1993

Model integration and a theory of models

Dolk, Daniel R.

Elsevier Science Publishers B.V.

Decision Support Systems 9 (1993) 51-63 North-Holland
http://hdl.handle.net/10945/48627

Decision Support Systems 9 (1993) 51–63
North-Holland

Model integration and a theory of models

Daniel R. Dolk

Naval Postgraduate School, Monterey, CA 93943, USA

Jeffrey E. Kottemann

The University of Michigan, Ann Arbor, MI 48109, USA

Model integration extends the scope of model management to include the dimension of manipulation as well. This invariably leads to comparisons with database theory. Model integration is viewed from four perspectives: Organizational, definitional, procedural, and implementational. Strategic modeling is discussed as the organizational motivation for model integration. Schema and process integration are examined as the logical and manipulation counterparts of model integration corresponding to data definition and manipulation, respectively. A model manipulation language based on structured modeling and communicating structured models is suggested which incorporates schema and process integration. The use of object-oriented concepts for designing and implementing integrated modeling environments is discussed. Model integration is projected as the springboard for building a theory of models equivalent in power to relational theory in the database community.

Keywords: Model integration, Schema integration, Process integration, Structured modeling, Communicating sequential processes, Integrated modeling environments.

1. Introduction

Model management research has been active for roughly a decade now and it's not stretching the truth to say that, during this period, model management has been model definition. Great effort and progress have been made in developing general model representations [1] in the form of executable modeling languages which lend themselves to computer manipulation. Structured modeling [15], logic modeling [26], and graph-grammars [22] are examples of model representation formalisms which have advanced the field significantly.

This concentration on representation has been a necessary and fruitful step in the evolution of model management. Model definition alone, however, is not sufficient to support a holistic view of model management. The equivalent of a viable calculus or algebra is also a necessary, but missing, part which allows us to think systematically about how to manipulate models once a suitable representation has been achieved.

Model integration provides a practical approach to thinking about 'modeling in the large' which forces our attention beyond the scope of definition to include operations upon models as well. Model integration is not a formal notion but rather a useful concept, which may be considered as another analogy of data management. In the same manner we are now accustomed to view

Daniel R. Dolk is Associate Professor of Information Systems in the Department of Administrative Sciences at the Naval Postgraduate School in Monterey, CA. Since receiving his Ph.D. in Management Information Systems from The University of Arizona in 1982, his research has focused primarily on model management and decision support systems. He has published extensively in this area in journals such as *Communications of the ACM, IEEE Transactions on Software Engineering, Interfaces*, and *Decision Support Systems*. He is currently Associate Editor for *ORSA Journal on Computing, Information Systems Research*, and *Journal of Database Administration*, and is a member of the ACM, IEEE Computer Society, and TIMS.

Correspondence to: Daniel R. Dolk, Naval Postgraduate School (AS/DK) Monterey, CA 93943-5000 USA.

Jeffrey E. Kottemann is Associate Professor of Computer Information Systems in the School of Business Administration at The University of Michigan. He received his Ph.D. in Management Information Systems from The University of Arizona and has published articles in *Communications of the ACM, Decision Sciences, Decision Support Systems, Information Systems, Journal of MIS, MIS Quarterly, Omega: The International Journal of Management Science, Organizational Behavior and Human Decision Processes*, and *Research Advances in Computers and Social Science*. His current research interests include devising principles and techniques for the development of large-scale information systems and modeling systems as well as assessing the effects of computer-based decision support technologies on decision makers' performance and on their performance beliefs.

[1] The terms 'model definition' and 'model representation' are used synonymously throughout this paper.

files in a DBMS as application-independent, integrable resources (via joins, for example), we can think of models as sharable resources which can be combined in ways unanticipated by their original developers. Although 'models as data' is old hat by now, examining model integration from a database analogy perspective nevertheless yields the following insights:

(1) Database theory (at least relational theory) is not sufficient to build an equivalent theory of models which includes model integration. Although data management philosophy has provided a convenient metaphor for model management, database theory has been as frequently confounding as enlightening. Part of the reason for this is that the focus of model management is on the schema rather than on the relation.

(2) All roads lead to object-oriented environments for implementation of integrated modeling environments. This is not surprising since models are complex data structures requiring complex manipulations. However, object-oriented is not a substitute for a theory which encompasses model manipulation as well as representation.

We will discuss model integration from four different dimensions: Organizational, definitional, procedural, and implementation. Initially, we take an organizational perspective to argue that effective strategic planning requires the integration of models developed for specific functional and operational applications. From a technical perspective, we view model integration in two dimensions: Definitional and procedural in accordance with the classical dichotomy of programming languages (functional languages excepted). Definitional integration corresponds to schema integration while procedural integration corresponds to process synchronization. Each of these is surveyed in detail. Taking an implementation perspective, we discuss the system requirements of an integrated modeling environment (IME) which supports these technical concepts Of model integration, and note the applicability of object-oriented concepts. We summarize by considering the theoretical perspective and suggest that research is needed to develop model integration from a loose federation of database-related and programming language-related concepts into a robust theory of models which unifies model definition and manipulation.

2. Organizational dimension: Strategic modeling

The utility of modeling has all too often been circumspect in organizations because of the technological barriers models present and the subsequent reluctance of management to use them. This is an ongoing battle which is continually being fought by the operations research community and to which model management can make genuine contributions by improving the accessibility and comprehensibility of models.

