European Journal of Operational Research 41 (1989) 33–43
North-Holland

Theory and Methodology

Computer-based modeling environments

Arthur M. GEOFFRION
John E. Anderson Graduate School of Management, University of California, Los Angeles, CA 90024-1481, USA

Abstract: This paper gives the author's views on the kind of computer-based modeling environment needed to properly support management science/operations research work, and on the design challenges that need to be met in order to bring such modeling environments into being. It is a written version of the main ideas of two addresses: a plenary at IFORS 87 in Buenos Aires (August, 1987), and the keynote at the 1988 Canadian Operations Research Society Meeting in Montreal (May, 1988).

Keywords: Modeling, practice, computer science, software engineering

Introduction

One of the greatest challenges facing Management Science/Operations Research (MS/OR) for the rest of the 20th century is the design and construction of better computer-based modeling environments within which to carry out most kinds of applied model-based work.

Modeling environments have the potential to greatly increase the productivity of model-based work through better tools, to improve the quality of model-based work through better support for good modeling style and work practices, and to improve the frequency of use of MS/OR by bringing about a more comfortable working relationship between MS/OR professionals and their constituencies.

Five main characteristics appear necessary in order for a modeling environment to achieve these benefits. Section 1 explains, justifies, and gives some of the design implications of each of these in turn.

Section 2 discusses in some detail three of the main design challenges that follow from the design implications of Section 1. One of the conclusions that emerges is the strong pertinence of several subfields of Computer Science to the overall conception, design, and implementation of modeling environments.

The final section indicates briefly that the structured modeling approach is consistent with the desired characteristics, and that it can deal successfully with the three design challenges. However, we give no details because the aim of this paper is not to introduce structured modeling, but rather to encourage the MS/OR and allied communities to think more deeply about modeling environments.

Readers interested in research will find many intriguing research questions raised by the ideas sketched here. Opportunities for cross-fertilization with Computer Science are especially abundant. Readers interested in systems development likewise will find many challenges. Readers mainly concerned with real applications will find nothing here of immediate applicability, but we hope that they will be inspired to make known their views

Received August 1988; revised January 1989

on what would constitute a truly useful computer-based modeling environment.

Improved computer-based modeling environments are not the only route to improved productivity, quality, and frequency of use for MS/OR. Other promising approaches to these objectives, most of them complementary to improved modeling environments, have been proposed by other authors (e.g., Bonder (1979), Fortuin and Lootsma (1985), Gass (1987), Pruzan (1988)).

1. Desired characteristics of a modeling environment

In my view, a modeling environment should have certain properties if the three benefits just noted are to be achieved. It should:

1. nurture the entire modeling life-cycle, not just part of it;
2. be hospitable to decision and policy makers, not just to MS/OR professionals;
3. facilitate ongoing evolution of the models and systems built within it;
4. enable all of its inhabitants to 'speak' the same paradigm-neutral language for model definition;
5. facilitate good management of key resources, namely data, models, solvers, and knowledge derived from these.

Readers are requested to keep their own modeling tools in mind as each of these is discussed, and to ponder how well their tools measure up. The answers may lead to a greater appreciation for why these five characteristics are so important.

1.1. Computer-based life-cycle support

Every modeling project and every model-based system has a life-cycle that spans conception, development, use, and eventual termination. A true modeling environment should support the entire life-cycle from cradle to grave (Gass, 1987). Anything less would mean lost opportunity.

Different authors propose different versions of a typical life-cycle, usually with between 10 and 15 phases. The definition of the phases is not as important as the fact that they are strongly coupled: the output of one is the input of another, and iteration is common. Consequently, most phases call for integrated treatment to the extent that they are supported by computer-based tools. The alternative, using disjoint tools, is expensive and inefficient owing to resulting wasteful overlaps and burdensome interfaces.

It follows that a modeling environment requires a high degree of software integration, especially with respect to tools and utilities for communication (e.g., business graphics, telecommunications, and word processing or even desktop publishing), for organizing things and ideas (e.g., configuration and version control, database management, file management, outlining, and project management), and for quantitative analysis (e.g., data acquisition, interactive data analysis, graphics, mathematical, spreadsheet, and statistical programs).

Another design implication is that a modeling environment should provide for linkable libraries of data sources, models, solvers, and derived results—the main things to which software tools are applied in the course of the modeling life-cycle. (In this paper, the term 'solver' means an equation solver, optimizer, equilibrium calculator, query processor, or other model manipulation apparatus; and the term 'linkable' means tractable for purposes of coupling or integration.)

1.2. Hospitable to decision/policy makers

A true modeling environment should be hospitable not only to modeling professionals, but also to those for whom the modeling work is done. Important aspects of hospitability include clarity of model representation, intuitive organization, ease of learning and use, and provision for specialized, fool-proof access paths for non-technical and infrequent users.

