_j, g_i) and (g_i, g_k), where $g_j, g_k \in M_i$, $j: 1,\dots,n_i$, $k: 1,\dots,n_i$, $j, k \neq i$, have to be deleted. Therefore, the procedure which formalizes the (b) elementary operation on nodes uses the Delete_Arc procedure.

```
procedure Add_Node (input: gi);
begin
    {define a new genus with a definitional language}
    /* This step is not formalized, since this has to be done using a definition
    language */
    Create a LIST of gk;
    /* genera gk are called by gi */
    while not end of LIST do
        Select gk from LIST;
        Add_Arc (gk, gi; (gk, gi));
        LIST := LIST - gk;
    end while
end.

procedure Delete_Node (input: gi);
begin
    Create a LIST_OUT of arcs (gi, gk);
    while not end of LIST_OUT do
        Select (gi, gk) from LIST_OUT;
        Delete_Arc (gi, gk; null);
        LIST_OUT := LIST_OUT - (gi, gk);
    end while;
    Create a LIST_IN of arcs (gj, gi);
    while not end of LIST_IN do
        Select (gj, gi) from LIST_IN;
        Delete_Arc (gj, gi; null);
        LIST_IN := LIST_IN - (gj, gi);
    end while;
    Delete the genus gi;
end.
```

The elementary operations on nodes can always be executed. They are not closed on $\mathcal{G}$.

The elementary operations described above are not exhaustive. Nevertheless, we have defined generic operations which are easy to assemble to create a large variety of procedures.

Let us prove that under defined conditions, a set of an arbitrary number of Structured Models can be integrated using elementary operations or combinations of them.

3.2. CLOSED SETS OF OPERATIONS

We give the definition of a closed set of operations, which is used in the following proposition.

DEFINITION 9: Closed set of operations

A set of elementary operations E is *closed* if $E(M_1,\ldots,M_n) = M^* \in SM$ for every $M_i \in SM$, $i:1,\ldots,n$, $n \geq 1$.

As an example, we rewrite the Normal procedure such that it is formed by elementary operations which are a closed set.

```
procedure Normal (input: Mi; output N(Mi));
begin
   Create a LIST of gj ∈ Ai ⊂ Mi;
   while not (end of LIST) do
      Select gj fro LIST;
      if (CSj has more than a segment) or (gj does not isomorphically call a
      compound entity genus)
      then
         Add_Node (dummyi);
         Create a LIST_ARC of arcs (gk, gj);
         while not (end of LIST_ARC) do
            Select (gk, gj) from LIST_ARC;
            Replace_Out_Arc (gk, gj, dummyi; (gk, dummyi);
            LIST_ARC := LIST_ARC – (gk, gj);
         end while;
         Add_Arc (dummyi, gj, (dummyi, gj));
         LIST := LIST – gj;
   end while
end.
```

The following proposition ensures that the elements $M_1,\ldots,M_n \in SM$, with $n \geq 2$, can be integrated using closed sets of operations.

PROPOSITION 3

Given $M_1,\ldots,M_n \in SM$, with $n \geq 2$, it is possible to create an integrated Structured Model M_k using a set $\{E_1,\ldots,E_k\}$ of closed sets of operations.

Proof

Trivial by recursive application of definition 9. □

In the following, we always use integration procedures which form a closed set of operations.

4. Level 1 integration: some results

In this section, we look at procedures defined to be on the first level of integration.

To show an example of the first level of integration, we need to introduce the definition of a function sub-model, which is a particular Structured Model. The goal is to select a function genus which can automatically replace an attribute genus.

DEFINITION 10: Function sub-model

A Structured Model $SubM_i(f)$ is called a *function sub-model* if it satisfies the following conditions:

(a) $SubM_i(f)$ is a normal model.

(b) $SubM_i(f)$ has at least one function genus $f \in FT_i$ which is a singleton.*

The following procedure, Create_Function_Submodel, needs as input a model M_i and a singleton genus $f \in FT_i \subset M_i$, and produces as output a function sub-model. This procedure is closed.

```
procedure Create_Function_Submodel (input: Mi, f; output: SubMi (f));
/* Modify Mi into a function sub-model SubMi (f) */
begin
   /* step I. "Normalize the model" */
   Normal (Mi; N(Mi));
   /* step II. "Merge functions" */
   Create a LIST of arcs (gi, f);
   while not end of LIST do
       Select (gi, f) from LIST;
       Select gi from (gi, f);
```

*This has the meaning "composed of a single element" [6].

```
      if (g_i ∈ FT_i)
      then
         /* a */ Create a LIST_A of arcs (g_j, g_i);
                 while not end of LIST_A do
                    Select (g_j, g_i) from LIST_A;
                    Replace_Out_Arc (g_j, g_i, f; (g_j, f));
                    LIST_A := LIST_A – (g_j, g_i);
                    if (g_j ∈ FT_i) and (g_j ∉ LIST) then
                       LIST := LIST \\ g_j;
                 end while;
         /* b */ Replace into the rule of f the value field of g_i with its rule;
                 /* This does not involve the graph structure */
         /* c */ Delete_Node (g_i);
      LIST := LIST – g_i;
   end while;
   /* step III. "Delete genera having no influence on f" */
   Create a LIST of g_j ∈ M_i;
   while not end of LIST do
      Select g_j from LIST;
      if (g_j ∈ FT_i and g_j ≠ f) then
         Delete_Node (g_j);
      if (g_j ∈ A_i ∪ PC_i and g_j is not called directly or indirectly by f) then
         Delete_Node (g_j);
      LIST := LIST – g_j;
   end while
end.
```

The next proposition ensures that the Create_Function_Submodel procedure creates a function sub-model.

PROPOSITION 4

Given a Structured Model M_i and an arbitrary singleton function genus $f \in FT_i \subset M_i$, there exists a transformation T, which is a neutral set of operations with respect to f, such that

$$T(M_i) = Sub\,M_i(f).$$

Proof

By applying the Create_Function_Submodel procedure which defines the procedure T. □

Let us show how a function genus f can be reused as an input parameter for other models. The genus f *replaces* the attribute genus g_i, if all its dependencies can be addressed to f. The automation is possible since genus f is a singleton. In fact, any function depending on the replaced attribute genus g_i does not need to be modified since the index function of f is set equal to the index function of the replaced genus g_i by the integration process.

Suppose we have two models M_1 and M_2, and we want to replace the genus $g_i \in A_1 \subset M_1$ with the computed value given by the genus $f \in FT_2 \subset M_2$. This goal is achieved by applying the following procedure (the symbol $[M_1, SubM_2]$ means the integrated output model):

```
procedure Reuse (input: M1, M2, gi, f; output: [N(M1), SubM2(f)]);
/* Integrate M1 and M2. gi is replaced by f */
begin
   /* Step I. "Changes in M2" */
   Create_Function_Submodel (M2, f; SubM2(f));
   Normal (M1; N(M1));
   Select {Dummy, gi} ⊂ N(M1);
   /* To select the index basis */
   Create a LIST of genera gi ∈ A2 ∪ PC2 ⊂ SubM2(f);
   while not (end of LIST) do
      Select gj from LIST;
      if (gj, f) then Add_Arc (Dummy, gj; (Dummy, gj));
      /* Add to the calling sequence of gj the calling sequence of gi */
      LIST := LIST − gj;
   end while;
   /* Step II. "Changes in M1" */
   Create a LIST of genera fi ∈ FT1 ⊂ N(M1);
   while not (end of LIST) do
      Select fi from LIST;
      if (gi, fi) then
         Replace_In_Arc (fi, gi, f; (f, fi));
         /* Substitute gi with f in the calling sequence of fi; */
      LIST := LIST − fi;
   end while;
   /* Step III. "Delete attribute genus" */
   Delete_Node (gi)
end.
```

PROPOSITION 5

Given two Structured Models M_1 and M_2, it is always possible to replace the attribute genus $g_i \in A_1 \subset M_1$ with a singleton function genus $f \in FT_2 \subset M_2$. The result is a Structured Model.