Models are also underutilized because the organizational perspective from which they are developed is often too limited, that is, at the func-

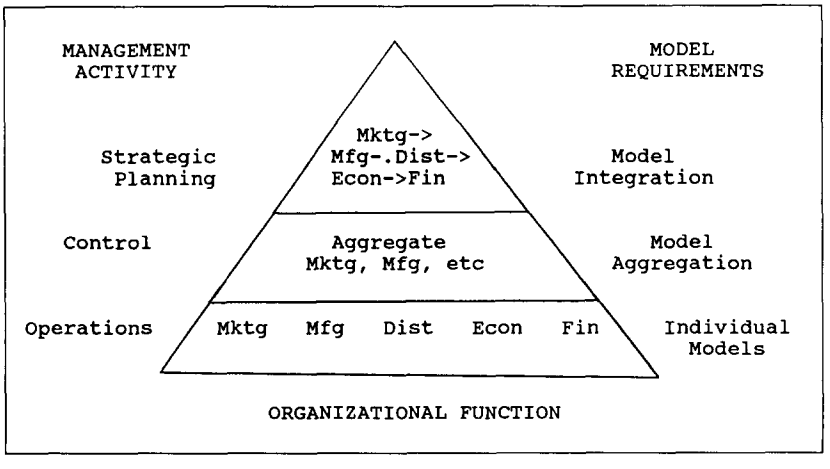

Fig. 1. Organizational view of modeling by management activity and function.

tional, operational level rather than at the control and strategic levels. Models developed at the operational level tend to stand in isolation and frequently don't provide the necessary information for organization-wide planning. For example, a production model developed without consideration of the organization's accounting model may maximize total output when it makes more sense to minimize unit cost. As one moves up the organizational pyramid, modeling requirements must evolve towards strategic objectives which, in turn, will place greater emphasis upon model aggregation and integration (fig. 1).

Consider a firm which has developed the following independent models (this example has been adapted from [3]):

(1) an econometric marketing model which forecasts demand in terms of sales volume for a product for the next fiscal year;
(2) a discrete event simulation manufacturing model which estimates the required expense to produce enough of the product to meet a specified demand;
(3) a transportation model which determines the minimal cost of distributing the product to customers;
(4) a pricing model which calculates a price for a product given demand volume, and production and distribution expenses; and
(5) a financial model which determines the revenues and net income from sales of the product given demand volume, manufacturing and distribution expenses, and product price.

The relatively narrow view of a model developed at the operational level is often unable to provide, or contribute to, the broader sensitivity analysis demanded by upper management. Suppose management asks the question "what effect will replacing two machines in the production process have on net income?", or alternatively, "what will happen to revenues if demand for our product softens as a result of decreased spending by the Department of Defense?" No single model can provide the desired information. A response to these 'what if' queries requires linking and running several or all of the models. Figure 2

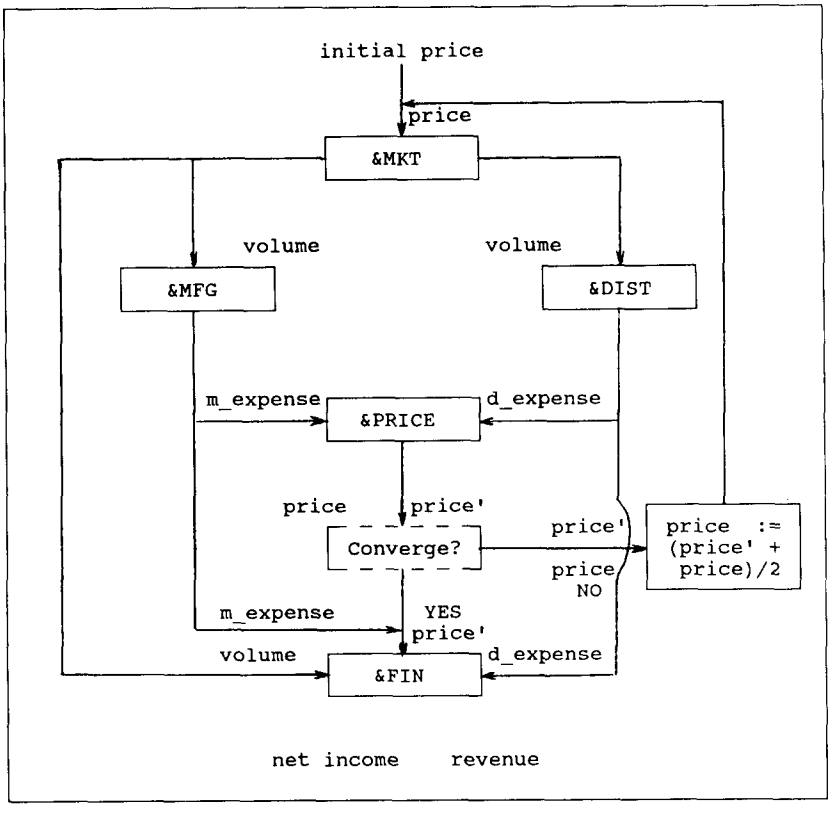

Fig. 2. Logical connection between independent models (adapted from [3]).

shows how these models might be interconnected conceptually by having the outputs of one serve as the inputs to another. Price computation is shown as an iterative process which will require multiple passes through the first four models to achieve convergence.