Hospitability is important because the best results usually occur when tools can be used directly by those who need them rather than indirectly by intermediaries. Another reason is that non-modeling professionals have been taking up computer-based tools on a massive scale as an irreversible consequence of the personal computer revolution. Consider, for example, that one rudimentary modeling tool, the spreadsheet, is said to have more than 6 million users, to say nothing of the widespread use of database packages, project management packages, statistical packages, and other quantitatively oriented software for personal computers. Clearly, either MS/OR must make its

tools and approaches comfortable for its ultimate consumers, or those people will turn to other tools they already know how to use and leave ours to rust for lack of use.

The benefits of a more hospitable modeling environment accrue not only to outsiders; modeling professionals themselves can benefit from spending more time thinking about significant problems and issues and less about inessential technical details.

Attempting to cater to the needs of modeling professionals and non-modeling professionals in a single environment sometimes will raise difficult conflicts. When resolution one way or the other is necessary, usually it should favor the modeling professional, for fully meeting the needs of modeling professionals is the *sine qua non* of a useful modeling environment. Within that requirement, decision and policy makers should be able to use most of the higher level tools and functions of the environment with only minimal assistance from modeling professionals. Lower level tools and functions should be visible only to those with the expertise needed to use them.

A key design implication is that an executable modeling language is required. That is, a language sufficiently natural that non-modeling professionals can understand it with only a modest amount of training, and yet with a formal structure that computers can be programmed to 'understand'. The following quote from Bisschop and Meeraus (1982) underscores and elaborates on this important concept in the context of optimization-oriented modeling:

> ... Based on our experience we have concluded that the key to success is a modeling technology where only one model representation is needed to communicate with both humans and machines. The language should be a powerful notation which can express all the... information contained in the real-world problem. In addition,... the model representation should be such that a machine can take over the responsibility for verifying the algebraic correctness and completeness of the model.

Executable modeling languages are discussed further in Section 2.

Other design implications are that a modeling environment should have a personal computer/workstation implementation with a carefully designed user interface; should provide powerful completeness and consistency checks, as many users will be relatively naive; and should submerge MS/OR technology whenever possible (e.g., solver internals should be hidden).

1.3. Evolutionary flexibility

A true modeling environment should make it easy to change (correct, improve, tailor) things built within the environment and even the environment itself.

Flexibility is important because few modeling professionals ever get a model or a model-based system 100% right the first time. Even if by some miracle they do, the requirements usually change over time and thus will soon induce the need for revision. In any case, evolution may well be essential for genuine excellence. The need for flexibility has been widely recognized in the closely related field of software engineering, where most of the associated arguments and implications (such as the value of rapid prototyping) carry over to MS/OR with surprisingly little change (see, e.g., Brooks (1987)).

This characteristic, like the last one, calls for an executable modeling language. Changes should be made to a declarative specification of a model or model-based system, not to the (probably procedural) computer code that implements that specification. The 'executability' property of the modeling language should enable the changed code to be generated easily from the changed specification, the old code then being discarded. See Balzer, Cheatham and Green (1983) for a compelling exposition of this idea in the context of software engineering.

In fact, it is desirable to carry the idea of an executable modeling language one step farther: specify much of the modeling environment itself in an executable metalanguage so that it, too, will be easy to change. This is an extension of the idea of making a modeling environment easily reconfigurable in terms of the user interface and what utilities it offers.

1.4. Single paradigm-neutral model definition language

In a true modeling environment, there should be a *lingua franca* (common language) for model definition that is very broadly applicable and not biased toward any particular problem domain, modeling paradigm, or solver technology. This

probably is the most controversial of the five desired characteristics.

To see why one language is so desirable, one need only look at the current situation with its profusion of paradigm-specific styles for model representation (several each for decision trees, flow networks, Markov chains, mathematical programming, queueing systems, etc.). This profusion is only partially driven by the quest for clarity and efficiency. Much of it is the result of arbitrary choices, historical accidents, and lack of standards. The resulting multiplicity of representational styles impedes communication between modeling professionals and their clients (to say nothing of communication among professionals in different sub-fields), and is a technical impediment to the integration of models and systems—something that is often needed to attack comprehensive or strategic problems.

The most profound design implication of a lingua franca is the necessity of a general framework for conceptual modeling to serve as its foundation. The language itself should be understandable by people with minimal specialized training, and yet have a formal structure that is computationally tractable.