Proof

By applying the procedure Reuse, we obtain as a result the model $[\mathrm{N(M_1)}, \mathrm{SubM_2}]$. Its graph of genera must be finite, closed and acyclic (the non-emptiness is obvious).

(a) *Finiteness*. Step III guarantees that the number of genera of $[\mathrm{N(M_1)}, \mathrm{SubM_2}]$ is equal to the number of genera of $(N(M)_1 \cup SubM_2(f))$ minus the deleted genus g_i.

(b) *Closure*. By steps I and II, there is at least one genus of $N(M_1)$ calling a genus of $SubM_2(f)$ and at least one genus of $SubM_2(f)$ calling a genus of $N(M_1)$. From the closure of $N(M_1)$ and $SubM_2(f)$, the closure of $[\mathrm{N(M_1)}, \mathrm{SubM_2}]$ follows.

(c) *Acyclity* (by contradiction). Let us consider a cyclic sequence of genera $G^* \subseteq [\mathrm{N(M_1)}, \mathrm{SubM_2}]$. By construction, it is as follows:

$$\{\ldots, g_j \in A_2 \cup PC_2 \subset SubM_2(f), f, \ldots\}.$$

The genus g_h following f in the sequence is necessarily $g_h \in FT_1$, while the genus g_1 preceding g_j is necessarily $g_1 \in PC_2$. Lemma 1 states that there are no direct or indirect references from compound and primitive entity genera to function and test genera. Therefore, G^* cannot be cyclic. □

Figure 3 shows how an arbitrary model M_1 is integrated with an arbitrary sub-model $SubM_2$. The values supplied by the user in the attribute genus g_i are replaced by the computed value with the rule defined in the function genus f.

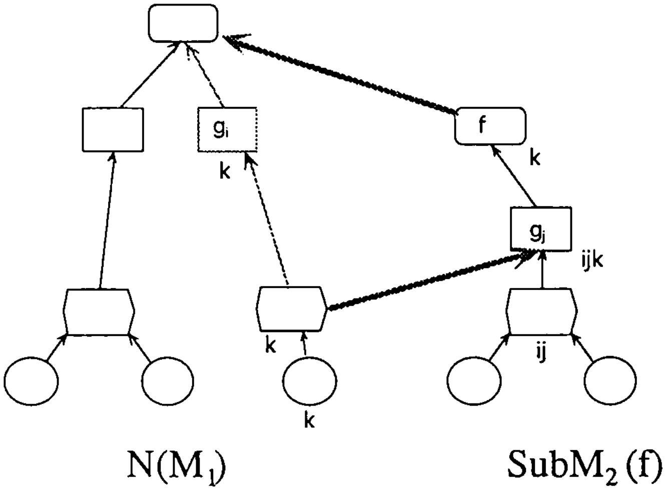


Figure 3. A hypothetical example of the Reuse procedure.

Classical Transportation Model

```
Genus PLANT primitive
/* there are some plants */
feature
    Label : string;
show Label

Genus SUP attribute
/* each plant has a given supply */
call (PLANT : iso);
feature
    sup : real+;
show sup

Genus LINK compound
/* There are links between plant and
customer */
call (CUST : one;
PLANT : one);
feature
    label : string;
    connect (PLANT, CUST)
    require  (PLANT; CUST)
covered;
    /* every plant has at least an outgoing
    link; every cust has at least an ingoing
    link */
show label

Genus T : DEM test
/* are the demand constraints satisfied? */
call (FLOW : iso
 (CUST.INDEX); DEM : iso);
feature
    dem_test : boolean is
    result := SUM
    [SUP.INDEX] flow = dem;
show dem test

Genus CUST primitive
/* there are some customers */
feature
    Label : string;
show Label

Genus DEM attribute
/* each customer has a given demand */
call (CUST : iso);
feature
    dem : real+
show dem

Genus FLOW variable
attribute
/* each link has a flow */
call (LINK : iso);
feature
    flow : real+;
show flow

Genus COST attribute
/* each link has a given cost */
call (LINK : iso);
feature
    cost : real+;
show cost

Genus T : SUP test
/* are the supply constraints satisfied? */
call (FLOW : iso (PLANT.INDEX);
SUP : iso);
feature
    sup_test : bollean is
    result := SUM
    [DEM.INDEX] flow ≤ sup;
show sup_test

Genus $ function
/* there is a computed total cost */
call (COST : all; FLOW : all);
feature
    totcost : real is
    result := SUM [LINK.INDEX] cost * flow;
show totcost
```

Exponential Smoothing Model

```
Genus TIME primitive
/* there are some times */
feature
    Label : string;
show Label

Genus P1 primitive
/* there are some primitive entities */
feature
    Label : string;
show Label

Genus ALPHA attribute
/* there is a smoothing constant for all
primitive entities */
call (P1 : all);
feature
    alpha : real+;
    invariant 0 ≤ alpha ≤ 1;
show alpha

Genus DEMAND attribute
/* there is a given demand for all primitive
entities at each time */
call (P1 : all; TIME : iso);
feature
    dem : real+;
show dem

Genus EXPONENTIAL function
call (ALPHA: all;
DEMAND: all (TIME.INDEX));
feature
    exp: real is result :=
    (IF TIME.INDEX > 1
    THEN
        alpha * dem +
        (1-alpha) *
        exp.TIME.INDEX-1.
    ELSE dem);
show exp

Genus SMOOTHED function
call (ALPHA : all;
EXPONENTIAL : all (TIME.INDEX));
feature
    smoothed : real is result :=
    (IF TIME.INDEX > 2
    THEN
        alpha *
        (exp.TIME.INDEX –
        exp.TIME.INDEX – 1) +
        (1-alpha) *
        smoothed.TIME.INDEX – 1.
    ELSE
        (IF TIME.INDEX = 2
        THEN exp.2 – exp.1
        ELSE 0));
show smoothed

Genus FORECAST function
call (ALPHA : all; EXPONENTIAL : last; SMOOTHED : last);
feature
    for : real is result := exp + smoothed / alpha;
show for
```

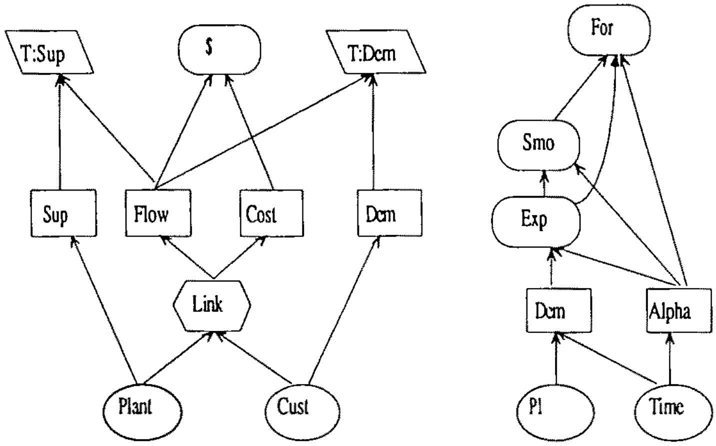


Figure 4. The two models before integration.