Another ramification of building models from strictly a functional or operational perspective is that each model is likely to be developed as a standalone tool in a separate software environment using different languages. For example, the marketing model may be developed in SAS, the production model in Simscript, the transportation and pricing models in GAMS, and the financial model in a spreadsheet. These separate software environments with their unique languages further isolate models from one another and restrict their integrability and potential utility. In order to satisfy the sensitivity analyses above, four different modeling environments and languages must somehow be linked. This represents a formidable programming challenge which may very well be prohibitively complex and expensive to implement. The following sections discuss the foundations of an integrated modeling environment (IME) which can overcome this technological barrier to model integration.

3. Definitional dimension: Schema integration

There are two dimensions which must be considered in the process of model integration: definitional (model representation) and procedural (model manipulation). Definitional integration involves the logical linking of similar model representations whereas procedural integration concerns the linking of processes to form operators

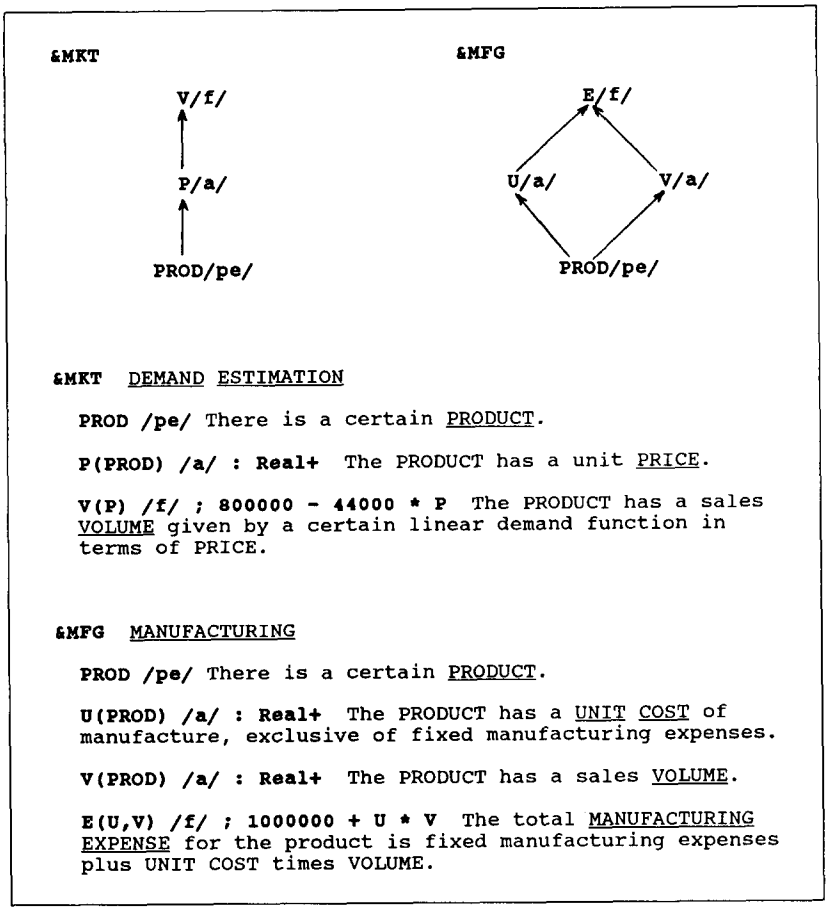

Fig. 3. Structured model genus graphs and associated schemas for the marketing and manufacturing models [16].

which subsequently manipulate these integrated representations. These two dimensions will be considered in turn in the next two sections.

A necessary prerequisite for model integration is that models be cast in some lingua franca so that they may eventually be joined. A large part of model management research has been devoted to the development of formal model representation schemes that facilitate this. Structured modeling [15], logic modeling [26], and graph grammars [22] are three such formalisms. Structured modeling shares a common ancestry with the data modeling approaches underlying database management, particularly the entity-relationship model [8). Structured modeling goes well beyond entity-relationship, however, and has particular relevance to applications in operations research and management science. Logic modeling originates largely from artificial intelligence concepts and relies (usually) on first order logic-not only for representation of models, but for manipulation as well. As a result, the dichotomy between definition and procedure is less of a problem with this approach. Graph grammars provide a graph-based paradigm for model representation which is especially effective for node-arc problems such as network analysis. This approach is also in concert with the trend toward graphical user interfaces which now earmark contemporary operating system environments.

These three approaches to model definition are as much complementary as they are competing formalisms (see [7]). We will initially use structured modeling to demonstrate definitional integration but will discuss contributions to model integration from logic modeling and graph grammars as well. We assume the reader has some familiarity with structured modeling; consult [15,17] for more details.

To illustrate integration at the definitional level, we provide simple examples of the marketing and manufacturing models discussed in Sec-