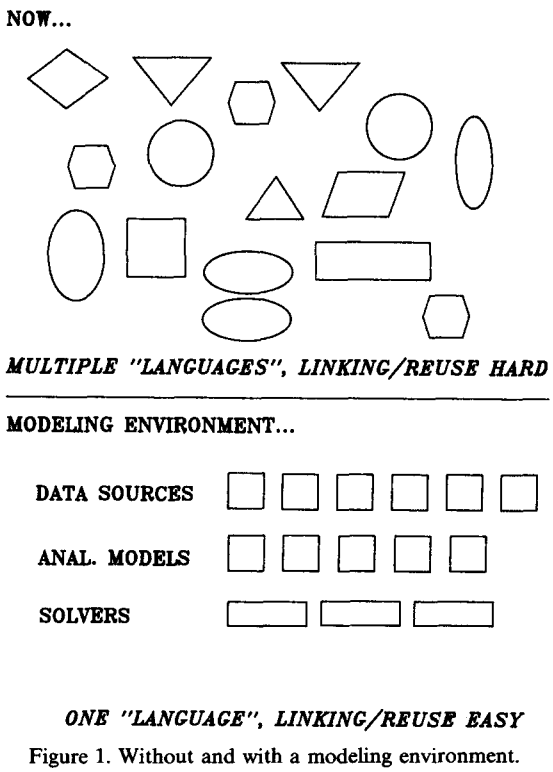

Figure 1. Without and with a modeling environment.

1.5. Good management of key resources

A true modeling environment should provide for the accumulation, sharing, and reuse of data, models, solvers, and derived knowledge. Accumulation is important because it is the basis of most progress. Sharing and reuse are important because they are a major source of gains in productivity; reinvention is just too expensive.

One evident design implication is that a modeling environment should incorporate extensive data management and model management facilities (e.g., Dolk and Konsynski (1985), Palmer (1984), and Sprague and Carlson (1982)). Another, which is shared with the first desired characteristic, is that a modeling environment should provide for linkable libraries of data sources, models, solvers, and results. A third is that stylistic guidelines are needed to avoid the confusion and anarchy caused by unnecessary differences in the representation of data, models, and derived knowledge.

It is also necessary to solve the much-lamented 'documentation problem' (e.g., Gass (1984)). Consider the problem of model documentation. Similar ideas may apply to the documentation of data, solvers, and derived knowledge. Human nature being what it is, there seem to be only two workable approaches: put simply, either make the documentation generate the model or make the model generate the documentation. The usual situation, in which both are generated separately without a causal link, fails so regularly in spite of good intentions that it no longer merits serious consideration. To be safe, a modeling system ought to take both approaches by using self-documenting representations and by providing automatic documentation capabilities.

Figure 1 is an impressionistic attempt to illustrate the significance of the last two desired characteristics. The top half is suggestive of the present situation in a typical organization: there could be a hodge-podge of 3 flat data files, 2 IMS databases, an INGRES database, a queueing model, a simulation model, 3 LP models (two with MaGen matrix generators and one with a FORTRAN matrix generator), 2 LP codes, a library of miscellaneous optimization routines, and some things that are unidentified because they have no

useful documentation and the developers are long gone.

The bottom half suggests how things could be in the same organization with a good modeling environment. Everything would be defined in one language, follow explicit stylistic guidelines, and be well organized. The same shape has been used for data sources and analytical models because, at an appropriate level of abstraction, there is no real difference between them; the notion of a 'model' should be general enough to subsume the notion of 'data'. Also, squares have been used for data sources and analytical models to suggest that almost any two (data/data, data/model, model/model) should be linkable. Rectangles have been used for solvers because they are a bit harder to link with data sources and models; but not very hard, for the world view of a solver always constitutes a kind of 'model'.

2. Major design challenges

The design implications noted in the previous section lead to these three major design challenges, among others:

1. a general framework for conceptual modeling;
2. executable modeling languages;
3. software integration.

Each is discussed in turn.

2.1. *First challenge: General framework for conceptual modeling*

A general framework for modeling concepts is the logical starting point for any sensible approach to the design of modeling environments. The framework should be

(a) generally applicable,

(b) rigorously formal,

(c) understandable and natural for the main players at each stage of the modeling life-cycle,

(d) paradigm-neutral yet compatible with most paradigms for modeling and model manipulation,

(e) consistent with 'good' modeling style (conducive to modularity, parsimony, etc.), and

(f) suitable for use as a foundation for the design of executable modeling languages (the second major design challenge).

It is interesting to note that, although the importance of such frameworks has not yet been recognized widely in MS/OR, analogous foundations have received much attention in certain neighboring fields. In particular, conceptual modeling is pursued as data modeling in database theory, as knowledge representation in artificial intelligence, and as programming language abstractions in high-level language design. See Brachman and Levesque (1985), Brodie et al. (1984), Shaw (1984), and Tsichritzis and Lochovsky (1982). To cite just one prominent example, the relational data model (Codd, 1970) has been immensely influential and successful in the database field. Its theory is elegant, powerful, and rich, and in practice is sweeping all else before it. Much of value can be carried over from conceptual modeling efforts in other fields to MS/OR.

For further discussion of the importance of conceptual modeling and pertinent interdisciplinary parallels, see pages 577 and 580 of Geoffrion (1987b).

Four approaches to the design of conceptual modeling frameworks for MS/OR are based, respectively, on: (1) entities, attributes, relations, and sets, (2) networks of modules, (3) attributed graphs, and (4) definitional systems.

