Integrating Model Management with Data Management in Decision Support Systems

Ting-Peng LIANG
Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, U.S.A.

Because of the increasing complexity of the problems faced by decision makers and the nature of decision support systems, decision models are becoming more and more crucial to their success. In this paper, a general framework for model management, which can integrate model management and data management and handle issues in model management such as model creation, model modification and model use is proposed.

Keywords: Decision Models; Model Management; Data Management; Data–Model Interface; Data–Model Management Integration

Ting-Peng Liang is completing his dissertation in the Department of Decision Sciences at the Wharton School of the University of Pennsylvania, He holds an M.A. degree in Decision Sciences, an M.B.A., and a B.S. degree in Engineering. Before pursuing his doctor's degree, he had spent two years in the military and three years in industry as a senior researcher, including one year as a research group director. He is the author of two books in information systems (in Chinese) and has published papers in the proceedings of national and local conferences.

North-Holland
Decision Support Systems 1 (1985) 221–232

1. Introduction

Decision support systems (DSSs) are designed to support semi-structured or unstructured decisions in order to improve the effectiveness of decision making. In general, a decision support system includes three highly interrelated components (database, model base, and user interface which include dialogue management and report generators) and interfaces to integrate these components (for example [1,2]). The database provides the data required for decision making, the model base provides appropriate models, and the user interface provides a channel so that the user can communicate with the system.

Of the interfaces integrating the components, the data–model interface is certainly one of the most important because both the input and output of a model are data and should be, in most cases, provided by or stored in the database of the system. Currently, most database management systems include a programming language so that the application program can access the data in the database via interface programs or embedded key words in the application program (e.g., Data Manipulation Language of the CODASYL model). However, this kind of linkage between a model and a database is on an individual model basis. It suffers from drawbacks such as redundancy in programming effort, difficulty in maintenance, and lack of flexibility.

In addition, the growing use of decision models has attracted research in model management. The integration of model management and data management is certainly one of the major considerations in developing a general model management system. There are three primary reasons to adopt decision models in DSSs:

(i) The rapid development of Management Science/Operations Research (MS/OR) since World War II has made some decision models very powerful in dealing with complex problems.

(ii) The complexity of the decision problems faced by the decision maker increases over time.

(iii) A data-oriented DSS has very limited capability for transferring data into useful information and, in most cases, can only support the intelligence stage of the decision process, while decision models can support the design and the choice stages. For example, the value of a portfolio selection DSS may significantly decrease if it is not armed with a powerful portfolio analysis model

Although a number of current systems such as Lotus 1-2-3 and IFPS claim to have the capabilities of model management, most of them still interface databases with models on an individual model basis and lack a common interface to integrate them. The primary reason for this is that we do not yet have a general framework to guide the development of model management systems. Most of the previous researchers in model management have tried to develop systems that were similar to a database management system. For example, Elam employed the entity–relationship approach [3–5] Blanning adopted the relational approach [6–14], and Konsynski and Dolk used the CODASYL approach [15,16]. Models are certainly similar to data in some sense, but they also have many differences [9]. Therefore, using an approach similar to data management without taking into account the specific characteristics of models and model management may have problems in both the integration and the implementation of model management and data management.

The purpose of this article is to propose a general framework for model management which can handle not only those functions facilitating model creation, model modification and maintenance, model retrieval. but also the integration of model management and data management. Although it may not be easy to define standard requirements for developing a model management system, the following four points can be considered as minimum requirements for a model management system:

1. It should have the capability of supporting various users with different requirements. Although there are many different ways to classify users, model users can be divided into three different roles: the information user (the decision maker); the model builder; and the toolsmith. A good model management system should support all three roles.
2. It should have the capability of integrating model and data in a manner other than the individual model basis in order to reduce programming work and the difficulty of maintaining the model base. In other words, the system should provide a common interface between the model base and the data base.
3. It should communicate with the user through the common user interface component of the DSS.
4. It should provide an environment to support model sharing. Since many DSSs are developed in a distributed environment and for multiple users, model sharing is one of the most important considerations in designing model management systems.

In the remaining parts of this paper, the evolution of model management will be described at first, followed by a general framework for model management and the process showing how model management can be integrated with data management. Finally, an illustrative example is provided.

2. Evolution of Model Management and Data Management

From the software design point of view, the integration of model management and data management has evolved through three generations, as shown in Fig. 1. In the first generation, models and data were integrated on the basis of an individual application program. The application programmer was not only responsible for developing the computational procedures of the program but also for designing the data structures and integrating the two. No databases or ready-to-run subroutines were available at that time. This approach was certainly very efficient for the machine because each program was tailored for the specific application. However, in addition to being time-consuming and expensive to develop models, it was also difficult to maintain them.