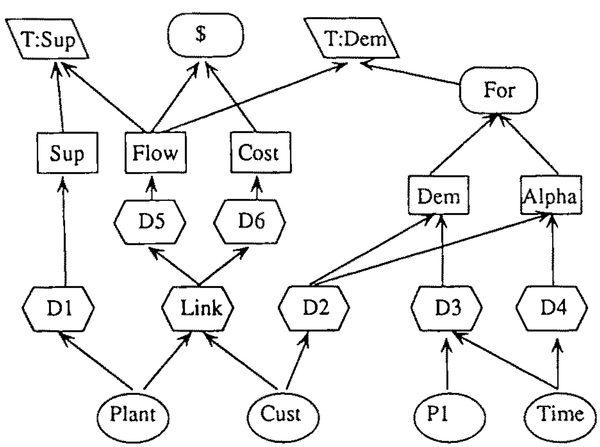


Figure 5. The integrated model.

We wish to point out that the result is not dependent on a particular model definition language for Structured Modeling. Effective integration procedures need to be defined as parts of a Model Management System, and to be consistent with the language used in the system.

We now give an example of model integration using the Reuse procedure. The example is quoted from [10]. A computer-forecasted value by an Exponential Smoothing Model replaces a given demand value in a Classical Transportation Model. The models expressed using the Object-Oriented language BLOOMS [4, 6] are given in the preceding pages.

We replace the given values of the attribute genus DEM belonging to the **Classical Transportation Model** with the computed value of the singleton function genus Forecast belonging to the **Exponential Smoothing Model**. The Reuse procedure is called with the following parameters:

Reuse (Classical Transportation Model, Exponential Smoothing Model, Dem, Forecast; Integrated Model);

Figure 4 shows the graphs of genera of the models before the integration. In figure 5 the integrated model is given. $D_1,\ldots,D_6$ are the dummy compound entity genera created by the Normal procedure executed in step 1 of Reuse.

The modified genera are presented below.

Genera modified in classical Transportation Model

Genus D1 **Compound**
Call (PLANT : **iso**);
Feature
  Label : **String**;
Show label

Genus SUP **attribute**
/* each plant has a given supply */
call (D1 : **iso**)
feature
  sup : **real+**;
show sup

Genus DEM **attribute**
/* each customer has a given demand */
call (D2 : **iso**);
feature
  dem : **real+**;
show dem

Genus D2 **Compound**
Call (CUST : **iso**);
Feature
  Label : **String**;
Show label

Genus T : DEM **Test**
/* are the demand constraints satisfied? */
call (FLOW : **iso** (CUST.INDEX);
FORECAST : **iso**);
feature
  dem_test : **boolean is result :=**
  sum [SUP.INDEX]
flow = for;
show dem_test

Genera modified in Exponential Smoothing Model

Genus D3 **Compound**
call (P1 : **all**; TIME : **iso**);
Feature
  Label : **String**;
Show label

Genus D4 **Compound**
Call (TIME : **iso**);
Feature
  Label : **String**;
Show label

Genus ALPHA **attribute**
/* there is a smoothing constant for all primitive entities */
call (D4 : **all**; D2 : **iso**);
/* Because DEM genus in Transportation model has an isomorphic call to D2 */
feature
  alpha : **real+**;
  invariant $0 \le$ alpha ≤ 1;
show alpha

Genus DEMAND **attribute**
/* there is a given demand for all primitive entities at each time */
call (D3 : **iso**; D2 : **iso**);
feature
  dem : **real+**;
show dem

Now we extend the previous results in order to allow an automatic replacement of an attribute genus $g_i \in M_i$ with a non-singleton function genus $f_j \in M_j$. The following propositions state the conditions for this action.

PROPOSITION 6

Given two normal models $N(M_i)$ and $N(M_j)$, the integrated model $[N(M_i), N(M_j)]$ obtained by replacing the input parameter $g_i \in A_i \subset N(M_i)$ with the output parameter $f_j \in FT_j \subset N(M_j)$ is a Structured Model if $i(g_i) = i(f_j)$.

Proof

The proof follows the same lines as in proposition 5. The necessary condition given by the equality of the index function leads to an analogy with the singleton case. □

PROPOSITION 7

Given a normal model $N(M_i)$, it is possible to replace the input parameter $g_i \in A_i$ with the output parameter $f_j \in FT_j$ if

$$i(g_i) = i(f_i); \tag{4.1}$$

f_i does not have direct or indirect definitional dependencies on any genus having direct or indirect definitional dependencies on g_i. (4.2)

Proof

The graph of genera after the replacement has to be (a) finite, (b) non-empty, (c) closed, and (d) acyclic. (a), (b) and (c) hold by construction. (d) is proved by contradiction. If a cyclic sequence is created by the replacement of the genera, it has to be as

$$\{g_1, \ldots, f_i, g_k, \ldots, g_1\},$$

where g_k had a definitional dependence on the replaced genus g_i.

Before, f_i had a definitional dependence on g_l, but g_l had an indirect definitional dependence on g_i. This violates (4.2). □

Based on the results of propositions 6 and 7, the following procedures can be constructed. The input parameters are an index basis $B_i \in N(M_i)$ and a function genus $f_j \in N(M_j)$, where $N(M_i)$ can coincide with $N(M_j)$; the output is a Structured Model $[N(M_i), N(M_j)]$. The procedure halts if conditions (4.1) and (4.2) do not hold.

```
procedure Use (Input: N(Mi), N(Mj), Bi, fj; Output: [N(Mi), N(Mj)]);
begin
   /* Step I: Examine if condition (4.1) is satisfied */
   Select gi ∈ Bi;
   Compute i(gi);
   Compute i(fj);
   if i(gi) ≠ i(fj) then exit;
   /* Step II: Examine if condition (4.2) is satisfied */
   Create a LIST of genera gh having direct or indirect definitional dependencies on
   gi;
   while not end of LIST do
      Select gh from LIST;
      if fj has direct or indirect dependence on gh then exit;
      LIST := LIST - gh;
   and while;
   /* Step III: Replace gi with fj */
   Create a LIST of genera gh ∈ FTi;
   while not end of LIST do
      Select gh from LIST;
      if (gi, gh) then
         Replace_In_Arc (gh, gi, fj; (fj, gh));
         /* Substitute the reference to gi with a references to fj */;
      LIST := LIST - gh;
   end while;
end.
```

5. Level 2 integration: some results

In this section, we look at procedures defined to be on the second level of integration. This means that, given a couple of genera $\{g_i \in M_i, g_k \in M_k\}$, the order of integration has to be set by the user, i.e. the user decides if g_i replaces g_h or vice versa.