The first approach has been around since the early 60s in the form of certain simulation languages, most notably GASP and SIMSCRIPT. Its most articulate proponent has been H. Markowitz, who has also demonstrated the generality of this approach by developing a powerful application development system called EAS-E (Markowitz, Malhotra and Pazel, 1984). Also in this general category are the relational data model extensions of Blanning (1987), the popular entity-relationship approach of Chen (1976), and the Extended Structured Systems Approach of Müller-Merbach (1983).

The second approach has its roots in systems theory. It views a model as an interconnected network of modules, each with input(s), output(s), and an internal transformation rule. An example in the context of integrated energy models is Hogan and Weyant (1983). A more recent development in this vein is the 'systems framework' of Muhanna (1987), which seems mainly to be concerned with describing models 'in the large' rather than with the smaller details. See also Liang (1986).

The third approach adopts attributed graphs as

its conceptual formalism. 'Nodes' are classified into 'node types', 'arcs' are classified into 'arc types', and each node type and each arc type can have its own list of 'attributes'. Attributed graphs have been used often in MS/OR and related fields. What has been lacking until recently are good ways to characterize the common structure shared by important classes of graphs. Jones (1985) overcomes this shortcoming by adapting the theory of graph grammars to make attributed graphs a much more powerful framework for conceptual modeling. See also Göttler (1987) and Jones (1988).

The fourth approach views a model as a collection of definitions that formalizes and organizes what is known or assumed about what is being modeled. This view is at the core of 'structured modeling' (Geoffrion, 1987b). More specifically, structured modeling formalizes a particular kind of definitional system, namely one that is correlated, acyclic, grouped, hierarchical, typed, and interpreted (Geoffrion, 1989).

To leave the impression that the four approaches are totally distinct would be wrong. A close examination shows that they have much in common. Further discussion of these and other approaches to conceptual modeling can be found in Geoffrion (1987a).

2.2. *Second challenge: Executable modeling languages*

An executable modeling language is needed to support whatever general framework is adopted for conceptual modeling. We can conclude from Section 1 that an executable modeling language should possess certain characteristics. In particular, points (a) through (e) given for the first design challenge apply here also. Some of these must be interpreted a bit differently because we are talking about a language rather than a general framework; for example, 'understandable and natural for the main players at each stage of the modeling life-cycle' has to do, among another things, with being declarative rather than procedural and highly mnemonic rather than cryptic. We can also conclude from Section 1 that an executable modeling language should be able to perform extensive consistency checks.

The adjective 'executable' refers to *functions* that programs in the modeling environment should be able to perform upon receiving a model written in an executable modeling language. (Note to computer scientists: this differs from the usual definition of 'executable'.) We make a distinction between a 'model instance' and a 'model class'. The former is a fully specified model including all relevant data, while the latter is a familial collection of model instances. One of the precepts of good modeling requires that the modeling language should be able to express both of these, with the former represented as a particularization of the latter.

For a model class expressed in an executable modeling language, desirable functions include the following:

1. *Error-trapping.* It is important to detect lexical, syntactic, and checkable semantic mistakes. Examples of these could be, respectively: a misspelled keyword, unbalanced parentheses, and a reference to something that is undefined.

2. *Automatic documentation.* There are obvious benefits to the automatic production of various kinds of documentation, such as an indirect cross-reference map of model elements.

3. *Solver interface setup.* Setting up solver interfaces for such model manipulation tasks as interactive expression evaluation, query processing, inference, and optimization has traditionally required tedious programming. Ideally, no such programming should be necessary for a model defined in an executable modeling language. For example, creation of the input file needed by an LP solver should be accomplished automatically by its interface once model instance data are provided. Other interfaces could enable a query processor (as in database systems), an inference engine (as in expert systems), or an expression evaluator (as in spreadsheets) to be used with no programming effort by the user.

4. *Smart loader / editor for detailed data.* This function involves (a) setting up suitable data structures to hold the detailed data needed to specify a particular model instance within the given model class, (b) creating suitable user-accessible input structures (perhaps tables) through which data can be entered and then edited, and (c) tailoring an editor for data entry and editing that 'understands' the model class at hand and uses that understanding to relieve the user of as many burdens as possible. As an example of the last subfunction, if a model class involves the cross product of two lists (say, to set up all possible

transportation links from plants to warehouses), then the cross product should be created automatically once both lists have been entered.

Similar modeling environment functions are appropriate in connection with particular model instances. Obviously error trapping and automatic documentation continue to be very important, although the particulars change. For example, error trapping takes on the character of 'run time' checks rather than 'compile time' checks (to use terminology from compiler theory). Invoking solvers and editing data replace functions 3 and 4 above; both should be very easy if functions 3 and 4 are done well.