Because many application programs may use some common computational procedures, these procedures were collected in a package in order to reduce the redundancy among programs. These kinds of subroutine packages, such as IBM's Sci-

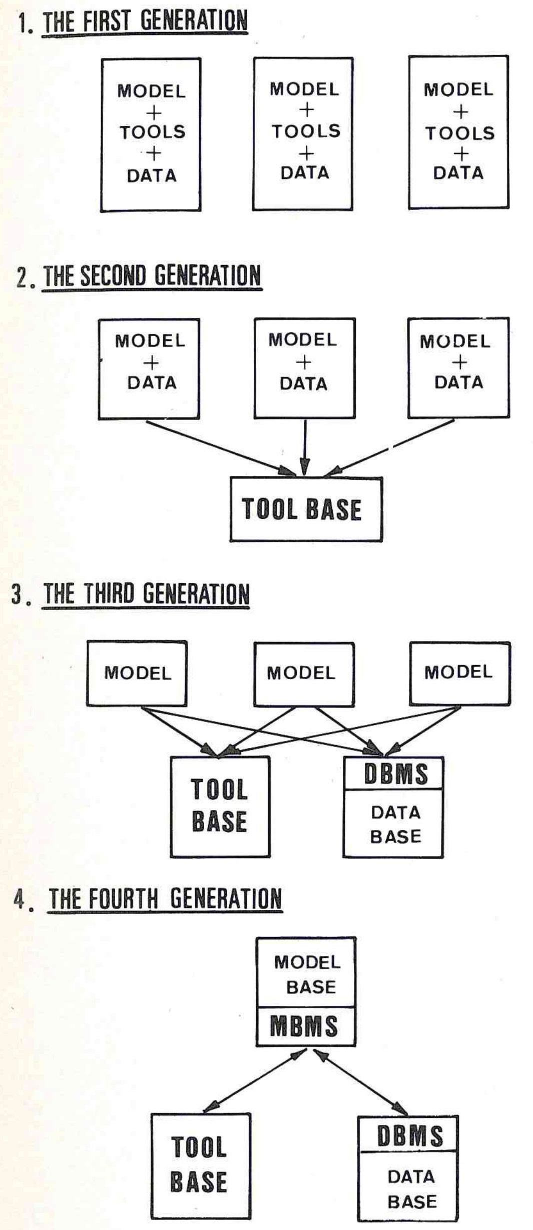

Fig. 1. The Evolution of Model Management.

entific Subroutine Packages written in FORTRAN, that provided a library of subroutines for statistical and matrix operations, featured the second generation. Although the computational subroutine package may be classified as a model base (for example, [2]), it is called the 'tool base' in order to distinguish from the 'model base' in this paper.

One of the most significant distinctions between models and tools is that models are problemdomain dependent while computational tools are problemdomain independent. From Fig. 2, we can see that a model is an abstraction of a real world problem, while the computational tool is a function that can transform inputs of a model to some useful outputs for solving the problem. Based on this criterion, the simplex method is a tool, but the formulation for solving a production mix problem by using the simplex method is a model.

The advances in database management systems since the late 60's led to the development of the third generation, the current state. The most important feature of this generation is the wide use of the database management system to manage a comprehensive database. Application programs can retrieve date or store results in the database by using the Data Manipulation Language (DML) or other database management languages. In this generation, models and data are integrated by way of the database management language. If the number of decision models employed is not large, this approach is certainly an efficient way. However, if we have a number of decision models to be managed, then the performance of this approach will not be satisfactory. The problems that occurred in data management two decades ago (such as inconsistency and redundancy) will definitely happen in the third generation model management systems. Therefore, a generalized model base management system is required.

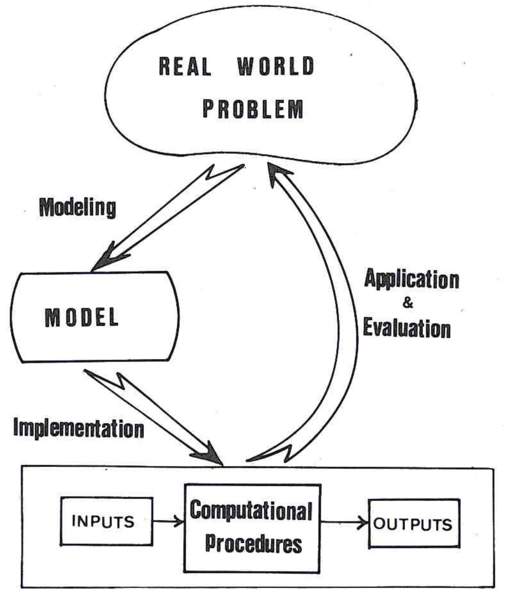

Fig. 2. The Model Development Cycle.