Here, we present two integration procedures, which need to be applied to normal models. The first allows the user to replace any definitional dependence to an attribute genus, with definitional dependence to another attribute genus; the second procedure does the same replacement on the index components of two index bases. The input and the output parameter of the procedures are the same; they need the index bases $B_i \in N(M_i)$ and $B_j \in N(M_j)$ ($N(M_i)$ can coincide with $N(M_j)$), and return a Structured Model $[N(M_i), N(M_j)]$.

```
procedure Replace_Attribute (input: N(Mi), N(Mj), Bi, Bj; Output: [N(Mi), N(Mj)]);
begin
    Select ai ∈ Bi;
    Select ci ∈ Bi;
    Create a LIST of genera gh ∈ FTi;
    while not end of LIST do
        Select gh from LIST;
        /* Substitute ai ∈ Bi with aj ∈ Bj in the calling sequence of gh */;
        Replace_In_Arc (gh, ai, aj; (aj, gh));
        LIST := LIST − gh;
    end while;
    Delete_Node (ai);
    Delete_Node (ci);
end.
```

```
procedure Replace_Index_Component (Input: N(Mi), N(Mj), Bi, Bj; Output: [N(Mi), N(Mj)]);
begin
    Select ci, ai ∈ Bi, cj ∈ Bj;
    /* Substitute ci with cj in the calling sequence of ai */;
    Replace_In_Arc (ai, ci, cj; (cj, ai));
    Delete_Node (ci);
end.
```

The following propositions ensure that both Replace_Attribute and Replace_Index_Component are closed procedures.

PROPOSITION 8

The Replace_Attribute procedure is closed under *SM*.

Proof

Given the input parameters of the procedure, two situations can arise:

(1) $N(M_i)$ and $N(M_j)$ are two separate models;

(2) $N(M_i)$ and $N(M_j)$ coincide.

(1) The graph of genera of the integrated model $[N(M_i), N(M_j)]$ has to be (a) non-empty, (b) finite, (c) closed, and (d) acyclic. (a), (b) and (c) hold be construction. (d) holds by lemmas 1 and 2. Therefore, Replace_Attribute returns a Structured Model and, since the procedure is composed of elementary operations, it is a closed procedure.

(2) In this case, the Replace_Attribute procedure returns a modified Structured Model $[N(M_i), N(M_j)]$. The proof follows as in (1). □

PROPOSITION 9

The Replace_Index_Component procedure is closed under *SM*.

Proof

The proof follows the same lines as in proposition 8. □

We give an example of integration partially quoted from [10]. We show that the integration can be carried out using first and second level integration procedures.

There are four Structured Models:

Financial (FIN). This model computes the net income *N*, given the price *P*, the sales volume *V*, and the manufacturing expenses *E* of a product *PROD*.

Marketing (MKT). This model computes the sales volume *V*, given the price *P* of a product *PROD*.

Mark-up (MAR). This model computes the mark-up *M*, given the price *P*, the sales volume *V*, and the manufacturing expenses *E* of a product *PROD*.

Manufacturing (MFG). This model computes the manufacturing expense *E*, given the cost per unit *U* and the sales volume *V* of a product *PROD*.

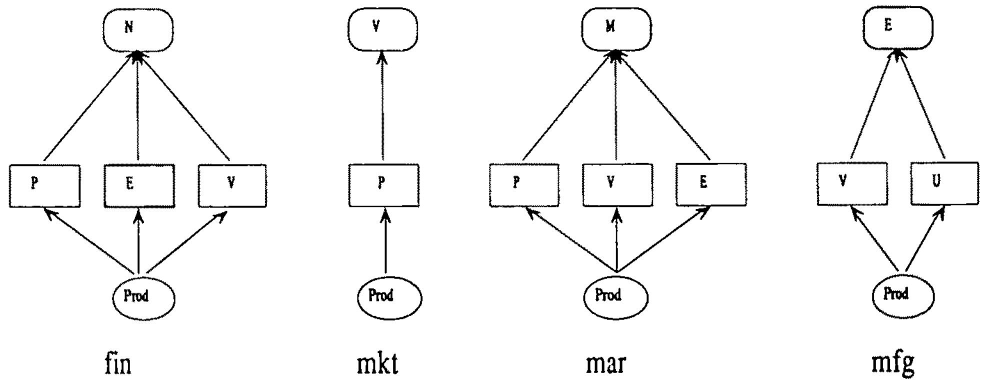


Figure 6. The four models to be integrated.

The goal is to create an integrated model which has the values supplied by the user replaced by the computed ones. This action has to satisfy all the theoretical requirements. Figure 6 shows the graph of genera of the "starting" models.

The definitions of the models expressed in BLOOMS are given below.

MARKET MODEL

Genus PROD_mkt **primitive**
/* There are some Products */
feature
Product_Label : **string**;
show Product_Label

Genus P_mkt **attribute**
/* Each Product has a given price P */
call (PROD_mkt : **iso**);
feature
Product_Price : **real+**;
show Product_Price

Genus V_mkt **function**
/* Each product has a computed sales volume */
call (P_mkt : **iso**);
feature
Sales_Volume : **real is result :=**
800000 – 4400 * Product_Price;
show Sales Volume

Genus E_mar **attribute**
/* each product has a given manufacturing expense */
call (PROD_mar : **iso**);
feature
Manufacturing_Expense : **real+**;
show Manufacturing_Expense

Genus M_mar **function**
/* there is a computed mark-up for every product */
call (P_mar : **iso**; V_mar : **iso**;
E_mar : **iso**);
feature
Markup : **real is result :=**
Product_Price * Sales_Volume /
Manufacturing_Expense;
show Markup

MARK-UP MODEL

Genus PROD_mar **primitive**
/* There are some Products */
feature
Product_Label : **string**;
show Product_Label

Genus P_mar **attribute**
/* Each Product has a given price P */
call (PROD_mar : **iso**);
feature
Product_Price : **real+**;
show Product_Price

Genus V_mar **attribute**
/* Each product has a given sales volume */
call (P_mar : **iso**);
feature
Sales_Volume : **real+**;
show Sales_Volume

→

MANUFACTURING MODEL

Genus PROD_mfg **primitive**
/* There are some Products */
feature
Product_Label : **string**;
show Product_Label

Genus U_mfg **attribute**
/* each product has a given unit cost */
call (PROD_mfg : **iso**);
feature
Unit_Cost : **real+**;
show Unit_Cost

Genus V_mfg **attribute**
/* each product has a given sales volume */
call (PROD_mfg : **iso**);
feature
Sales_Volume : **real+**;
show Sales_Volume

→

```
Genus E_mfg function
/* there is a computed manufacturing
expense for every product */
call (U_mfg : iso; V_mfg : iso);
feature
  Manufacturing_Expense : real is
  result := 1000000 + Unit_Cost *
  Sales_Volume;
show Manufacturing_Expense
```

```
FINANCIAL MODEL

Genus PROD_fin primitive
/* There are some Products */
feature
  Product_Label : string;
show Product_Label

Genus P_fin attribute
/* Each Product has a given price P */
call (PROD_fin : iso);
feature
  Product_Price : real+;
show Product_Price

Genus V_fin attribute
/* every product has a given sales
volume */
call (PROD_fin : iso);
feature
  Sales_Volume : real+;
show Sales_Volume

Genus E_fin attribute
/* every product has a given manu-
facturing expense */
call (PROD_fin : iso);
feature
  Manufacturing_Expense : real+;
show Manufacturing_Expense

Genus N_fin function
/* there is a computed net income for
every product */
call (P_fin : iso; V_fin : iso; E_fin : iso);
feature
  Net_Income : real is result :=
  Product_Price * Sales_Volume –
  Manufacturing_Expense;
show Net_Income
```

Step I: *Model normalization*

The four models are normalized using the Normal procedure as indicated below:

Normal (Fin; N(Fin));
Normal (Mkt; N(Mkt));
Normal (Mar; N(Mar));
Normal (Mfg; N(Mfg));

This step is necessary because some first-level, and all second-level, procedures require as input normal models (see figure 7).