Designing an executable modeling language and its associated user interface involves several important trade-offs. Some are obvious, like generality versus executability (e.g., English is at one end of the spectrum and the MS-DOS command language is toward the other). Other trade-offs are less obvious, like the conflict between direct manipulation and journalizability. The former, exemplified by spreadsheet programs and display editors, tends to produce high user productivity and enthusiasm (Shneiderman, 1987). But direct manipulation makes it difficult to journalize one's modeling work in written form for purposes of documentation, of leaving an audit trail, and of storage for later editing and reuse. Two additional trade-offs deserving consideration are achieving flexibility versus enforcing 'good' modeling style (like the separation of general model structure from detailed data), and making light demands on the modeler versus achieving powerful checks on model completeness and consistency.

Those who are inclined toward research will find that many of these design issues pose juicy research problems. A fine example of a research contribution to the last issue mentioned is Bradley and Clemence (1987), which develops a 'type calculus' by which units of measurement can be introduced into many algebraic modeling languages in a rigorous, elegant, and active way. The rewards for making some additional demands on the model designer include some powerful consistency checks and novel services for automatic units conversion and scale factoring.

It is one thing to design an executable modeling language together with the associated modeling environment functionality, but it is quite another thing to actually achieve a successful implementation. The design of an executable modeling language and the engineering of its implementation obviously must go hand in hand.

It is evident that compiler and related technologies from computer science are needed. Moreover, the need for evolutionary flexibility of the modeling environment itself implies the need for program generation technology that can accept a formal specification of the grammar of the executable modeling language and of optional modeling environment features, and generate the programs needed to provide the desired functionality and features. For example, if one wishes to add calendar dates to an executable modeling language, then it should only be necessary to change the formal grammar and regenerate the affected modeling environment programs. Or if one wishes to drop business graphics capabilities so that the modeling environment will fit on a 2 megabyte portable computer with a 20 megabyte drive, then this should be a simple reconfiguration task.

There is a lot of work going on in computer science that may help solve the technical difficulties associated with modeling environments. Three areas especially deserve mention as probable sources of applicable technology. All three are thriving at present, owing in part to the much-publicized 'software crisis'. The first is programming environments; see, for example, Balzer (1985), Barstow et al. (1984), Conradi et al. (1986), and Henderson and Notkin (1987) (the guest editors' introduction to a special issue devoted to integrated design and programming environments). The second is computer-aided software engineering (CASE); see, for example, Chikofsky (1988) (the guest editor's introduction to a special issue devoted to CASE). So-called 'back end' or 'lower' CASE is particularly pertinent because it is more concerned with the thorny automatic code generation problem than 'front end' or 'upper' CASE. The third pertinent area is reusability; see, for example, Freeman (1987a). Among the topics included here are very high level program-producing systems such as DRACO (e.g., Freeman (1987b)).

It should be obvious that all three areas are closely related to one another. Their influence on developers of modeling tools is in its infancy. The infusion of such ideas to date is most clearly discernible in the subfield of simulation, which perhaps can be explained by the traditionally strong identification of simulation modeling with

computer programming. See, e.g., Balci and Nance (1987), Donohue et al. (1986), Henriksen (1983), and Muntz and Parker (1988).

Numerous executable languages useful for modeling have been designed and implemented. Many of these are mentioned or discussed in Geoffrion (1987a), and so need not be listed here. We mention only that there has been considerable recent interest in languages for mathematical programming, including AMPL (Fourer, Gay and Kernighan, 1987), CAMPS (Lucas and Mitra, 1985), GAMS (Bisschop and Meeraus, 1982), LINGO (Cunningham and Schrage, 1988), LPL (Hürlimann and Kohlas, 1988), MIMI/LP (from Chesapeake Decision Sciences, Inc.), and PAM (from Ketron, Inc.). Other languages could be added from the neighboring model-related fields of artificial intelligence, database management, and programming languages, not to mention the important advent of so-called fourth generation languages (claimed to be learnable in no more than two days and to boost productivity by one or two orders of magnitude). Some useful references are Date (1986), Jarke and Vassiliou (1985), Martin (1985), Rich (1983), and Shaw (1984).

Although the accomplishments and usefulness of existing languages are impressive, one may reasonably conclude that none is fully adequate for the kind of modeling environment envisioned here: for each language, either the conceptual modeling framework it is intended to support is not sufficiently clear, or it does not meet the requirements posed at the outset, or it lacks the breadth of functionality called for earlier, or it has a combination of these deficiencies. Nevertheless, many of these languages are instructive precursors of the kinds of languages needed for true modeling environments.

For other critiques of the adequacy of existing executable languages in two important subfields of MS/OR, see the excellent reviews by Fourer (1983) and Overstreet et al. (1986).

2.3. *Third challenge: Software integration*

It is clear from what has come before that a modeling environment is an ambitious undertaking. It includes linkable libraries of data sources, models, solvers, and derived results. It includes programs to supply the four kinds of functionality needed for 'executability'. And it includes many tools and utilities needed for total life-cycle support of modeling work.