The feature of the fourth generation is that the integration of models with data is no longer on the individual model basis. Rather, a model management system will take over and be responsible for all communication between the model base and the database.

Advantages of the fourth generation integration between models and data, which are similar to the advantages of the development of database management systems, include the following:

(1) *Redundancy can be reduced:*

The model builder does not need to worry about the integration between the model and the data since the system will handle this issue automatically. This will certainly reduce the programming work and make the model building easier.

(2) *Models can be shared:*

Many DSSs are developed in a distributed environment or for multiple users. Model sharing is certainly an important consideration. With a good model base management system, all authorized users can access the model with minimum additional effort once the model has been built.

(3) *Flexibility can be increased:*

The model management system provides more flexibility to the DSS and serves as a buffer between the model base and the database so that the impact of any changes in the database on models can be reduced. For example, if a new database management system is adopted, all changes can be absorbed by the model base management system. The model management system would have to be modified in order to adapt to the new DBMS, but all models in the model base still work without the need to change anything. Otherwise, we would have to modify all models.

In summary, with the first and second generations, integration between models and data is basically within the individual application program. The difference between them is that the latter has a common tool base that the application program can use by invoking commands such as CALL and RETURN. The major difference between the second and third generation is that in the third generation the integration is accomplished by linking models with the database management language rather than with the data directly. Fourth generation integration is accomplished through the linkage between the model base management system and the database management system. Neither direct relations between models and data nor between the model and database management systems exist.

3. Previous Research in Model Management

Models can be viewed as data, statements, or subroutines [2]. Regarding fourth generation model management, previous research includes two of these three concepts: treating a model as a subroutine and treating a model as a datum. However, due to the thorough investigation of database management in the past two decades and the importance of the compatibility between the model base and database in DSS, most researchers view models as data. They focus on building a model management system similar to database management systems either according to the CODASYL network or relational approach [1,4,9–12,14, 16–18]. Others tried to facilitate the modeling process by using AI techniques, such as SI–NETs [5] or natural language processing.

Early work in MMS were focused on introducing the concept of model management and describing the functions that a model management system should have. Will introduced the term ‘model bank’ and argued that the five functions a model management system should have are: model description; manipulation; scheduling; execution; and information display [19]. Sprague and Watson discussed the same issue from a different point of view and proposed that four model management functions are essential: model generation; restructure; update; and report generation [20]. Both of these papers are pioneering works in this area and have touched the core of model management. However, neither delineated the idea to a general framework nor did they provide enough guidelines for designing a system.

Elam et al. came up with the idea of developing a knowledge-based model management system [3,4,5]. They began with an entity–relationship approach and ended up with a model including five components: analyzer; builder; interrogator; processor; and knowledge base [3–5]. Basically, they treat a model as a subroutine and tries to build some capabilities of automatic modeling into

the knowledge base with the SI-NET (a graphical language composed of nodes and links for describing concepts and the interrelationships between these concepts). They mentioned four different roles of users explicitly (although the names of roles were different in two papers), but did not tailor a framework to fit them.

Konsynski and Blanning, on the other hand, viewed models as data. Konsynski and Dolk introduced a generalized model management system which treates models as a data abstraction consisting of elements, equations, and solution procedures [15,16]. While this approach basically corresponds to the CODASYL network, it does not consider different users' roles and provides no external schema which reflects the direct mapping relationships between input and output to information users. Blanning's model was a relational one that treated a model as a properly restricted subset of the Cartesian cross product of its inputs and its outputs. He suggested that a model bank may be viewed as a set of virtual relations with input and output attributes and functional dependencies between them, just as a set of relations with key and content attributes [9–14,17]. This approach appropriately reflects the information user's point of view to the model and certainly can borrow some of the theoretical background from relational database management systems. However, multiple mappings and the link among models are two major problems since it does not explicitly handle the computational procedures, the kernel of models. For example, an integer programming model may be solved by the cutting-stock algorithm, the Lagrangian relaxation or the Kuhn–Tucker condition, and a problem may be formulated both as an integer programming problem or a linear programming problem. Therefore, the relationships between inputs and outputs may not be unique.

In general, previous works have revealed some interesting issues, but they could be further developed to reach a general framework. Elam et al. noticed the requirements of different users and specified the possible knowledge base but did not provide a detailed framework for all uses. The relational approach is good on the external levels which clearly represents the mapping from input to outputs but it needs a mechanism to handle the modeling in a multi-mapping situation. Therefore, a framework that can fulfil both the requirements of different users and those technical requirements, and that can also fully integrate model management with data management is needed. This is described in Section 4.

