Object-Oriented Model Management Support System

Park, Sung Joo Kwon, O Byung

Deptartment of Management Science
Korea Advanced Institute of Science and Technology

ABSTRACT

Increasing concerns about model management system lead to studies of user friendliness and model executions. This paper presents model as object's method based on the Object-Oriented Concepts which makes it possible to represent model's operation and enables general decision makers to identify and select more appropriate models. This capability reduces the semantic gap between decision maker and model builder. This view is also able to execute models by way of automatic Ada code generation and specific LP formulation for LINDO. A prototype system is implemented in Pascal.

I. INTRODUCTION

Model Management System (MMS) is an important component of DSS. In the managerial domain, there are a lot of decision makings to solve the various problems, and decision makers develop models fitting and solving their problem spaces, especially in MS/OR domain. Therefore a clear structure for storing and using models is needed which requires appropriate specification about models. The two general definitions in MS/OR domain are as follows :

First, model is a selective representation of the reality.

Second, model is a specification of a set of variables and their relationships designed to represent some real system.

Many MMS researches were based on second definition for it is easier to manage mathematical models than conceptual models. They focused on the representation of models which contributes to develop frames to represent and store the models. However, satisfying the needs about the problem of transforming stored models into executable forms become critical when MMSs are to be used in the real domains.

The conventional model representation frame is model as data and the frame is suitable to the abstraction and storing models. However, it would be restricted to answer the questions of identifying the operations of models and of joining models with solvers naturally. Therefore, a concept of solver base for storing application problems should be needed and managing them will require additional laborous work. Blanning [3] indicated that if model solution technique needs programmimg, then the main issue can be on automatic programming. This implies that if a system which can transform on-executable models in model base into executable models automatically be existed then most problems relating solver base will be resolved. Issues on executable modeling languages are in progress [1,2] and has been employed quite successfully in some domains. However, to build more powerful model management, more steady view of models and its frames are required.

To address the above research issues, Object-Oriented approach is adopted in this paper for the reason that

1. while data in itself represents the abstraction of the concepts, object stresses on the manipulation of concepts [13] . Therefore model as object's method can represent model's characteristics more natural than model as data, and

2. because object-oriented programming languages are well-structured in comparing with another programming languages, automatic code generations to execute models are relatively easier.

Several conceptual object-oriented schemes have been suggested [9,15] in model management, and now having above merits, an extensive Object-Oriented approach in model management might lead to model execution issues [15].

Thus, the main research objective of this paper is to construct a new Object-Oriented modeling environment for ease-of-execute model management system without solver base using Ada code generation technique. We successfully built a prototype, DOMS, to show how to implement the above frames and found that additional profits about high modularity and clear problem-model interface under Object-Oriented concepts have contributions to automated modeling life-cycle.

II. GRAPH BASED MODEL REPRESENTATION

Graph based model representation method is appropriate for understanding models and their relationships. DOMS has two graph-based model representations, Petri Net and Object-Oriented Diagram. The former represents model manager's input/output view and the latter represents user's object-operation view.

2.1 Two distinct views to models

Generally, system users are unfamilier with the model manager's technical jargon and even reject the whole problem solving systems developed by model managers. Many MIS researchers indicated the communication problems between system developers and users. Ein-Dor & Segev [6] indicated that the communication barrier between them will beat the successful system development.

Lucas [11] said that MIS expert (now we identify him as MIS manager) has difficulties in the system user's way of understanding and using informations and capturing what the users really need. DeBrabander [5] indicated that MIS specialist perceives his work in the respect of systematic properties and the users are inclined to think it as operational characters. DeBrabander concluded that the major reason of communication problem between them is based on their semantic gap.

In the model representation issues, we assume that the general MMS user may be able to see his real system space as a set of Objects which have one or more methods. On the other hand, model manager will see real system as a set of fragments which have input, output, and process. Liang [10], Kats & Miller [8], Shaw [14] suggested input/output view for models. Fig. 1 shows different approaches to understand models.

2.2 Model manager's view (Petri Net)

2.2.1 Why Petri Net ?

Petri Net can represent all possible model's input/output informations and solution paths. Unlike AND/OR Graph and another graph based model representation methods, Petri Net can represent model's dynamic view - - which model should be selected and when it to be selected.