Nearly all of these components should, for reasons cited in Section 1, be well integrated.

Not only must software integration be accomplished on a grand scale, but it should be done without compromising the five desired characteristics of modeling environments. Hence there are requirements like a good user interface, easy reconfigurability, and so on.

Integration on this scale poses numerous problems, not the least of which is how to accomplish it technically. A promising line of attack is to attempt to apply what has been learned by the programming environment community. Two good reviews of that work are Barstow, Shrobe and Sandewall (1984) (see especially the chapters on INTERLISP, SMALLTALK, and UNIX) and Conradi, Didriksen and Wanvik (1986) (see especially the section on Tool Integration and the article by Kaplan et al.). See also Clemm and Osterweil (1986), which describes an object management approach that has been used successfully several times; and Vo (1985), which describes the integration approach used by a UNIX-based analytical modeling environment at AT&T Bell Laboratories called ANALYTICOL, for which a five-fold productivity gain is claimed.

Certainly it is impractical to custom build every utility and functional module. Ground-up construction of all parts of a modeling environment would be prohibitively expensive and likely to yield a lower quality result than using proven code written by inspired specialists. For example, a single excellent text editor probably should be adopted for use in all parts of the modeling environment where a more advanced editor cannot be justified (e.g., a structure editor, Reps and Teitelbaum (1987)).

One might build on an existing platform such as UNIX, as ANALYTICOL does (Childs and Meacham, 1985). Or one might build a modeling environment around a suitable, and probably relational, database system like INGRES. It is noteworthy that prototype extensible database systems are in development that may prove to be better hosts than any existing DBMS (e.g., Batory and Mannino (1986)). A related possibility would be

to build on top of an information resource dictionary system (Dolk, 1988).

Software integration is about how to achieve acceptable performance within available machine resources. But it is also, and very importantly, about how to achieve conceptual unity of the many parts and tools that make up a modeling environment. Conceptual unity is an essential pursuit if any but the most diligent and experienced users of a modeling environment are to profit from its rich variety of capabilities.

A promising approach for achieving conceptual unity is this: use the conceptual modeling framework (see the first challenge) to 'model' most or all of the modeling environment's parts and tools. This will be possible if the framework is sufficiently general. The advantages of doing this should be obvious. They include reduced learning time because the structure of the environment will be represented in a familiar way, and the ability to manipulate a detailed description of the environment using the specialized tools of the environment itself (e.g., a query processor can be used to answer questions concerning features of the environment).

An appeal for conceptual unity through unified modeling has been made in the analogous context of computer software systems by Markowitz (1978). Here is a short quote from the closing section of that paper, from which the reader may be able to glimpse the basic idea:

> My hypothesis is that software systems have moderately complex EAS structures; that most subsystems or functional areas like job control or spooling, have fairly simple EAS structures; and that the computer system would be an order of magnitude easier to grasp if its EAS structure were documented, the user were given a meaningful response to any request to take any elemental action on any part of the system, and these requests could be made in (one or another) language style applicable *throughout* the system.

('EAS' stands for the Entity-Attribute-Set formalism, which was mentioned briefly in the earlier discussion of four basic approaches to general frameworks for conceptual modeling.) Markowitz later implemented this idea partially in the application development system EAS-E cited earlier, for which substantial advantages were claimed (Markowitz, Malhotra and Pazel, 1984).

3. Conclusion

This paper has argued that five characteristics are particularly desirable for a good modeling environment, and has discussed three of the main design challenges which follow. (These characteristics and challenges are listed at the beginning of Sections 1 and 2.) It is not important that the reader accept all that has been said along these lines. What is important to the aim of this paper is that readers think about their current modeling tools in terms of the five characteristics and perhaps others that come to mind, and that they ponder how to conquer design challenges like those discussed here:

* To what extent *do* the current tools possess the five characteristics?
* To what extent *should* modeling tools possess these characteristics?
* How do current tools deal with the three design challenges?
* What can be done to help close the gaps?

Probably there is no one correct set of answers to the issues raised. However, one promising approach is emerging from work on structured modeling. Structured modeling is proving to be consistent with all five desirable characteristics, and the three major design challenges discussed in this paper are yielding to sustained attack:

* A coherent framework for conceptual modeling, based on the extremely general idea of a definitional system as mentioned earlier, is now in place (Geoffrion, 1987b; 1989).
* An executable modeling language called SML that supports this conceptual modeling framework also is in place (Geoffrion, 1988).
* One of several possible approaches to the software integration challenge is being pursued in a prototype implementation called FW/SM (user and technical documentation in preparation). Other prototype implementations also exist or are under development. The great generality of the definitional system framework makes structured modeling a good candidate for pursuing the modeling approach to conceptual unity described toward the end of Section 2. This will be undertaken as soon as FW/SM stabilizes.