4. Users' Roles

Users with different responsibilities and different backgrounds may play different roles in using a model management system. Assessing the user's role can identify different requirements of various roles and make the system more effective. Although several names have been used to classify users' roles under different situations, users of model management systems can be classified into three categories: information user; model builder; and toolsmith. Information users are decision makers or their representatives who need the output information to make decisions. They are interested in the output and its correctness but may not understand or have any interest in how the model works. Model builders are responsible for developing a model that can provide the required information to the decision maker. They deal with what tools are available for developing the model, how to develop the model, what input data are required, and the validity of the model. They may not care how information users use the output of the model. The toolsmith is a very technically oriented person who is responsible for developing new computational algorithms or coding computational algorithm into programs that can be integrated into the model management system as tools. A toolsmith is concerned with the correctness and efficiency of an algorithm, the coding of programs, and the development of new algorithms.

Although there are three different roles, it does not mean that each user should only play one role. A user may play more than one role, and a role may involve more than one user. Which of the roles an individual assumes depends upon the complexity of the problem and the model, the background and motivation of the user, and the expertise and time available. Let's take the capital budgeting problem as an example to describe different roles. The information user may be a CEO or a planning staff member. What concerns him is the optimal combination for investment under the limitation of available funds and the expected profit of that plan. If he really has a good background in management science, he may want to build a model by himself. Otherwise, the user can ask for a model builder to develop a capital budgeting model. The model builder normally should have good background in both model building and the decision problem. If the model builder finds that integer programming is required to solve this capital budgeting problem and formulates it, then, the next issue will be how this integer programming formulation can be solved? The toolsmith is the one responsible for answering this question. He has to provide the required tools, including algorithm development and program coding. A model builder may develop an algorithm to solve the capital budgeting model and code it so that he plays two roles, but a more reasonable way would be to pick up a ready-to-run computational tool which has been developed by the toolsmith, say a cutting stock algorithm. The toolsmith, therefore, includes those people who develop algorithms and code programs, such as operations researchers, statisticians, or computer programmers.

5. A Framework for Model Management Systems

A model is composed of three components: inputs; outputs; and computational procedures. The computational procedures can be further divided into computational modules (tools) and the logical formulation (logical sequence of tools). For example, a linear programming model includes the input data (Ci, Bi, Aij), the logical formulation, and the computational tools (e.g., LINDO provides the computational tool). Since different users may play different roles and have different responsibilities in using a model management system, their concerns are different. The design of model management systems should take into account these facts and the requirements discussed

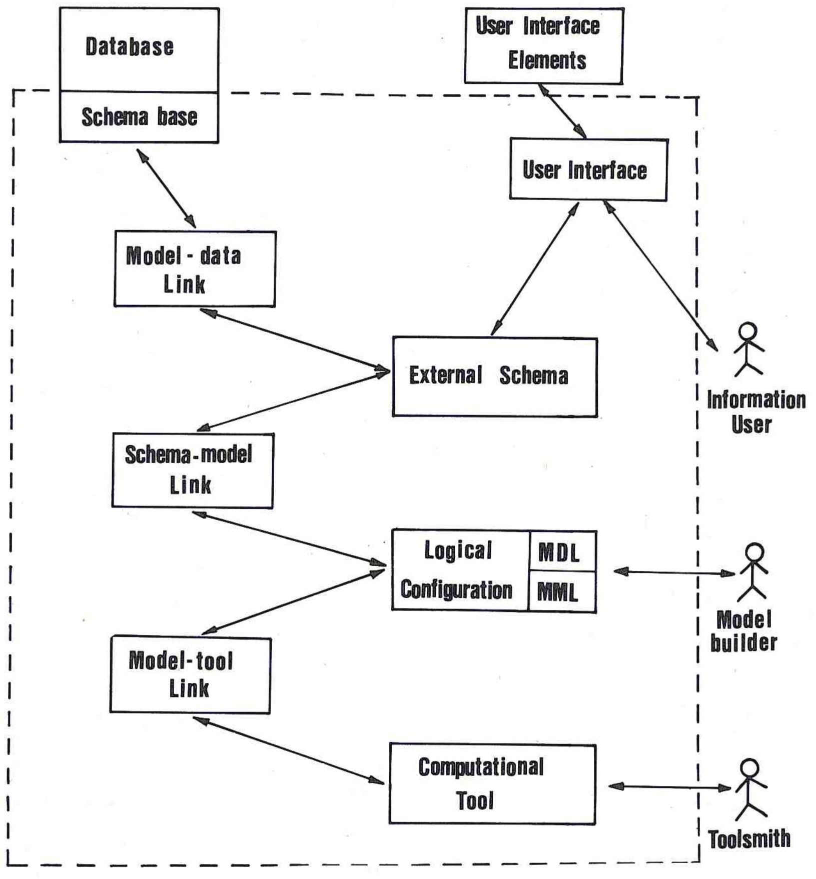

Fig. 3. Framework for Model Management Systems.