Step II: *Choose any two models and integrate them using first- and second-level procedures*

Let us consider the models *N(Mkt)* and *N(Mar)*. Both show the attribute genera Ps (which are the prices of the products). The attributes Ps correspond to P_mar and P_mkt, as indicated in the BLOOMS formulation of the models. We kept this convention also for the other genera. To replace P_mkt with P_mar, we call the Replace_Attribute procedure as indicated below:

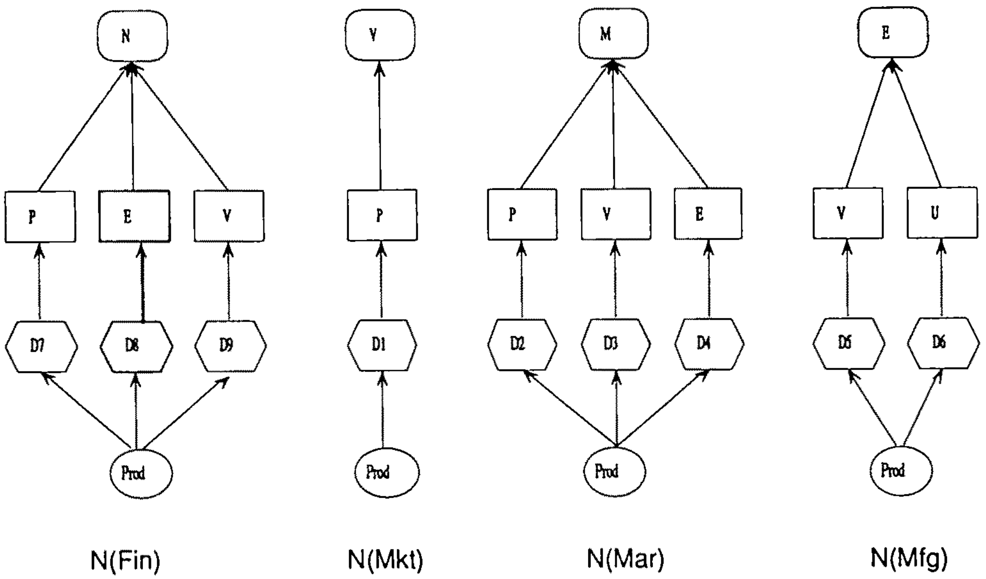


Figure 7. The normalized models.

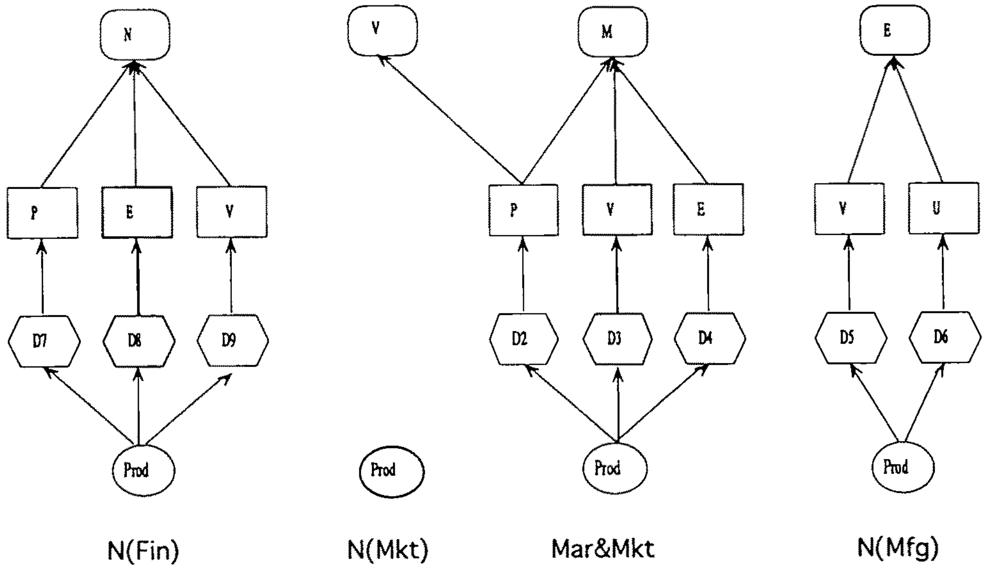


Figure 8. The models after the Replace_Attribute procedure execution.

Replace_Attribute (N(mkt), N(Mar), [D1,P_mkt], [D2,P_mar]; [N(Mkt), N(Mar)]);

This action produces the integrated Structured Model depicted in figure 8.

For brevity of notation, we write Mar&Mkt instead of [N(Mkt), N(Mar)]. Now the function genus V_mkt has to replace the attribute genus V_mar. This can be done if proposition 7 holds. In this case, this task is accomplished by the Use procedure:

Use (Mar&Mkt, Mar&Mkt, [D3,V_mar], V_mkt; Mar&Mkt);

Note that Use works on a single model. The result is a normal model. At the end of step II, *N(Mkt)* has only the primitive entiry Prod_mkt, and since the model has no meaning, it can be deleted (see figure 9).

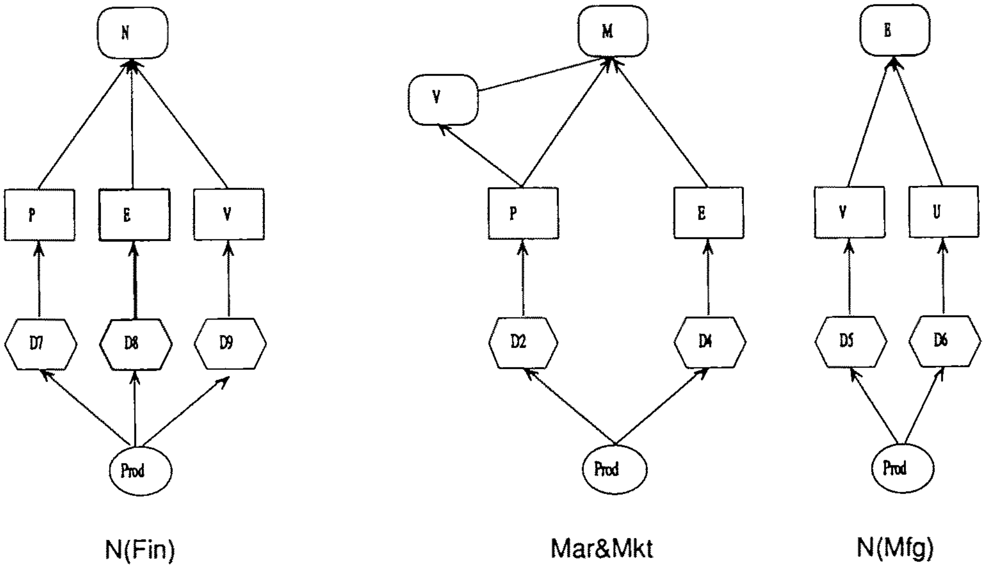


Figure 9. The situation after step II.