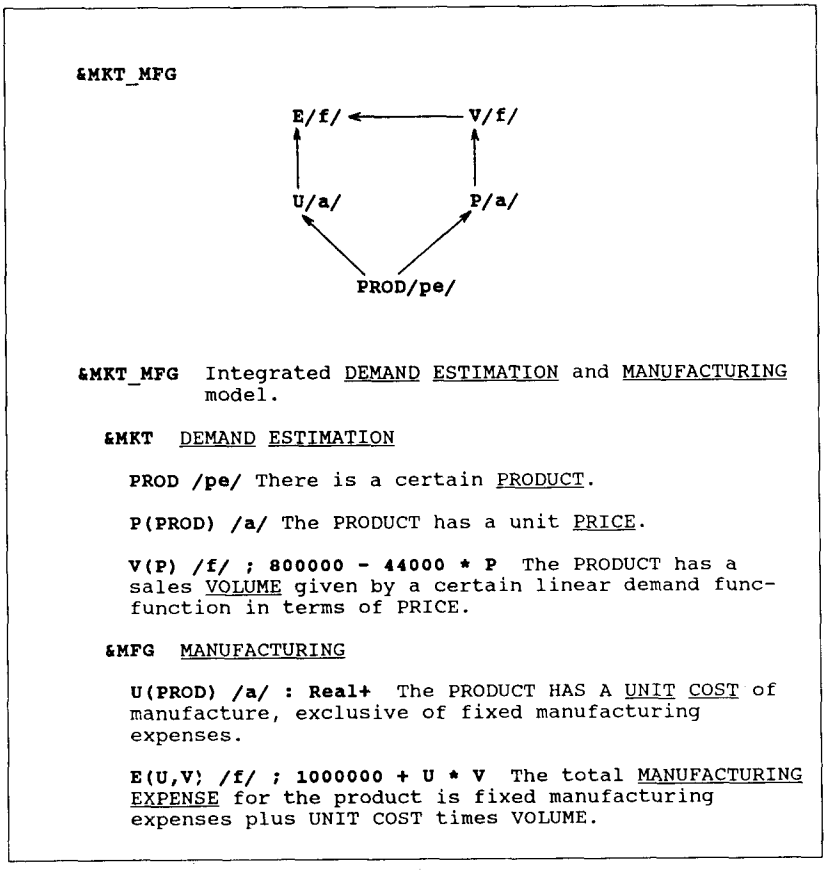

Fig. 4. Structured model genus graph and schema for integrated marketing and manufacturing model.

tion 2, represent them as structured models, and then integrate them as described in [16]. Figure 3 shows structured model genus graphs and schemas for the two models.

The genus graphs probably provide the most intuitive medium for understanding how to integrate these two models, so we will describe the major steps involved in the integration process from this perspective:

(1) identify places where the graphs can be 'joined';
(2) 'join', the graphs;
(3) modify the resultant graph to maintain structural consistency;
(4) regenerate the associated relational schemata for ((4) storing the model's elemental detail (data).

The first step is perhaps the most important one in model integration. Identifying commonalities between models where they can eventually be 'joined' is the problem of *variable correspondence* as we describe in the section about procedural integration. This involves discerning which components of models are really the same or which components can be made the same by simple transformations. In our-simple example, PROD (product) and V (sales volume) appear in both models although V is a calculated quantity in the &MKT model and a simple attribute in the &MFG model.

Variable correspondence can be considerably complex. Assume, for example, that the volume in the &MKT model is expressed in units whereas the volume in the &MFG model is in 100's of units. The integration process must be able to recognize this and make the necessary conversions to the underlying elemental detail to ensure that the integrated model is dimensionally consistent. Structured modeling has no features to support this critical aspect of integration. Typing schemes to support model integration are discussed below.

Assuming that PROD and V are resolved to be the same entities, this suggests a 'join' which substitutes the &MKT genus graph for the PROD $\rightarrow V$ link in the &MFG genus graph (fig. 4). Once two models have been 'joined' graphically, its then necessary to check whether the integrated model satisfies the structural properties of a structured model. This requires that we look at the new model schema.

The result of the graphical 'join' is reflected in the model schema by first concatenating the two schemas, dropping the superfluous PROD paragraph in &MFG and then deciding which of the V paragraphs to retain. In this scenario, V is an output of the marketing model which serves as an input to the &MFG model (see fig. 2), so V as calculated in &MKT is what we need to retain. Therefore we drop the in paragraph in the &MFG module from the integrated schema which now constitutes a valid structured model schema (fig. 4).

Structured modeling was developed partly to provide more information about a model than previous 'black box' representations that show only model inputs and outputs. Structured modeling does not deal with inputs or outputs explicitly since these can be viewed as application-dependent designations. Nevertheless it is often useful for variable correspondence determination to know which variables are inputs and/or outputs. Output variables in structured modeling will usually be designated as either variable attribute (va) or function (f) elements. Model inputs usually will be fixed attribute (a) elements although it's conceivable that primitive entity (pe) elements could also be inputs.

The last step in the integration process is determining the resultant relational schemata from the new schema. Structured model genus graphs form functional dependency graphs which can be translated into relational schemata in third normal form. The relational schemata for the original marketing and manufacturing models and the integrated model are as follows

(1) &MKt: PROD(**prod _id**, p, v);
(2) &MFG: PROD(**prod _id**, u, v, e);
(3) &MKT_MFG: PROD(**prod _id**, p, u, v, e).

Notice that, in this case, the schemata corresponding to the integrated model could be formed as a view joining **prod _id** of the original relations (assuming they were named differently, of course).

As we suggested above, model integration introduces the need for typing schemes and inheritance schemes to facilitate variable correspondence. In general, one may need rather extensive knowledge about a variable's type in order to resolve two variables and subsequently integrate their associated models. This has led to some interesting research in typing schemes to support

model integration. Bradley and Clemence [4,5] have developed a concept hierarchy typing calculus which assigns units, dimensions, and concepts to model variables. If two variables are similar in these three attributes, then they can be used to 'join' models, perhaps through some intermediate transformations.

In the domain of logic modeling a similar effort is underway. Quiddity is an approach to typing which is broader in scope than the concept hierarchy but with the same objectives for model integration [1]. By defining the quiddity of variables, their similarity and mergeability can be determined and implemented, if possible.