before. One way to do this would be to divide the model management system into three different levels based on the correspondence of three components and three roles. The framework for model management, therefore, should include three levels: the external level; the logical level; and the physical computational level. Each level should be highly modular so that it is easy to manage, maintain, update, etc.. Interfaces are required to connect different levels and connect the whole model management system with other components in the DSS. A framework for model management that reflects these requirements has been proposed by Liang [21]. Its graphical representation can be shown as in Fig. 3. From the figure, we can see that three levels are tailored for three different users roles, and different levels can communicate with each other through interfaces, such as the schema–model link, the model–tool link, etc.. The model management system, as a whole, can communicate with the database through the model–data link and with the user through the user interface.

5.1. External Schema

The external schema of a model is the logical representation of the mapping from input to output which reflects the information user's point of view, because what concerns the information user is the output under the limitations of available inputs. A relational schema is the most appropriate approach to represent the external schema of models. Because there are a number of models in a model base, a schema base is required to manage models from the external point of view. Two relations can be identified to represent a model, as follows:

INPUT (model name, input name)

OUTPUT (model name, output name)

For example, suppose a model for computing sales called SALE is SALES = PRICE * QUANTITY. Then the external schema can be represented as:

INPUT		OUTPUT	
model name	input name	model name	output name
SALE	PRICE	SALE	SALES
SALE	QUANTITY		

The advantages of this relational schema are three-fold:

(1) This schema can be stored in a database, and therefore not only simplifies the model retrieval and storage but also simplifies the compatibility between the model base and the database.

(2) It is highly modular and independent of the logical computational sequence of the model. Changes in either the logical computational level or physical computational tools will have limited effect on this level.

(3) The user can get the required output information without even knowing the name of the model. The user can specify the output required and the input available through the user interface; then the system retrieves the model by the input and output names, executes the model, and tells the output value to the user. All current systems require the user to remember the name of the model to be executed. Therefore, if a user wants to use a model which was developed six months ago, the first thing he must do is to find the name of the model. If he cannot find the name, he loses the model. The external schema requires the user to know only what information he wants, which is more reasonable.

To link the external schema with the logical configuration there is a help module in to which the model builder enters the characteristics and explanatory notes on each model in plain language, and this information is made available to the information user whenever an automatic selection is made. This ensures that the user is kept informed not only of his input and output, but also of the basic function of the model.

Moreover, the user is provided with descriptions of certain categories of models, so that he can even select the suitable model manually.

5.2. Logical Configuration

The second level of the framework is a local configuration that reflects the logical relationships among the computational tools used in the model. The logical configuration should be evoked by the external schema and organizes all elements of the model in an appropriate way.

In the external level, models are viewed as data relations and treated as data. However, in the logical level and the physical computational level, a model is viewed as a subroutine and treated as a program written in a very high-level language.

Two languages are required in order to handle a model in this level: MODEL DEFINITION LANGUAGE (MDL) and MODEL MANIPULATION LANGUAGE (MML). MDL defines every component in the model, including model name, input required, output data, tools used, and other models linked. MML deals with the manipulation of models such as create, store, modify, add, delete, link, use, etc.. A model configuration on the logical level can be organized as in Fig. 4.

For those simple models, such as the model 'SALE' described above, the configuration may not be so complex. Simple models may not need to use any tool, nor do they need to link any other models. For example, the logical configuration of the model 'SALE' can be represented as follows:

```
NAME: SALE
OUTPUT: SALES
INPUT: PRICE, QUANTITY
BEGIN
  SALES = PRICE*QUANTITY
END
```

This configuration is not very different from higher-level languages in formulating simple models, but it does provide the capability of formulating complex models and managing them.

From the configuration in Fig. 4, it is clear that any changes of the logical computational sequence or any changes in the tool used or model linked will not affect the external schema accessed by the external information user. On the other hand, the development and addition of new tools to the model management system will not affect the model built unless the name of the tool is changed.

```
NAME: < model name >
OUTPUT: < output,... >
INPUT: < input,... >
TOOL: < optional,... >
MODEL: < optional,... >
BEGIN
  USE < tool >
  IF < condition > THEN
     LINK  < model >
  ELSE  USE < tool2 >
END
```

Fig. 4. Organization of the Logical Configuration.

5.3. Physical Computations

The lowest level in the framework is the physical computational level where a computational tool bank is maintained. Each tool is viewed and treated as a subroutine, and the model builder can use them by a single command, USE. For example, the program that executes the cutting stock algorithm for solving integer programming problems is a tool, as is the computational part of LINDO, but it should not include the problem formulation – the formulation belongs to the logical formulation.