Step III: *Starting with the result obtained at step II, choose two models and integrate them using first- and second-level procedures.*

Let us consider the models Mar&Mkt and *N(Mfg)*. The goal is to replace the given values of the attribute genus V_mfg with the computed values of the function genus V_mkt and the given values of the attribute genus E_mar with the computed values of the function genus E_mfg. To proceed, we need the index functions of the attribute genera and of the function genera which replace them to be equal:

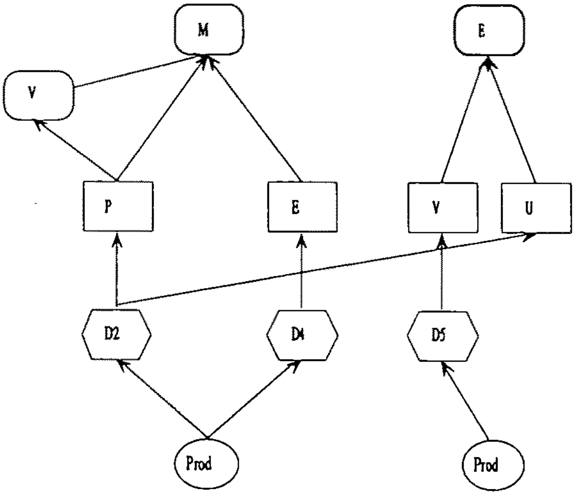
$$i(\text{V\_mfg}) = i(\text{V\_mkt}), \tag{5.1}$$

$$i(\text{E\_mar}) = i(\text{E\_mfg}). \tag{5.2}$$

This can be accomplished by setting the P_mar and the U_mfg attribute genera to call the same index component. Therefore, we use the Replace_Index_Component procedure as below:

Replace_Index_Component (N(Mfg), Mar&Mkt, [D6, U_mfg], [D2,P_mar]; [N(Mfg), Mar&Mkt])

This action produces the integrated model in figure 10. For short, we write Mar&Mkt&Mfg instead of [N(Mfg), Mar&Mkt].



Mar&Mkt&Mfg

Figure 10. The integrated model after the Replace_Index_Component procedure.

Now the V_mkt function genus can replace the V_mfg attribute genus, since proposition 7 holds. This can be accomplished by calling the Use procedure:

Use (Mar&Mkt&Mfg, Mar&Mkt&Mfg, [D5, V_mkt], V_mfg; Mar&Mkt&Mfg)

The result is the Structured Model shown in figure 11. Now, again, the E_mfg function genus can replace the E_mar attribute genus since proposition 7 holds. This is accomplished by calling the Use procedure as below:

Use (Mar&Mkt&Mfg, Mar&Mkt&Mfg, [D4, E_mar], E_mfg; Mar&Mkt&Mfg)

The result is a Structured Model indicated as the goal of this step (see figure 12). As in step II, at the end of step III the Prod_mfg primitive entity genus has no meaning and it can be deleted.

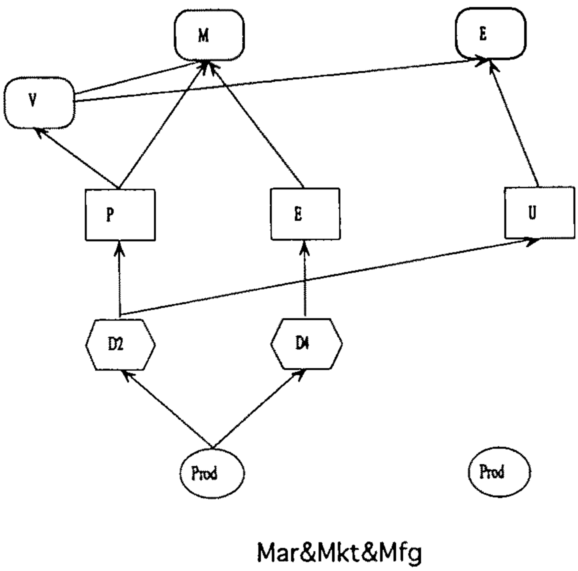


Figure 11. The situation after the replacement
of the V_mfg attribute genus.

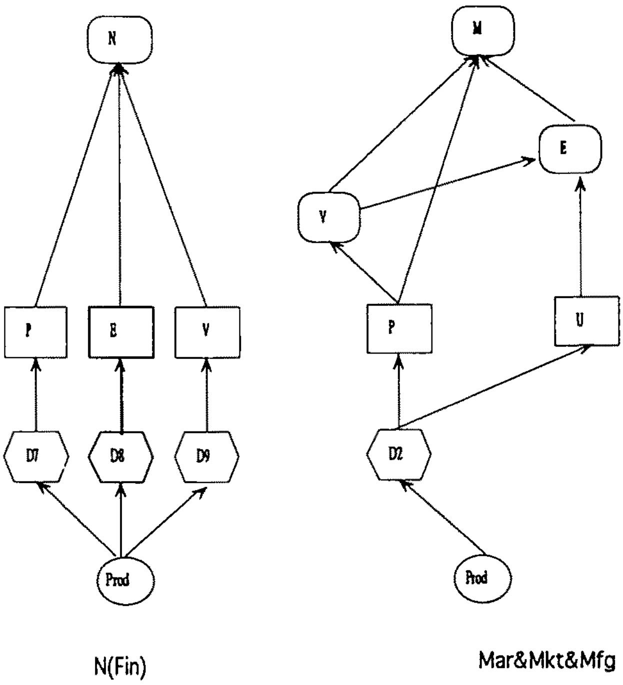


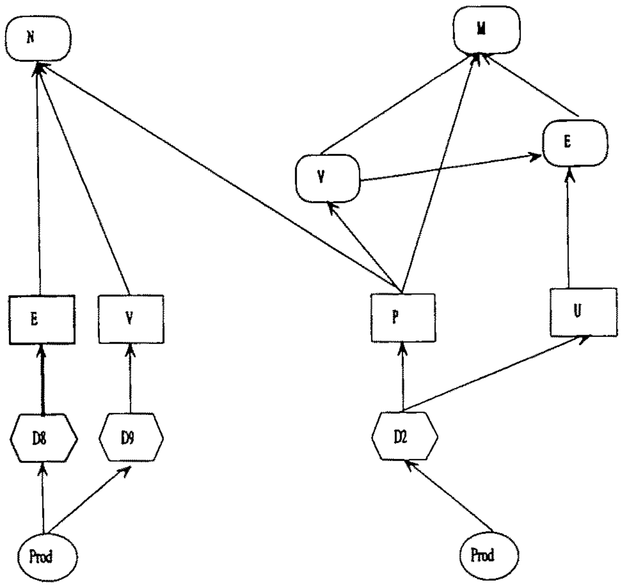
Figure 12. The situation after step III.

Step IV: *Starting with the result obtained at step III, choose two model and integrate them using first- and second-level procedures*

We consider the models *N*(*Fin*) and the Mar&Mkt&Mfg. Both show the attribute genera P_fin and P_mar, which are the prices of the products. To replace P_fin with P_mar, we call the Replace_Attribute procedure as below. The result is the integrated model shown in figure 13.

Replace_Attribute (N(Fin), Mar&Mkt&Mfg, [D7, P_fin], [D2, P_mar];
[N(Fin), Mar&Mkt&Mfg])

For short, we write Mar&Mkt&Mfg&Fin instead of [N(Fin), Mar&Mkt&Mfg].



Mar&Mkt&Mfg&Fin

Figure 13. The model after the replacement of the P_fin genus.