2.2.2 Definitions

Applying Petri Net, we define as follows in order to represent models and their execution procedures.

Definition 1. PLACE

A circle node called Place reprsesents input or output of a set of models.

Definition 2. TRANSITION

A bar called transition represents stand-alone model or module which converts a set of input values to one output value.

Definition 3. SOLUTION PATH

A solution path is a finite sequence of places that,

1. all these places are connected,
2. all these places are firable.

Finiteness implies that a solution path is acyclic and is similer to Geoffrion's generic graph in Structured modeling. [7]

An example EOQ model is shown in Fig. 2.

Now how can we get solution paths from the Petri Net ? To answer the question, Reversed Petri Net, Problem Queue, M/D Matrix are produced and defined as follows.

Definition 4. REVERSED PETRI NET

In Petri net, if all edges are converted their directions, the Petri net is reversed.

Definition 5. PROBLEM QUEUE

Problem Queue (PQ) is a set of Places that specifics user's problem and has elements as follows :

$$PQi = \begin{cases} 1 & \text{if i th data should be fired} \\ 0 & \text{otherwise.} \end{cases}$$

$i = 1,\ldots\ldots,m$ m is the number of data

Definition 6. M/D MATRIX

An M/D Matrix, {mij}, which is similar to T-Invariant matrix in Petri Net is a matrix such that,

$$mij = \begin{cases} -1 & \text{if i th data is an output of model j.} \\ 0 & \text{no concerns.} \\ 1 & \text{if i th data is an input of model j.} \end{cases}$$

Fig. 3 is an M/D matrix which has a process informations in Fig. 2. Using the matrix, we can do the following matrix operation to get a solution path.

Bk = A * Xk + Bk-1

A : M/D Matrix

B : Problem Vector (B1 : Problem Queues)

X : Model Vector

k : iteration number (2 <= k <= n , n is the number of models)

Definition 7. ACTIVATED PLACE SET (APS)

APS is a set of sequenced places that is fired by Problem Queues through matrix operations.

$$APSi = \begin{cases} 1 & \text{if i th data has been fired during matrix operation time.} \\ 0 & \text{otherwise.} \end{cases}$$

Or,

$$APSi = \begin{cases} 1 & \text{if} \quad bij + PQi > 0 \\ 0 & \text{if} \quad bij + PQi = 0 \end{cases}$$

$$\text{where } bij = \begin{cases} 1 & \text{if i th data fired in the jth iteration,} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 8. SOLVING PATH MATRIX (SPM)

SPM, {Aij}, is a matrix such that,

aij, i = 1,...,m   m : number of data
      j = 1,...,n   n : total number of

$$a_{ij} = \overset{\text{iterations}}{\begin{cases} 1 & \text{if ith data is fired in the jth iteration.} \\ 0 & \text{otherwise.} \end{cases}}$$

SPM embedds fired model's input/output relationships as follows.

$$a_{i+1,j} - a_{ij} = \begin{cases} -1 & \text{if ith data is required for output in the jth iteration} \\ 0 & \text{no concerns} \\ 1 & \text{if ith data is required for input in the jth iteration} \end{cases}$$

2.3 Model User's view (Object-Oriented Diagram)

Object-oriented diagram, especially Booch[4]'s notations, is one of the appropriate graphic representation method for object-oriented paradigm and the papadigm is meaningful for a better communication with model developer and MMS users. Inventory and finance system can be conceived by the users as shown Fig. 4.

However, there is no message passing in the figure. This implies that the user don't know and even need not know about specific models, and their input and output relations. On the other hand, Object-oriented approach has information-hiding aspect hence, the MMS user need not know the information about his(her) problem. He(She) only need to know problem itself, named Problem Queue which was defined in the previous section.

Also, APS selects necessary objects and operations, and SPM generates message passing. Being an object with its operations selected by APS, then it is called that the object and operations are activated. Because SPM has a information about input/output view, the message passing can have process, that is to say, it has a information when to activate. This overcomes the lack of dynamic view of the traditional object-oriented approach.

In Fig. 4, a box represents object that is considered as the independent part of a real system. And a small box in the objects which calls methods of the object represents model or datum. If the method is model then input and output message passing is performed, else the method is for data retrieval.