Each tool in the tool base, which is similar to the building block addressed by Sprague and Carlson [2], is an independent module. Tools can not communicate with each other. Rather they connect to an interface which links the logical level and the physical tool level of the framework.

5.2. Interfaces

Interfaces between different levels are very important. Sprague and Carlson identify two interfaces which are important for DSSs – the model–data link and the model–dialog link [2] (p. 273). Bonczek, Holsapple, and Whinston also specified three interfaces in DSSs: the user–model interface; the model–data interface; and the user–data interface [1,20]. As mentioned previously, there are also two important interfaces within the model management systems:

- One is the interface between the external schema and the logical configuration (the schema–model interface);
- The other is the interface between the logical configuration and the computational tool (the model–tool interface).

The functions of the schema–model interface include:

(1) Creating the external schema of the model based on its logical configuration and storing the external schema the schema base;

(2) Executing the logical configuration of the model once the external schema of that model is retrieved from the schema base and executed;

(3) Transferring data and commands between the two levels;

(4) Serving as a buffer between these two levels to provide independence.

The functions of the model–tool interface include:

(1) Executing tools when they are called by a model;

(2) Transferring data and commands between these two levels;

(3) Serving as a buffer between these two levels to provide independence.

5.5. Summary

The framework proposed in this paper includes three levels corresponding to three users' roles. The external level provides a relational external schema to information users who are either decision makers or their representatives. The logical configuration reflects the structure of a model. The model builder, who may be a management scientist with good understanding of the decision problem or a decision maker with good background in modeling, is the one most concerned with the logical level. The toolsmith, who may be an operations researcher who develops new algorithm or a computer programmer who codes problems, deals mainly with the tool level.

In addition to the three levels, four interfaces are required to connect different components in a DSS. They are the model base–database link, the model base–user interface link, the schema–model link, and the model–tool link.

Two modeling languages are required in this model management system: model definition language and model manipulation language. The model definition language allows the specification of models, and the model manipulation language provides an environment for utilizing models. The model builder can use these languages to develop and use models. However, when the decision maker needs information for decision making, they can get it in a way similar to retrieving information from the database, as long as the model has been built. Neither the model name nor the knowledge of how the model works is necessary to get the output.

The advantages of this framework include:

(1) Providing a good environment for model management to all kinds of users, including the information user, the model builder and the toolsmith.

(2) Integrating data management with model management. The information user can retrieve the required information from both the database and the model base by the same command. For example, if the query is 'OUTPUT: SALES OF 1986', the DSS can go through the data dictionary to find whether the sales for 1986 is available. If it is available, the system directly reports the information to the user. If it is not yet available, the system can check whether it can be projected from some models. Fortunately, the model 'SALE' is found. Then, the system goes back to check whether the required inputs are available. If so, the system can retrieve the inputs and execute the model to provide the information. If the input data are also not available, the system can ask the user to provide this information.

(3) Having the power and user friendliness of the relational approach in the external level as well as the flexibilities in modeling and maintenance in the logical and the computational tool levels.

(4) The ability to supply the information user with a short description of a model via a help module.

In the following section, a capital budgeting problem will be presented as an illustrative example to show how this model management system works.

6. An Illustrative Example

Suppose a DSS, with a model management system as proposed in this paper, is available for company X and the CEO is going to use it to support a capital budgeting decision that allocates available funds to some projects in order to maximize the expected return on investment (ROI) of the money invested. If this decision is made every year and the model has been developed and stored in the model base, then the CEO can access the model by specifying the outputs that he wants, such as:

OUTPUT: project combination, total ROI

or:

OUTPUT: project combination, total ROI
GIVEN: # of proj, ROI/proj, total funds, inv/proj

Then the system will retrieve the capital budgeting model for the user so that the user can input the data and receive the output. If some of the input data are available from the data base, then the system, of course, should retrieve those data automatically.

If the model has not been built, the CEO should have the model builder (or himself) develop the model. The model builder recognizes that this a knapsack problem and can be solved by integer programming. If the tool for solving integer programming has been built into the tool base, the model builder can develop the model by using the model definition language and model manipulation language. First, he accesses the logical configuration module. Then, a model as described in Fig. 5 is built. After finishing the logical configuration of the model and storing it, all modeling work is done and the CEO can then use it. If the tool for solving integer programming problems is not yet available, then the model builder has to provide a ready-to-run tool or have a toolsmith (maybe himself) develop a tool and append it to the system toolbase.