It is tempting to view model integration as a direct corollary of the relational join. This is a naive approach, however. The appropriate database analogy for definitional integration is not the relational join but rather schema integration. In other words, definitional integration involves 'joining' at the conceptual model level rather than at the relational level. This in turn requires development of typing and inheritance schemes, much like the work currently being done in object-oriented databases [32].

4. Procedural dimension: Process integration

Definitional integration is only one side of the integration coin. Even if we can find robust ways to integrate the logical description of models, the question still remains of how we manipulate this newly created object called an integrated model. As the previous section indicated, we are not dealing with so tidy a world as relational theory

```
PROCESS INTEGRATED_MODEL

    # Declare models.
MODEL &MKT, &MFG, &DIST, &PRICE, &FIN

    # Prompt user for initial price.
PROMPT('Price?', PRICE)
REPEAT UNTIL DONE
      # Marketing model is SAS econometric model.
  &MKT.PRICE = PRICE
  SOLVE &MKT USING SAS
      # Variable correspondence..units in manufacturing
      #  model are in 100's.
  &MFG.VOLUME  = &MKT.VOLUME * 100
  &DIST.VOLUME = &MKT.VOLUME
      # Manufacturing model is a Simscript DEVS model.
      # Transportation model is GAMS optimization model.
      # &MFG and &DIST can be run in parallel.
  SOLVE CONCURRENTLY &MFG USING SIMSCRIPT AND
                     &DIST USING GAMS
      # Variable correspondence between &MFG and &PRICE
      #  and &DIST and &PRICE.
  &PRICE.VOLUME  = &MFG.VOLUME
  &PRICE.EXPENSE = &MFG.EXPENSE + &DIST.EXPENSE
      # Pricing model is GAMS optimization model.
  SOLVE &PRICE USING GAMS
      # Test for price convergence.
  IF FAIL IN TESTCONVERG.SV
     PRICE = (PRICE + &PRICE.PRICE) / 2
  ELSE
        # Variable correspondence between &PRICE,
        #  &DIST, &MFG and &FIN models.
     &FIN.PRICE   = &PRICE.PRICE
     &FIN.VOLUME  = &MFG.VOLUME
     &FIN.EXPENSE = &MFG.EXPENSE + &DIST.EXPENSE
        # Finance model is spreadsheet model.
     SOLVE &FIN USING LOTUS123
     DONE
  ENDIF
END REPEAT
```

Fig. 5. Process model for integrating solvers.

with its properties of completeness and transitive closure. What then are the counterparts of relational algebra and calculus which apply to model manipulation? This section attempts to lay a groundwork for this issue.

One of the tenets of model definition is that representation and manipulation are separable functions. For example, a model representation should be as independent as possible from any solver(s) which eventually may act upon it. This is vital to the ultimate comprehensibility of models. In the optimization world, model representation has traditionally been tightly bound to the data structures required by solution algorithm software. This has resulted in models remaining relatively inaccessible to the decision makers for whom they were intended to benefit.

The separation of the modeling world into representations and solvers has significant ramifications for model integration. If we connect two or more models at the logical level, we need to determine the corresponding action at the solver level. For example, consider a situation where we have separate transportation models for eastern and western regions and we want to integrate them into a national transportation model. In this case, which is one primarily of model aggregation or homogeneous integration, the solver will be exactly the same for the national model as for each of the regional models. Only the logical schemas have to be integrated.

Our example from Section 2, on the other hand, requires a more complex approach. In this case, the notion of using the same solver for the integrated model can be rejected out of hand because the models are fundamentally of such different types. What makes more sense in multiparadigmatic, or heterogeneous integration, is to concatenate each of the model's solvers in roughly the same way as their schemas in order to derive an integrated solver. This solver is, in fact, nothing more than a process which controls the individual processes corresponding to each model's solver. Figure 5 indicates how this might be accomplished in a hypothetical language which we can think of as a model manipulation language (MML). Our assumption here is that we have structured model schemas for each of the five models (&MKT, &MFG, &TRANSP, &PRICE, and &FIN respectively), and a corresponding library of solvers which can be invoked via a SOLVE command. For the time being, we will also assume that the &MKT and &MFG models are more complex than shown in Section 3, although the variable correspondence will remain the same.

There are several aspects of this MML process that bear mentioning:

(1) models are the basic objects being manipulated; [2]
(2) variable correspondence is handled explicitly, for example the conversion of VOLUME in &MKT from units to 100's of units in &MFG. This process can be done automatically with an appropriate typing scheme [5];
(3) SOLVE executes a process which solves a model, for example, the statement "SOLVE &MKT USING SAS" when executed, would invoke the SAS program;
(4) SOLVE implies an underlying transformation which converts the elemental detail tables (the model's data) to the appropriate data structures for the specified solver. For example, solving &PRICE requires that elemental detail tables be converted to GAMS format before the GAMS program is executed;
(5) the order of processes is important. &MKT must be executed before &MFG since &MFG requires as input the VOLUME output from &MKT;
(6) processes may be run in parallel. The "SOLVE CONCURRENTLY &PRICE ... &TRANSP" command is meant to indicate that these two models could be solved in parallel, perhaps in two separate windows.