2.4 An Integrated Representation for model management (OOD & PN Map)

Thus, two different model representations are introduced. Now, to merge them into a unique diagram, we introduce new graphcal representation called OOD&PN map which integrate OOD(Object-Oriented Diagram) and Petri Net. The relations among them are shown in Fig. 5. As shown in the figure, the map can represent how to join user's problem specifications under Object-Oriented paradigm and solution

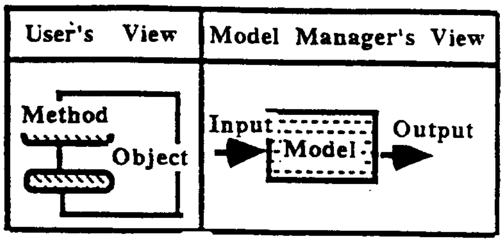


Fig. 1 Different Views for Model

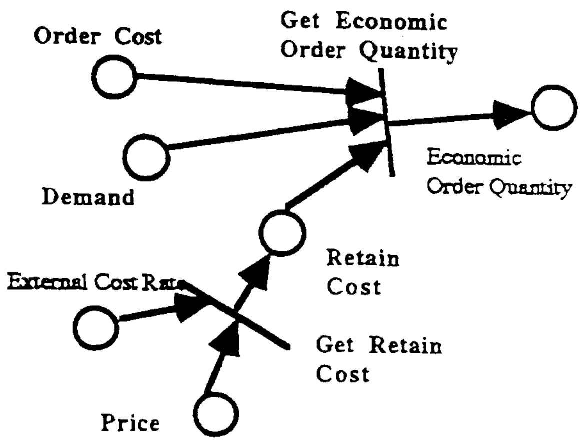


Fig. 2 EOQ Model

	Get Economic Order Quantity	Get Retain Cost
Economic OrderQuantity	-1	
Order Cost	1	
Demand	1	
Retain Cost	1	-1
External Cost Rate		1
Price		1

Fig. 3. M/D Matrix for EOQ Model

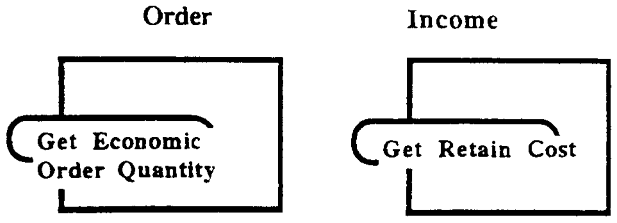


Fig. 4. Object Oriented Diagram for EOQ Model

paths to solve them under input/output features of the models. and then makes it possible to automate Object-Oriented code generations.

III. DOMS SYSTEMS FLOW

3.1 An Overview

Based on two graph representation schemes, DOMS has two subsystems to create, store, select, and execute models named model representation subsystem and model execution subsystem. Model representation subsystem creates models initially, updates thier contents or relationships, and stores them in model base. Model base has model list, M/D Matrix, and model contents.

Model execution subsystem selects an ordered set of modules (that is, solution paths), activate both graphs, and execute by the way of Ada code generation and runs them in the solver environment.

3.2 Model Representation Subsystems flow

A model is represented by its class name, object name, model name, input lists, and model contents. Fig. 6 is a sample representation of the EOQ model.

Assuming that a model can be divided into a set of modules having only one output, the procedures to construct a model base including model integrations are as follows.

```
Step 1. Input model name (M).
Step 2. If M is found in existing data list then go to
     step 4. If found in existing model list, then go to
     3.
     If not found, then go to 5.
Step 3. (update model) If any slots are to be changed,
     then update them.
     If object name is changed and it is not known,
     then go to step 6.
     Otherwise, go to step 7.
Step 4. (change data to model) Specify inputs, their objects
     and contents.
     If the objects are changed and are not known, then
     go to step 6.
     Otherwise, go to step 7.
Step 5. (new model) Specify model object, inputs, their
     object,and contents.
     If the objects are not known, then go to step 6.
     Otherwise, go to step 7.
Step 6. (class input) Input the new object's class name.
Step 7. (M/D matrix updating) Update M/D matrix to fit
     to the changed or aggregated input/output
     informations.
Step 8. If more models are to be specified, then go to
     step 1. Otherwise, Stop.
```