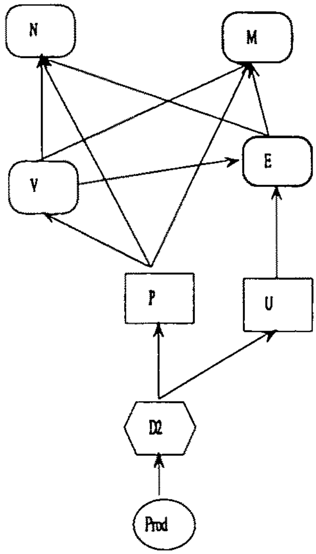
Now the function genus V_mkt can replace the attribute genus V_fin and the function genus E_mfg can replace the attribute genus E_fin. In both cases, proposition 7 holds. This can be accomplished by calling the Use procedure twice, as below:

Use (Mar&Mkt&Mfg&Fin, Mar&Mkt&Mfg&Fin, [D8, E_fin], E_mfg;
Mar&Mkt&Mfg&Fin);

Use (Mar&Mkt&Mfg&Fin, Mar&Mkt&Mfg&Fin, [D9, V_fin], V_mkt;
Mar&Mkt&Mfg&Fin);

As in steps II and III, the vestigal primitive entity Prod_fin can be deleted, since it has no meaning.

The resulting integrated model corresponds to the one of the example in [10]. Let us point out that, since the used procedures are all closed, the order of the sequence of steps II–IV is arbitrary. In fact, we could choose any two models to be integrated, and formulate the correct sequence of integration procedures according to the rules we have defined.



Mar&Mkt&Mfg&Fin

Figure 14. The final integrated model.

6. Future extensions

In the previous sections some procedures, classified to be on first or second level, were presented. They are the first steps toward the definition of a formal theory.

Of course, the elementary operations described are not exhaustive. For example, two other simple operations on nodes are: split and merge.

These operations are not easy to formalize, especially when applied to function and/or test genera. In this case, it is possible to construct different procedures which depend on the rules of the genera and the will of the model integrator.

The procedures described can cover many situations, but there are cases where they fail. Here is a simple example [10]:

To define a Two-Echelon Transshipment Model integrating two Classical Transportation Models* such that the output of the first becomes the input of the second.

*The formulation is given in section 4.

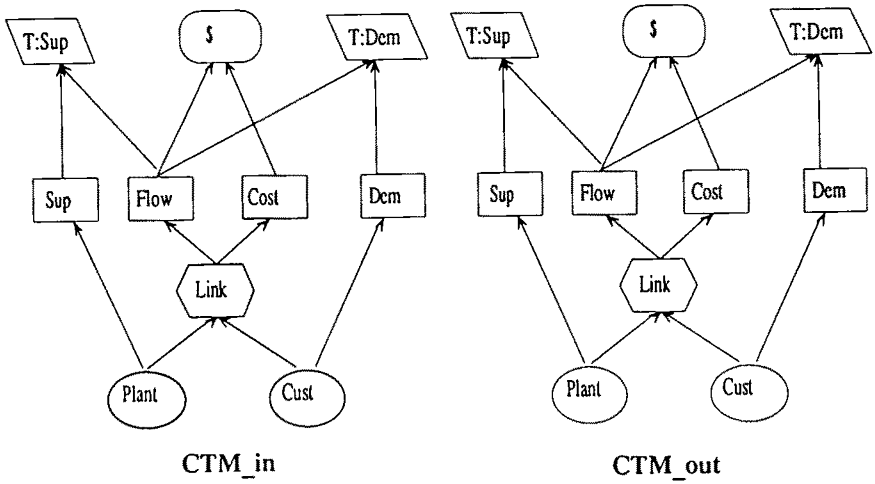


Figure 15. The two Classical Transportation Problems (CTM) to be integrated.

For notation, we suffix the genera of the Classical Transportation Problem used as input in the integrated model with _in, with _out the other.

The following are the steps necessary to integrate.

Step I: *Delete the genera not required by the integrated model*

The DEM_in and the T: DEM_in genera are deleted because the input section of the integrated model does not need to deal with the demand of the customers. For a similar reason, the Sup_out and T: Sup_out genera are deleted. To accomplish this task, the Delete_Node procedure is called four time, as shown below:

```
Delete_Node (Dem_in);
Delete_Node (T_Dem_in);
Delete_Node (Sup_out);
Delete_Node (T_Sup_out);
```

The resulting models, which in this particular case are also Structured Models, are shown in figure 16.

Step II: *Identify the genera*

The Cust_in and the Plant_out genera need to be merged because the final integrated model identifies the arrival nodes of CTM_in with the starting nodes of CTM_out. The merged genus is renamed. Its formulation is as follows:

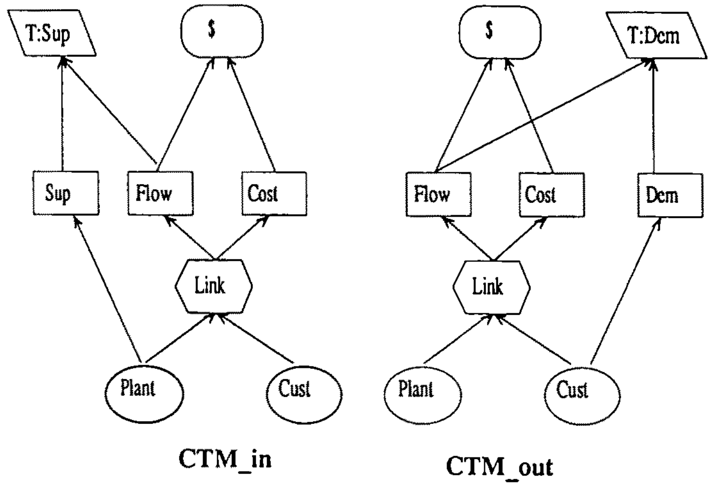


Figure 16. The CTMs after the Delete_Node procedures.

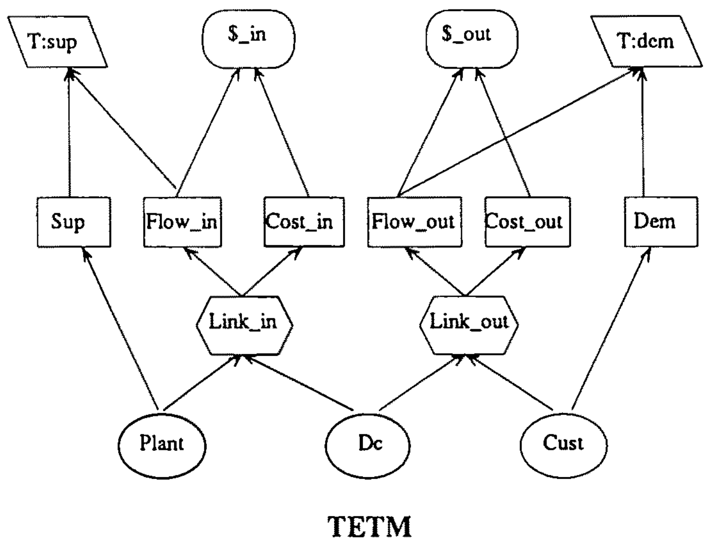


Figure 17. The integrated model after the merging.

Genus DC **primitive**
feature
label : **string**;
show label

The merge operation reroutes the definitional dependencies to this new genus. At the moment, this step cannot be executed using first and/or second level procedures. The result in shown in figure 17. For short, we write TETM (Two-Echelon Transshipment Model) instead of [CTM_in, CTM_out].