Whatever degree of success may be achieved ultimately by the structured modeling approach,

there is ample room for a variety of different approaches to the design of modeling environments that will enable MS/OR to achieve, at long last, its full potential.

Acknowledgments

I gratefully acknowledge the helpful comments of Leonard Fortuin, Melanie Lenard, Laurel Neustadter, and Fernando Vicuña.

This work was supported partially by the National Science Foundation, the Office of Naval Research, the Navy Personnel Research and Development Center (San Diego), and Shell Development Company. The views expressed are those of the author and not of the sponsoring agencies.

References

Balci, O., and Nance, R.E. (1987), "Simulation support: Prototyping the automation-based paradigm", *Proc. Winter Simulation Conference.*

Balzer, R. (1985), "A 15 year perspective on automatic programming", *IEEE Transactions on Software Engineering* 11/11 (November), 1257–1268.

Balzer, R., Cheatham, T.E., Jr., and Green, C. (1983), "Software Technology in the 1990's: Using a New Paradigm", *Computer* 16/11 (November), 39–45.

Barstow, D.R., Shrobe, H.E., and Sandewall, E., (eds.) (1984), *Interactive Programming Environments*, McGraw-Hill, New York.

Batory, D.S., and Mannino, M. (1986), "Panel on extensible database systems", *ACM SIGMOD 86.*

Bisschop, J., and Meeraus, A. (1982), "On the development of a general algebraic modeling system in a strategic planning environment", *Math. Programming Studies 20* (October), North-Holland, Amsterdam, 1–29.

Blanning, R.W. (1987), "A relational theory of model management", in: C.W. Holsapple and A.B. Whinston (eds.), *Decision Support Systems: Theory and Application*, Springer-Verlag, Berlin–New York.

Bonder, S. (1979), "Changing the future of operations research", *Operations Research*, 27/2 (March–April), pp. 209–224.

Brachman, R.J., and Levesque, H.J. (1985), *Readings in Knowledge Representation*, Morgan Kaufmann, Los Altos, CA.

Bradley, G.H., and Clemence, R.D., Jr. (1987), "A type calculus for executable modeling languages", *IMA Journal of Mathematics in Management* 1/4, 277–291.

Brodie, M., Mylopoulos, J., and Schmidt, J. (1984), *On Conceptual Modeling*, Springer-Verlag, Berlin–New York.

Brooks, F. (1987), "No silver bullet: Essence and accidents of software engineering", *Computer* 20/4 (April), 10–19.

Chen, P. (1976), "Entity-relationship model: Toward a unified view of data", *ACM Transactions on Database Systems* 1/1 (March) 9–36.

Chikofsky, E.J. (1988), "Software technology people can really use", *IEEE Software* 5/2 (March), 8–10.

Childs, C., and Meacham, C.R. (1985), "ANALYTICOL—An analytical computing environment", *AT&T Technical Journal* 64/9 (November), 1195–2007.

Clemm, G., and Osterweil, L. (1986), "A mechanism for environment integration", CU-CS-323-86, Dept. of Computer Science, University of Colorado, Boulder, April.

Codd, E.F. (1970), "A relational model of data for large shared data banks", *Communications ACM* 13/6 (June) 377–387.

Conradi, R., Didriksen, T.M., and Wanvik, D.H. (eds.) (1986), *Advanced Programming Environments*, Lecture Notes in Computer Science No. 244, Springer-Verlag, Berlin–New York.

Cunningham, K., and Schrage, L. (1988), "The LINGO Modeling Language", LINDO Systems, Inc., May 3.

Date, C.J. (1986), *An Introduction to Database Systems*, Volume 1, Fourth Edition, Addison-Wesley, Reading, MA.

Dolk, D.R. (1988), "Model management and structured modeling: The role of an information resource dictionary system", *Communications ACM* 31/6 (June), 704–718.

Dolk, D.R., and Konsynski, B.R. (1985), "Model management in organizations", *Information and Management* 9/1, 35–47.

Donohue, G., Bennett, B., and Hertzog, J. (1986), "The RAND military operations simulation facility: An overview", N-2428-RC, April.

Fortuin, L., and Lootsma, F.A. (1985), "Future directions in operations research", in: A.H.G. Rinnooy Kan (ed.), *New Challenges for Management Research*, North-Holland, Amsterdam.

Fourer, R. (1983), "Modeling languages versus matrix generators for linear programming", *ACM Transactions on Mathematical Software* 9/2 (June), 143–183.

Fourer, R., Gay, D.M., and Kernighan, B.W. (1987), "AMPL: A mathematical programming language", Computing Science Technical Report No. 133, AT&T Bell Laboratories, Murray Hill, NJ, January.

Freeman, P. (1987a), *Software Reusability*, IEEE Cat. No. EH0256-8, Computer Society Press, Washington, DC.