The MML in fig. 5 is a simplified version of a model integration control language (MICL) proposed by Kottemann and Dolk [24]. Besides variable correspondence and sequentiality, the MICL supports model (process) synchronization as well. Synchronization occurs when two concurrent processes must exchange variables during their respective executions. For example if the pricing model were geographically sensitive and the transportation model were price sensitivel it may be necessary for the &PRICE and &TRANSP

[2] There are a number of manipulations which can be performed on models besides solve, e.g., evaluate, retrieve, modify, create, etc. For the purposes of this discussion, however, we will restrict the scope to the solve operation only. Extension of the concepts to other operations is straightforward.

solvers to exchange information at specified intervals during their respective executions (after every recalculation of a basis, say). This is a more complex form of model integration and solver interdependence, but one which occurs frequently in dynamic models such as discrete event simulation. Synchronization also requires demon constructs which act as dynamic interrupts during process execution to alert other processes that a certain status has been achieved (e.g., a new basis has been recalculated). An example of an MICL for econometric modeling is discussed in [12].

The basic argument here is that solvers are processes and therefore solver integration requires process integration. This places us squarely in the bailiwick of operating systems theory where several formalisms based on message passing have been developed for process coordination. Communicating sequential processes (CSP) [21] is one such formalism which has been adapted to solver integration [24]. Generative communication [6] is a 'lazy' form of message passing where model outputs are not targeted to any specific process, as in CSP, but instead are stored in common areas which can be accessed by all processes when needed. Concurrent Prolog is a logic-oriented counterpart that allows dynamic communication between Prolog processes, and which could implement model integration within logic modeling environments [31].

The process view of model integration is a significantly different perspective from the schema integration approach. Whereas the latter provides a logical view of model structure, the former corresponds to more traditional 'black box' model representations which emphasize inputs and outputs as opposed to the interrelationships of variables. The alert reader may be wondering at this point what the connection between schema and process integration is, and whether or when we need each. For example, since the integrated solver of fig. 5 requires solving the individual models, why do we need an' integrated schema? Isn't it the individual models which are of interest in this case?

A plausible response to this question was hinted at before. Integrated schemas are probably more appropriate for the homogeneous case where the same model solver will be used for the integrated model as for the constituent models. Process integration, on the other hand, may be more suitable for multiparadigmatic modeling where the models and their associated solvers are heterogeneous. Another distinction between schema and solver integration is that we are dealing at a higher level of abstraction with the latter. The model is the basic object of inquiry for solvers whereas the model variable is the basic unit in schemas. Thus, it appears, that as a counterpart to variable typing for schema integration, we may need the notion of a *model type* for solver integration. A model type would determine which solvers could be applied to an instance of a model schema. When two models of the same, or inherited type, are to be integrated, then schema integration is appropriate; otherwise solver integration is desirable.

One of the themes running through this paper is the limitation of the "modelbase as extended database" analogy. We have seen, for example, at the schema integration level that joining models is more complex than joining relations. Transitive closure and relational completeness do not apply because the appropriate kernel of scrutiny is the conceptual model rather than the relation. Similarly, we see that process integration is also not conducive to manipulation by relational calculus or algebra, and requires more dynamic formalisms such as those which underlie operating systems and programming language design. Although we would like a theory comparable in richness to relational algebra and calculus for manipulating models, the database arena might not be the place to look. Indeed, in the next section, we take the natural step of suggesting the object-oriented paradigm for implementing an IME which supports both schema and solver integration, and we note that attempts to devise algebras for object manipulation have been unsuccessful as well.

5. Implementation dimension: Object-oriented integrated modeling environments [3]

The concept of an IME which can handle models as flexibly as DBMS's handle data is an

[3] IME is being used here in the same sense that 'model management system' has been used in other contexts. We prefer IME because it encompasses a broader notion of integration; not just integration of models and data but integration of software tools (DBMS, GUI, solvers, etc.) as well [18].

appealing prospect. In this respect, database analogies seem apropos, with model definition as the counterpart of data definition and model integration the equivalent of data manipulation. As discussed before, however, direct database analogies can be misleading. We review briefly the requirements which model integration imply for an IME and suggest possible blueprints to building such a system.

One way of looking at IME requirements is to determine the extent to which model integration can be automated. For example, if we want to integrate two or more model schemas, what support can an IME provide? A necessary condition is a common model definition formalism and associated language such as structured modeling and SML [19] in which schemas over a wide class of models can be created and linked. An associated requirement is that transformation facilities be available to convert other model representations to this common definition formalism. This would provide interfaces to other external modeling systems such as AMPL [14], for example, and would allow modelers familiar with these systems to work comfortably within the IME.

Even with a common internal model representation, it is unreasonable to expect a system to do the entire process of model schema integration without some human intervention. Some steps can be reasonably achieved automatically, however. For example, identification of synonyms (variables with the same semantics) and homonyms (variables with the same names but different semantics) is possible if there is a sufficiently powerful variable typing scheme such as concept hierarchy or quiddity in effect. The act of 'joining' models can be done as well, both at the graphical and schema levels. In the case of structured modeling, an IME could also inform the modeler of schema errors and inconsistencies existing in the 'joined' schema. Once a semantically and syntactically correct integrated schema was developed, the IME could then generate the new relational schemata and offer the modeler the choice of creating these new relations explicitly or building views from existing relations.