3.3 Model Execution Subsystems flow

Model execution subsystems are based on conventional modeling life cycle.

```
Phase I (Problem Identification)
  Step 1. Describe problem in statement form.
  Step 2. Search model base. If not found, Stop.
     Otherwise, go to step 3.
  Step 3. Make Problem queues (PQ).
Phase II (Solution Path Finding)
  Step 4. Using PQ, do matrix operations until all elements
  that have
     value 1 in Problem Vector (B) are not able to be
     transformed into 0.
  Step 5. If operation stops, then generate solution path
  and SPM.
Phase III (Solving Strategy Construction)
  Step 6. Using SPM, determine APS to retrieve models
  and query data.
  Step 7. Using APS, Identify solvers.
Phase IV (Executable Model Generation)
  Step 8. Generate executable Ada codes using SPM and
  APS.
  Step 9. Generate document for user how to run the
  executable codes.
```

Fig. 7 illustrates the process for model execution.

IV. CODE GENERATIONS

To solve specific problems, DOMS generates Ada code for two reasons. First, Ada is strongly typed object oriented programming language. Second, Ada is very structured, so it can generate codes more easily in comparing with another languages. As shown in Fig. 6, a model consists of model (output) name, object name, class name, input names and their object names, and contents. These component can corresponded to the components in Ada as follows.

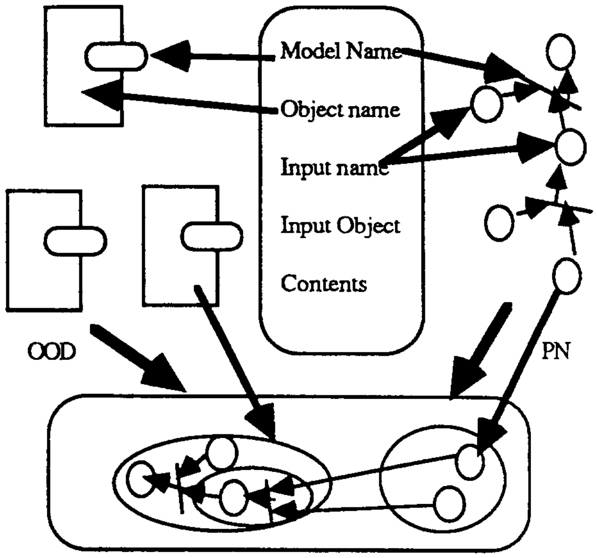
```
OBJECT ======> PACKAGE
MODEL NAME ======> PROCEDURE
INPUT DATA ======> ELEMENTS
CONTENTS ======> BODY PART
```

The syntax is as follows. Capital letters are reserved words in Ada.

```
package_specification ::=
   PACKAGE model's_object_name IS
     { declarative_part}
   END [ model's_object_name ]
declarative_part ::= PROCEDURE  model_query_declaration
                            | data_query_declaration
model_query_declaration ::=
   model_name ( output_data_name : OUT
   subtype_indication {, input_data_name : IN OUT
                                subtype_indication })
```

```
data_query_declaration ::= data_query_name ( output_
                data_name : OUT subtype_indication )
data_query_name := GET_data_name
package_body ::=
   PACKAGE BODY model's_object_name IS
      { model_statement | data_statement }
   END [ model's_object_name ];
model_statement ::=
   PROCEDURE model_query_declaration IS
   use_of_generic_statement
   BEGIN
     { procedure_call }
     model_function ;
   END [ model_name ]
procedure_call ::= [ model_call | data_call ]
model_call ::= input_model_name(output_data_name,
                input_data_name {,input_data_name})
data_call ::= data_query_name(output_data_name)
with_clause ::=
      WITH input_object_name {,input_object_name} ;
input_object_name ::= input_model's_object_name |
                         input_data's_object_name
use_clause ::=
      USE input_object_name {,input_object_name} ;
main_subprogram ::=
   PROCEDURE main_io IS
     use_of_generic_statement
     { input_data_declaration }
     BEGIN
        procedure_call ;
        output_display_statement
     END [ MAIN_IO ];
use_of_generic_statement ::=
       PACKAGE identifier IS generic_instantiation;
data_statement ::=
   PROCEDURE data_query_delaration IS
     use_of_generic_statement
     BEGIN
        data_query_statement;
     END [ data_query_name ];
data_query_statement ::=
   query_statement;
   data_retrieval_statement;
data_retrieval_statement ::=
     identifier.GET (data_query_name)
input_data_delaration ::=
     input_data_name : subtype_indication;
output_display_statement ::=
   display_statment;
   output_value_display_statement;
output_value_diplay_statement ::=
     identifier.PUT (output_data_name)
```