Step III: *Create a new test for inflows and outflows*

A completely new test genus has to be defined. It checks if the incoming flow equals the outgoing flow for each transshipment node. It is necessary to use the definition language to create the genus.

Genus T_DC **test**
call (FLOW_in : **iso** (DC.INDEX); FLOW_out : **iso** (DC.INDEX));
feature
DC_test : **boolean is result** := (**SUM** [PLANT.INDEX]
flow_in = **SUM** [CUST.INDEX] flow_out);
show DC_test

This action *cannot be* accomplished using first and/or second level procedures. The resulting Structured Model is shown in figure 18.

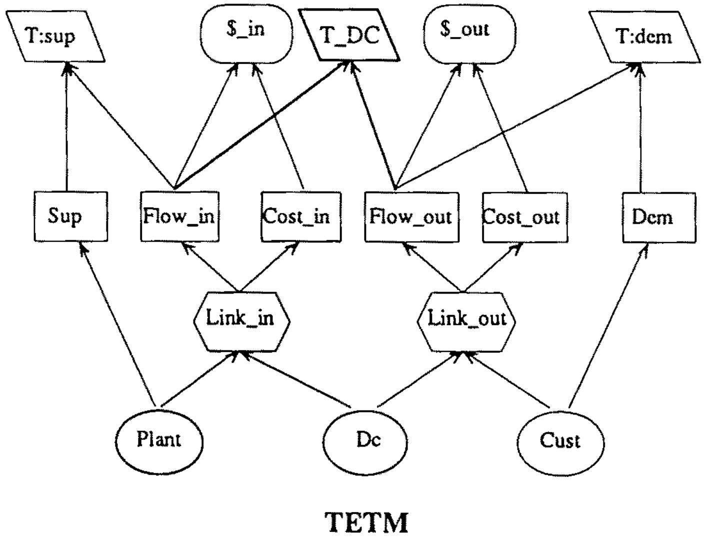


Figure 18. The situation after step III.

Step IV: *Create a new function genus which sums the $_in and the $_out values*

Again, a completely new function genus has to be defined, which sums the cost for the input section and the cost for the output section. It is necessary to use the definition language to create the genus.

Genus TOT **test**
call ($_in : **iso** $_out : **iso**);

feature
Sum_cost : **real is result** := totcost_in + totcost_out;
show Sum_cost

At this time, this action cannot be accomplished using first and/or second level procedures, but it looks more promising for the future when the merge procedure will be defined on the rules of function genera. The resulting Structured Model is shown in figure 19. This model corresponds to the resulting integrated model as in [10]. At

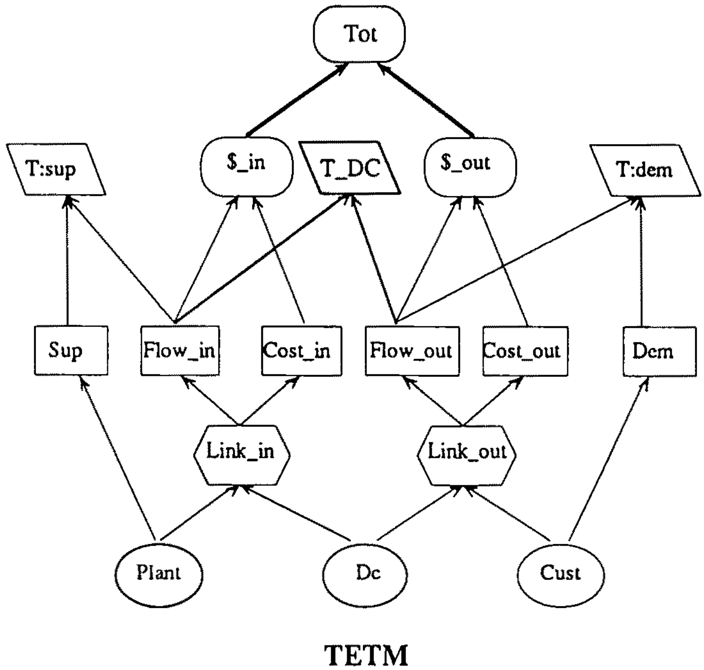


Figure 19. The final integrated model.

this time, steps II, III and IV procedures lie on the third level of integration. In fact, the user is required to define ad hoc genera and to set the definitional dependencies. It is desirable that the integration work is done using, as much as possible, automated procedures.

Our research line takes two directions:

- the first tries to develop new procedures, so that much of the work to integrate models can be done at levels 1 and 2;
- the second defines within a Model Management System a language and graphics tools to create procedures not available at levels 1 and 2. So doing, errors are minimized and productivity is increased.

Acknowledgements

The authors wish to thank Professor Arthur M. Geoffrion for helpful comments on a very early draft of this paper. All eventual errors which may be present must, however, be considered only the responsibility of the authors.

References

[1] A. Andronico, L. Cossa, M. Gagliardi and C. Spera, An object oriented approach to a model management system: Characteristics and examples, *Proc. 36th Annual ANIPLA Conf.*, ed. Bottaro and Zoppoli (Pirella, Genoa 1992).

[2] G.H. Bradley and R.D. Clemence, Jr., Model integration with a types executable modeling language, *Proc. 21st Annual Hawaii Int. Conf. on System Sciences*, Vol. 3 (IEEE Computer Society Press, Washington, 1988) pp. 403–410.

[3] D.R. Dolk and J.E. Kotteman, Model integration and a theory of models, Dec. Support Syst. 9(1993)51–63.

[4] M. Gagliardi and C. Spera, BLOOMS: Basic Language Object Oriented for Modeling Systems, Working Paper No. 11, Department of Quantitative Methods, University of Siena, Italy (1994).

[5] M. Gagliardi and C. Spera, The syntax of BLOOMS, Part I: Introduction, Working Paper No. 10, Department of Quantitative Methods, University of Siena (1994).

[6] M. Gagliardi and C. Spera, The syntax of BLOOMS, Part II: Grammar, Draft Technical Report, Department of Quantitative Methods, University of Siena (1994).

[7] A.M. Geoffrion, An introduction to structured modeling, Manag. Sci. 33(1987)547–589.

[8] A.M. Geoffriom, Integrated modeling systems, Comp. Sci. Econ. Manag. 2(1989)3–15.

[9] A.M. Geoffrion, The formal aspects of structured modeling, Oper. Res. 37(1989)30–51.

[10] A.M. Geoffrion, Reusing structured models via model integration, in: *Current Research in Decision Support Technology*, ed. R. Blanning and D. King (IEEE Computer Society Press, 1990).

[11] A.M. Geoffrion, The SML language for structured modeling, Oper. Res. 40(1992)38–75.

[12] J.E. Kottemann and D.R. Dolk, Model integration and modeling languages: A process prospective, Inf. Syst. Res. 3(1992)1–16.

[13] W. Muhanna and R. Pick, Composite models in SYMMS, *Proc. 21st Annual Hawaii Int. Conf. on System Sciences*, Vol. 3 (IEEE Computer Society Press, Washington, 1988) pp. 418–427.

[14] Y. Tasi, An operational approach to model integration using a structured modeling framework, *Proc. 1993 Pan Pacific Conf. on Information Systems*, Kaohsiung, Taiwan (1987). Full version: Research Paper, Anderson Graduate school of Management, UCLA.