If the model builder thinks that the efficiency of the integer programming algorithm is unacceptable when the number of projects is greater than 30 and that a linear programming approximation will be better, then the statement 'USE IP' can be replaced by the statement:

IF i < 30 THEN USE IP
ELSE USE LP

In this example, it is clear that the model includes three levels. The external schema will automatically be generated and stored into the schema base by the schema–model link after the logical configuration of the model has been stored in the system. Therefore, the information user does not need to know how the model is built unless he is really interested in the model formulation. Someday, if a more efficient algorithm for solving integer programming is developed, the model builder will not need to modify any logical configuration to take advantage of the new tool.

```
NAME: capital budgeting
OUTPUT: project combination, total ROI
INPUT: # of proj, ROI/proj, total funds, inv/proj
TOOL:  IP
  BEGIN
    i = # of proj
    C(i) = ROI/proj
    B = total funds
    A(i) = inv/proj
    X(i) = proj i
    MAX  SUM ( i, C(i) * X(i) )
    SUBJ TO  SUM ( i, A(i) * X(i) ) <= B
    USE  IP
  END
```

Fig. 5. Logical Configuration of the Capital Budgeting Problem.

7. Integrating Model Management With Data Management

By implementing the framework described in the previous section, the model base can be completely integrated with the database in a DSS. Although this kind of integration is suggested for implementation at the level of Decision Support System Generators (DSSG), it is also good for complex Specific DSSs.

A simulation of the process of using the integrated system is shown in Fig. 6 and can be briefly described as follows:

(1) The user needs some information for decision making, so he asks the system to provide the information. Suppose the user wants to develop the production plan for 1985. He turns on the system and types: OUTPUT: PRODUCTION PLAN OF 1985

(2) The query processor of the system receives the request and analyzes the query to determine the type of the query: query by input and output; query by the model name; or query by output only. In this case, only the output is specified, so the system will check the database first.

(3) The system checks the database to see whether the output is available in the database. If the production plan of 1985 has been scheduled and stored in the database, then the system will retrieve the data and report them to the user. The selection program may be as follows:

SELECT production plan
FROM database
WHERE year = "1985"

(4) If the data is not available in the database, the system checks the model base to see which model can provide the required output. If no model is available, the system will report the unavailability and provide the modeling phase to the user. If only one model is available, the system can check whether the inputs required for that model are available or not. If all inputs are available, the system will execute the model and report outputs to the user. Otherwise, the system will report the unavailability of some input data and request them from the user.

(5) If more than one model is available to provide the same output, the system will check the inputs required for each model in order to screen some models out. If more than one model remains after the primary screen, the system will provide a menu to the user so that he can choose one. If the information user is given a number of models to decide upon, or if a model is executed, the user is at liberty to request a description of the models via the help module.