Others are equivalent to Ada syntax.[8] The knowledges about automatic code generations are implemented in Pascal. An example about generated Ada code is shown in Fig. 8.



OOD & PN Map

Fig. 5 OOD, PN, and OOD & PN Map

Model Name (Output Name)
Economic_Order_Quantity

Model Object (Output Object) : Order
Input Name : Retain_Cost Demand
Order_Cost
(Object Name) (I/S_A) (Customer)
(Order)

Functions : Economic_Order_Quantity :=
SQRT(2*Order_Cost*Demand
/ Retain_Cost) ;

Fig. 6 Sample Representation of EOQ Model

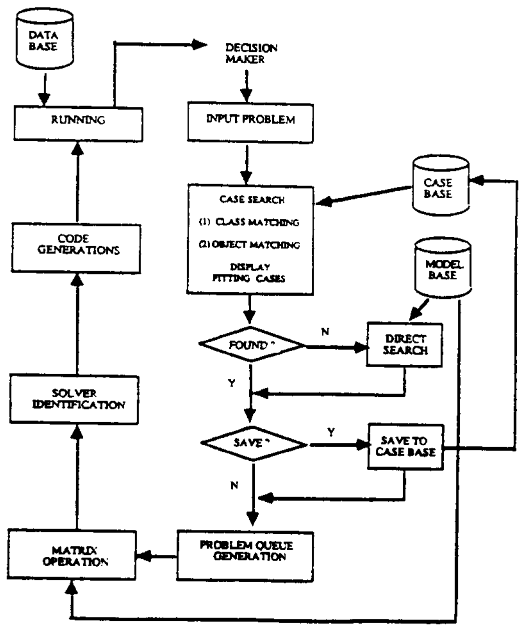


Fig. 7 Model Execution Procedures

V. EXAMPLES

Let's consider the following example as shown in Fig. 9. Company A is selling goods that is delivered from Company B, a subcontractor. So, A and B emphasize inventory management and production planning, respectively. Based on maximizing their sales profits, they want to contract for 3 months. By the way, the customer of B is only A and therefore the inventory policy of A strictly influences on the production planning of B. Managerial rate of the two companies are constant. The decision makers want to solve these problems using appropriate managerial models, however, he(she) don't know which model to use. The example for DOMS's problem solving processes from prblem specification to Ada code generation and specific LP formulation are shown in Fig. 10. In Fig. 10, italics are some data to be specified by users and underlines illustrate internal operations. Generated Ada code and LP formulations are not displayed so as to show man-machine interface in DOMS clearly.

VI. CONCLUSIONS

Despite the researches on MMS have drawn high attentions in the past decade, model execution and user friendly MMS issues are remained to be solved. DOMS tries to provide some insights about these issues as follows.

First, DOMS can manage two different views, input and output view, and object- operation view and then it would remove semantic gaps between decision maker and model manager. In addition to removing the gap, DOMS can handle a dynamic modeling problem which is the weakpoint of the traditional object oriented approach. These trials imply that richer user participation will affect more successful MMS development.

Second, DOMS can execute models automatically by the minimum problem specifications. So, an MMS user need not know which models should be selected and which input will be necessary for executing. Necessary informations are provided in model specification phase, so additional information insertion might not be required.

Third, inheritance and encapsulation are valid in DOMS and it is possible to maintain model base naturally.

The characteristics would contribute to DSS which requires never-ending development and which possesses communication problem as a critical success factor. Also a view of model as object's method would give implications for distributed model management in the organizational level.

[REFERENCES]

1. BHARGAVA, H.K. and KIMBROUGH S.O., "On Embedded Languages for Model Management", *Proc. of the Twenty third Hawaii International Conference on Systems Sciences*, January, 1990.