From a process integration perspective, an IME supporting 'automatic integration would facilitate the conversion of a process diagram such as fig. 2 into the integrated solver of fig. 5. The degree to which this can be done fully automatically depends on the complexity of the model integration, particularly with respect to the difficulty of the variable correspondence involved and the degree of process synchronization required. Muhanna and Pick [29] have implemented such a system under a simplified set of assumptions which minimizes these difficulties. Their SYMM system supports a graphical interface for representing the model integration (similar to fig. 2), but there is no associated model manipulation or control language. In the general case, as we've

Table 1
IME requirements for supporting model integration and selected references

IME requirement	Relevant research
Uniform *internal* model definition scheme capable of representing many classes of models.	Geoffrion [15,17], Jones [22], Lee and Krishnan [26]
Conversion of *external* model definition schemes into internal scheme	Maturana [28], Bhargavs and Kimbrough [2], Chari and Krishnan [7]
Robust typing and inheritance at both the variable and model level.	Bradley and Clemence [4], Bhargava et al. [1], Liang [25]
Model manipulation language based on message passing to support solver integration.	Muhanna and Pick [29], Kottemann and Dolk [24]
Model solution libraries with transformation routines for conversion of internal data structures to solver data structures.	Eck et al. [13], Maturana [28], Ramirez et al. [30]
Graphical user interfaces and views for supporting model definition and integration.	Jones [23), Muhanna and Pick [29], Ma et al. [27], Greenberg and Murphy [20]
DBMS tools for model management.	Dolk [11], Desai [10]

shown in Section 4, some form of model manipulation language will be necessary, and useful, for specifying this form of model integration.

At the tool level, IMEs clearly require graphical user interfaces for specifying and integrating models, whether at the schema or the process level. Jones [23], for example, has developed a graph-based modeling system wherein model representation and integration are done entirely at a graphical level. Again, however, the user is responsible for ensuring proper variable correspondence across models. The DBMS is also a vital tool needed by an IME for handling the complex data manipulation that earmarks large scale modeling and model integration. Finally, we should note that improvements in operating systems over the years may very well provide some of the model integration features we've been discussing. For example the Mach version of Unix, which is the host operating system for the NeXT computer, provides advanced process communication capabilities which could be adapted as the basis for a message passing MML.

Table 1 summarizes some of the major requirements for an IME which result from model integration. It's interesting to note that most of the research has been directed towards model definition with only tentative forays into the model manipulation area. One anticipated benefit from thinking in a model integration context is that the scope of model management research will broaden to include this dimension.

Another reason that model manipulation has been largely ignored can be traced to the 'object-oriented' phenomenon. Numerous authors have recognized the applicability of this design methodology to model management and the associated benefits of models as objects, solvers bound to these models, and inheritance hierarchies. This has probably done as much harm as good in the advancement of model management research. Object-oriented approaches are too often used as an implementation panacea for sweeping difficult conceptual and theoretical problems under the rug. Model integration is one such problem.

Having said this, it will perhaps appear contradictory to now claim that object-oriented environments are promising implementation vehicles for IMEs. Nevertheless, the object-oriented paradigm has been shown as feasible and beneficial for building IMEs. For example, Dempster and Ireland [9] have implemented a debt management system using the frame-based KEE™ environment and Desai [10] has proposed implementing structured modeling representations in an object-oriented DBMS. Not surprisingly, the need for typing/inheritance schemes, message passing, and process coordination discussed in the previous sections leads us directly into the object-oriented camp. We briefly describe our own proposal for an object-oriented IME which we call Communicating Structured Models (CSM).

CSM is based upon structured modeling as the internal definition-medium, concept hierarchies as the variable and model typing scheme, and CSP as the process integration formalism. The basic idea is to develop an MML similar in structure to the hypothetical example in fig. 5, which will exist as a shell around SML. CSML (Communicating SML) will allow users to integrate models either schematically in the homogeneous case through graphical interfaces (similar to CASE tools) or procedurally in the multi-paradigmatic case through an appropriate language syntax. CSML must support at least the following features:

(a) the basic structured programming constructs of sequence, selection, and iteration;
(b) demons;
(c) embedded SML statements for model definition;
(d) parallel execution of processes;
(e) transformation operators to solver data structures;
(f) embedded SQL statements for data manipulation.

In short, CSML would require many of the characteristics of a discrete event simulation programming language such as Simscript™ but with hooks to model schemas, solvers, and relational data as well.

Implementation of CSML would itself constitute a significant software integration effort, undoubtedly requiring some existing object-oriented environment as a foundation. This only reinforces the link between IMEs and object-oriented concepts. However, we reemphasize our view that object-oriented is primarily an implementation choice for building modeling environments rather than a substitute for model theory. Although we have taken pains to describe the challenges of model integration independent of any particular

implementation methodology, it is very difficult to separate the issues of model integration from those of object-oriented modeling. One of the goals of model management research should be to take a fresh look at how to develop a more theoretical foundation for model integration which avoids this confusion.

6. Conclusions

The purpose of this paper has been to explore model integration as a vehicle for thinking about "modeling in the large". This has served several useful purposes by:

(1) surveying the main aspects of model integration, specifically schema and process integration,
(2) exposing limitations of relational database theory as a paradigm for model management theory;
(3) extending the scope of current model management research beyond model definition to include model manipulation.

The dimensions of model integration as described herein seem to lead us in one form or another to object-oriented concepts for implementation. This is quite natural given the increasing symbiosis between models and computers in both the scientific and business worlds. However, this is somewhat less than satisfying theoretically. If we are to develop a theory of models which encompasses definition and manipulation, it will require answers to at least the following list of research questions:

(1) Can a calculus or algebra be devised for particular classes of models which exhibit the property of transitive closure as in the relational model?
(2) What constitutes a complete set of primitives for a model manipulation language?
(3) Is there a set of necessary and sufficient abstract data types and inheritance hierarchies for modeling?