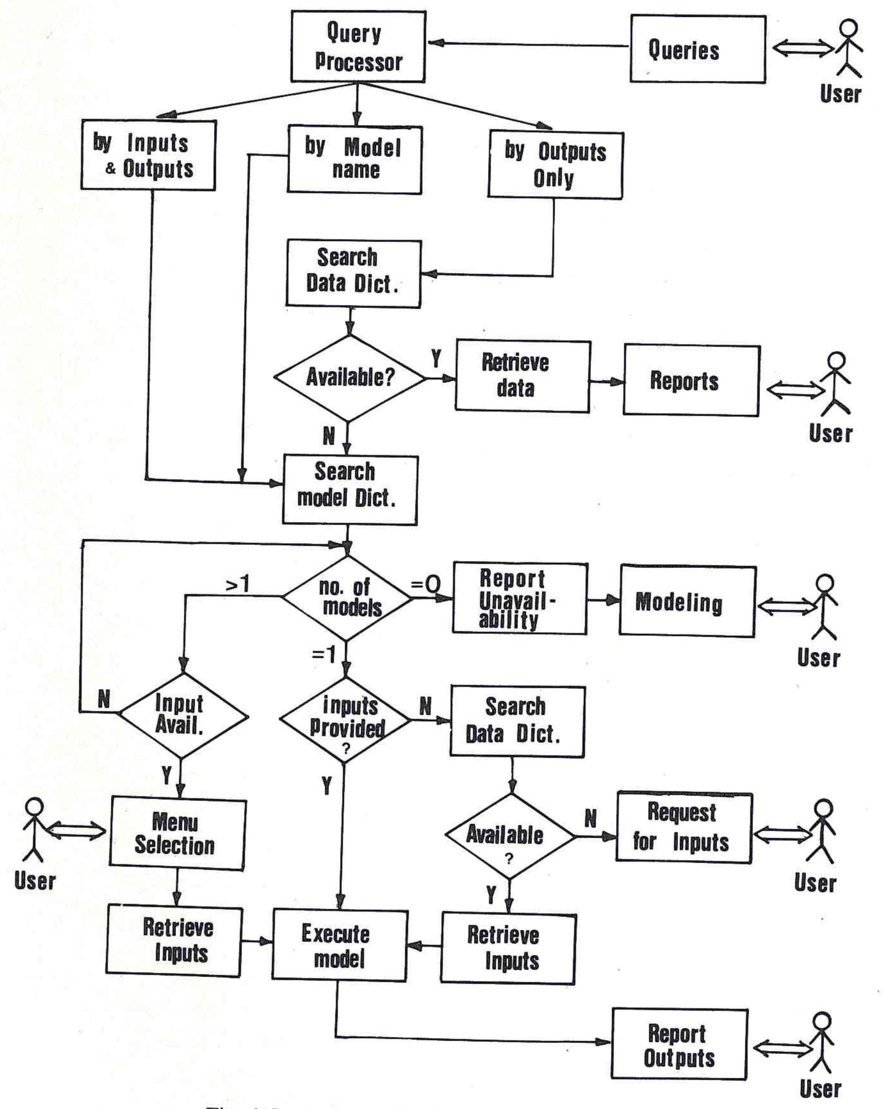

Fig. 6. Implementation of the Integrated System.

8. Conclusions

Because of the growing use of decision models in decision support systems, the management of models is becoming more and more important. A good model management system should have not only those general model management functions such as model creation, model modification and maintenance, and model retrieval, but also capabilities to integrate various components of the system. In this paper, the evolution of the integration between model management and data management is first described. Then, a three-level framework for model management, which not only has the required capabilities but also takes various requirements of different users' roles into account, is proposed. Finally, the possible process of using a system developed with the framework addressed in this paper is described.

Acknowledgement

The author would like to thank Dr. James C. Emery, Dr. Gerald Hurst, Dr. Ari Vepsalainen, Ms. Leslie Aucoin-Drew and an anonymous referee for their valuable comments on an earlier version of the paper.

References

[1] Bonczek, R.H., C.W. Holsapple, and A.B. Whinston, The Evolving Roles of Models in Decision Support Systems, *Decision Sciences* 11 (1980) 337–356.

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

[3] Elam, J.J., *Model Management Systems: An Overview*, Working paper 79-12-04, Decision Sciences, The Wharton School, University of Pennsylvania, PA, p17 (1979).

[4] Elam, J.J., Model Management Systems: A Framework for Development, *Proc. 1980 SE AIDS* (1980) 35–38.

[5] Elam, J.J., J.C. Henderson and L.W. Miller, Model Management Systems: An Approach to Decision Support in Complex Organizations, Proc. First Int. Conf. Information Systems (1980) 98–110.

[6] Blanning, R.R., The Functions of Decision Support System, *Information and Management* 2 (1979) 87–93.

[7] Blanning, R.W., Model Structure and User Interface in Decision Support Systems, *DSS-81 Transactions* (1981) 1–7.

[8] Blanning, R.W., Model-based and Data-based Planning Systems, *OMEGA* (1981) 163–168.

[9] Blanning, R.W., A Relational Framework for Model Management in Decision Support Systems, *DSS-82 Transactions* (1982) 16–28.

[10] Blanning, R.W., Data Management and Model Management: A Relational System, *Proc. ACM 12th SE Regional Conf.* (1982) 139–149.

[11] Blanning, R.W. What is happening in DSS, *Interfaces* (1983) 71–80.

[12] Blanning, R.W., Issues in the Design of Relational Model Management Systems, *IFIPS Conf. Proc.* (1983) 395–401.

[13] Blanning, R.W., Language Design for Relational Model Management *in*: S.K Chang (ed.) *Management and Office Information Systems,* (Plenum, New York, 1984).

[14] Blanning, R.W., Conversing with Management Information Systems in Natural Language, *CACM* 27 (1984) 201–207.

[15] Dolk, D.R., *Model Management in Organization*, Paper presented at ORSA/TIMS Conf. San Francisco (1984).

[16] Konsynski, B. and D. Dolk, Knowledge Abstractions in Model Management, DSS-82 Transactions (1982) 187–202.

[17] Blanning, R.W., *A Relational Framework for the Organization of Model Banks*, Working paper, Owen Graduate School of Management, Vanderbilt University NY.

[18] Minch, R.P., *Design of a DSS Facilitating Model Management and Utilization*, DBA Thesis, Texas Tech. University TX (1982).

[19] Will, H.J., Model Management Systems, *in* Grochla, E. and Szyperski (eds.), *Information Systems and Organization Structure*, pp. 467–482 (Walter the Gruyter, Berlin, 1975).

[20] Sprague, R.H., Jr. and H.J. Watson, "Model Management in MIS", Proceedings of the 7th National AIDS Meeting, 1975, pp. 213–215.

[21] Liang, T.P., *A Multi-level Framework for Model Management in DSS*, Working Paper, Decision Sciences, The Wharton School, University of Pennsylvania, PA (1984).