Freeman, P. (1987b), "A conceptual analysis of the Draco approach to constructing software systems", *IEEE Transactions on Software Engineering* 13/7 (July), 830–844.

Gass, S.I. (1984), "Documenting a computer-based model", *Interfaces* 14/3 (May–June), 84–93.

Gass, S.I. (1987), "Managing the modeling process: A personal reflection", *European Journal of Operational Research* 31/1 (July), 1–8.

Geoffrion, A.M. (1987a), "Modeling approaches and systems related to structured modeling", Working Paper 339, Western Management Science Institute, UCLA, February.

Geoffrion, A.M. (1987b), "An introduction to structured modeling", *Management Science* 33/5 (May) 547–588. A version that includes a section on implementation is available as Working Paper 338, Western Management Science Institute, UCLA, 75 pages, most recently revised 3/88.

Geoffrion, A.M. (1988), "SML: A model definition language for structured modeling", Working Paper 360, Western Management Science Institute, UCLA, May.

Geoffrion, A.M. (1989), "The formal aspects of structured modeling", *Operations Research* 37/1 (January-February), 30–51.

Göttler, H. (1987), "Graph grammars and diagram editing", in: H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld (eds.), *Graph Grammars and Their Application to Computer Science*, Springer-Verlag, Berlin.

Henderson, P.B., and Notkin, D. (1987), "Integrated design and programming environments", *Computer* 20/11 (November), 12–16.

Henriksen, J.O. (1983), "The integrated simulation environment: Simulation software of the 1990s", *Operations Research* 31/6 (November-December), 1053–1073.

Hogan, W.W., and Weyant, J.P. (1983), "Methods and algorithms for energy model composition: Optimization in a network of process models", in: B. Lev (ed.), *Energy Models and Studies*, North-Holland, Amsterdam.

Hürlimann, T. and Kohlas, J. (1987), "LPL: A structured language for linear programming modeling", *OR Spektrum* 10, 55–63.

Jarke, M., and Vassiliou, Y. (1985), "A framework for choosing a database query language", *ACM Computing Surveys* 17/3 (September), 313–340.

Jones, C.V. (1985), "Graph-based models", Ph.D. Thesis, Cornell University.

Jones, C.V. (1988), "An introduction to graph-based modeling systems", Working Paper 88-10-2, Department of Decision Sciences, The Wharton School, University of Pennsylvania.

Liang, T.P. (1986), "Toward the development of a knowledge-based model management system", Ph.D. Dissertation, University of Pennsylvania.

Lucas, C., and Mitra, G. (1985), "CAMPS: Preliminary User Manual", Department of Mathematics and Statistics, Brunel University, Middlesex, U.K., July.

Markowitz, H.M. (1978), "SIMSCRIPT: Past, present, and some thoughts about the future", RC 7075 (#30326), Thomas J. Watson Research Center, Yorktown Heights, April 20.

Markowitz, H.M., Malhotra, A., and Pazel, D.P. (1984), "The EAS-E application development system: Principles and language summary", *Communications ACM* 27/8 (August) 785–799.

Martin, J. (1985), *Fourth-Generation Languages*, Prentice-Hall, Englewood Cliffs, NJ.

Muhanna, W.A. (1987), "A systems framework for model software management in organizations", Ph.D. Dissertation, School of Business, Univ. of Wisconsin-Madison.

Müller-Merbach, H. (1983), "Model design based on the systems approach", *J. Operational Research Society* 34/8, 739–751.

Muntz, R.R., and Parker, D.S. (1988), "Tangram: Project overview", CSD-880032, Computer Science Department, University of California, Los Angeles, April.

Overstreet, C.M., Nance, R.E., Balci, O., and Barger, L.F. (1986), "Specification languages: Understanding their role in simulation model development", Technical Report TR-87-7, Dept. of Computer Science, Virginia Tech, December.

Palmer, K. (1984), *A Model Management Framework for Mathematical Programming*, Wiley, New York.

Pruzan, P. (1988), "Systemic OR and operational systems science", *European Journal of Operational Research* 37/1 (October) 34–41.

Reps, T., and Teitelbaum, T. (1987), "Language processing in program editors", *Computer* 20/11 (November) 29–40.

Rich, E. (1983), *Artificial Inteligence*, McGraw-Hill, New York.

Shaw, M. (1984), "The impact of modeling and abstraction concerns on modern programming languages", in: Brodie et al. (1984).

Shneiderman, B. (1987), *Designing the User Interface*, Addison-Wesley, Reading, MA.

Sprague, R.H., Jr., and Carlson, E.D. (1982), *Building Effective Decision Support Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Tsichritzis, D.C. and Lochovsky, F.H. (1982), *Data Models*, Prentice-Hall, Englewood Cliffs, NJ.

Vo, K.-P. (1985), "IFS—A tool to build integrated, interactive application software", *AT&T Technical Journal* 64/9 (November) 2097–2117.