Much of the development of computer languages has been essentially constructivist in nature, that is, a set of requirements is generated and a language is then built which attempts to satisfy these requirements. This was also the modus operandi for database systems and languages until development of the relational theory and model. The existence of a strong theoretical foundation nurtured and strengthened the discipline of database management significantly. Model management research is now at about the equivalent stage of evolution with respect to languages. The question remains, will model manipulation languages be built in a constructivist mode or will they evolve from an appropriate theoretical foundation? This is a major challenge for model management and one for which the concept of model integration provides invaluable perspective.

References

[1] H. Bhargava, S. Kimbrough, and R. Krishnan, Unique names violations: A problem for model integration, ORSA Journal on Computing 3, 2 (1991) 107–120.
[2] H. Bhargava, and S. Kimbrough, Model management: An embedded languages approach, Forthcoming in Decision Support Systems.
[3] R.W. Blanning, An entity-relationship approach to model management, Decision Support Systems 2, 1 (March 1986) 65–72.
[4] G.H. Bradley, and R.D. Clemence, Jr., A type calculus for executable modeling languages, IMA Journal of Mathematics in Management 1, 4 (1987) 277–291.
[5] G.H. Bradley, and R.D. Clemence, Jr., Model integration with a typed executable modeling language, Proceedings of the Twenty-First Hawaii International Conference on System Sciences III, IEEE Computer Society press (1988) 403–410.
[6] N. Carreiro, and D. Gelernter, Linda in context, Communications of the ACM 32, 4 (April 1989) 444–459.
[7] S. Chari, and R. Krishnan, Towards a logical reconstruction of structured modeling, Forthcoming in Decision Support Systems.
[8] P.P.S. Chen, The entity-relationship model: Toward a unified view of data, ACM Transactions on Database Systems 1, 1 (1976) 9–36.
[9] M.A.H. Dempster, and A.M. Ireland, Object-oriented model integrate in a financial decision support system, Forthcoming in Decision Support Systems.
[10] S. Desai, Are extensible database systems better than relational database systems for model management? Anderson Graduate School of Management, UCLA, Los Angeles, CA (1991).
[11] D.R. Dolk, Model management and structured modeling: The role of an information resource dictionary system, Communications of the ACM 31, 6 (June 1988) 704–718.
[12] D.R. Dolk, and D.J. Kridel, An active modeling system for econometric analysis, Forthcoming in Decision Support Systems.
[13] R. Eck, A. Philippakis, and R. Ramirez, Solver representation for model management systems, Proceedings of the Twenty-Third Annual Hawaii International Confer-

ence on System Sciences III, IEEE Computer Society (1990) 474–483.

[14] R. Fourer, P.M. Gay, and B.W. Kernighan, A modeling language for mathematical programming, Management Science 36, 5 (May 1990) 519–554.

[15] A.M. Geoffrion, An introduction to structured modeling, Management Science 33, 5 (May 1987) 547–588.

[16] A.M. Geoffrion, Reusing structured models via model integration. Proceedings of the Twenty-Second Annual Hawaii International, Conference on System sciences, IEEE Computer Society (1989) 601–611.

[17] A.M. Geoffrion, The formal aspects of structured modeling, Operations Research 37, 1 (January–February 1989) 30–51.

[18] A.M. Geoffrion, Integrated modeling systems, Computer Science in Economics and Management 2 (1989) 3–15.

[19] A.M. Geoffrion, SML: A model definition language for structured modeling, Western Management Science Institute, UCLA, Los Angeles, CA (November 1989).

[20] H.J. Greenberg, and F.H. Murphy, Views of mathematical programming models and their instances, University of Colorado at Denver, Denver, CO (May 1991).

[21] C.A.R. Hoare, Communicating sequential Processes (Prentice-Hall, Englewood Cliffs, NJ, 1985).

[22] C.V. Jones, An introduction to graph-based modeling systems, Part I: Overview, ORSA Journal of Computing 2, 2 (1990) 136–151.

[23] C.V. Jones, An integrated modeling environment based on attributed graphs and graph-grammars, Forthcoming in Decision Support Systems.

[24] J.E. Kottemann, and D.R. Dolk, Process-oriented model integration, Proceedings of the Twenty-First Hawaii International Conference on System Sciences III, (IEEE Computer Society Press, 1988).

[25] T–P. Liang, Analogical reasoning and case-based learning in model management systems, Forthcoming in Decision Support Systems.

[26] R.M. Lee, and R. Krishnan, Logic as an integrated modeling framework, Computer Science in Economics and Management 2 (1989).

[27] P. Ma, F.H. Murphy, and E.A. Stohr, Design of a graphics interface for linear programming, Communications of the ACM 32, 8 (1989) 996–1012.

[28] S. Maturana, Integration of a mathematical programming solver into a modeling environment, Anderson Graduate School of Management, UCLA, Los Angeles, CA (October 1988).

[29] W.A. Muhanna, and R.A. Picks, Composite models in SYMM. Proceedings of the Twenty-First Hawaii International Conference on system sciences III (IEEE Computer Society Press, 1988) 418–427.

[30] R.G. Ramirez, C. Ching, and R.D. St. Louis, Independenc and mappings in model-based decision support systems, Forthcoming in Decision Support Systems.

[31] E. Shapiro, ed, Concurrent Prolog Collected Papers, Volumes 1 and 2 (The MIT Press, 1987).

[32] M. Stonebraker, and G. Kemnitz, The POSTGRES next generation database management system, Communications of the ACM 34, 10 (October 1991) 78–92.