2. BHARGAVA, H.K. and Krishinan, R., "A Formal

Input your Problem : *Profit Maximization of Company B.*
Case Search:
What is the core class of your problem? : *Company_B*
What is the core object of your problem? : *I/S_of_B*
<u>(Searching Case Base)</u>
In below cases, are there any fitting ones?
(1) Optimal Production Planning
(2) Maximum Total Profit and Their Levels
(3) Get Total Revenue Choose : *2*
O.K. Now I'll recognize your problem as Case (2)
<u>(Problem Queue Generation (PQ)) PQ := <1,0,0,0,0,0,0></u>
<u>(Matrix Operation)</u>

$$\underset{\text{M/D MATRIX}}{\begin{bmatrix} -1 & & & \cdots \\ & -1 & & \cdots \\ 1 & & & \cdots \\ 1 & & 1 & \cdots \\ & 1 & -1 & \cdots \end{bmatrix}} \underset{\text{Xk}}{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}} + \underset{\text{Bk-1}}{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}} = \underset{\text{Bk}}{\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}$$

<u>APS = <1,1,1,1,1,0,0,0,1,0,0></u>
<u>SPM</u> =

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 1 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 & 0 & . \end{bmatrix}$$

Are there any ones to use Special Solvers? (Choose if any)
<u>(Display elements of APS which valuse is 1)</u>
(1) Get_B_Total_Profit (2) Get_Total_Cost.. Choose : *2*
that needs (1) LINDO (2) SPSS Choose : *1*
<u>(Divide SPM according to problem solving strategy)</u>
Please wait. I'm generating Ada Program for you....
<u>(Ada code generation)</u>
End of generation.
Please Compile Ada Code as follows:
ADA RUN2.ADS RUN2.ADB -M MAIN_IO -o out MMSLIB
and you will get RHS value for LP Model
<u>(Compile Ada Program and Run)</u>
About GET_B_TOTAL_COST, Please Answer the questions.
1. TIME: DATA FROM YEAR : *1980* MONTH : *1*
2. TIME: DATA TO YEAR : *1980* MONTH : *3*
3. WORKING MODE
(1) REGULAR (2) OVERTIME (3) RECEIVE
CHOOSE : *1,2*
4. DELIVERY MODE (1) RATIO (2) ADD CHOOSE : *1*
DELIVERY QUANTITY : *32*
<u>(Generate LP Model and Call LINDO)</u>
You get the value of B_TOTAL_COST
Please Compile Ada Code as follows:
ADA RUN1.ADS RUN1.ADB -M MAIN_IO -o out MMSLIB
and you will get B_TOTAL_PROFIT
<u>(Compile Ada program and Run)</u>

Fig. 10 An Example

Approaches for Model Formulation in a Model Management System", *Proc. of the Twentity third Hawaii International Conference on System Sciences*, January, 1990.

3. BLANNING, R.W. "A Framework for Expert Modelbase Systems", *Proc. National Computer Conference*, 1987, pp. 14 - 17.

4. BOOCH, G. Software Engineering with ADA, The BENJAMIN/CUMMINGS PUBLISHING COMPANY, Inc, 1988.

5. DeBrabander, B. , G. Theirs, "Successful Information System Development in Relation to Situational Factors Which Affect Effective Communication between MIS-Users and EDP Specialists", *Management Science*, 1984, pp. 137 - 155.

6. EIN-DOR, et al., Managing Management Information Systems, Lexington, Massachusetts, D.C. Health and Company, 1978.

7. GEOFFRION, "An Introduction to Structured Modeling", *Management Science*, Vol.33, No.5, May, 1987, pp. 547 - 588.

8. KATZAN, H., Invitation to Ada, PETROCELLI BOOKS Inc., 1982.

9. LENARD, M. "An Object-Oriented Approach to Model Management", Proc. Hawaii International Conf. On System Sciences, 1987.

10. LIANG, T.P. "Development of a Knowledge based Model Management System", *Operations Research*, Vol.36, No.6, November-December, 1988, pp. 849 - 863.

11. LUCAS, C. HENRY, "A Descriptive Model of INformation Systems in the Context of the Organization", Data Base, 1973, pp. 27 - 36.

12. MILLER, L.W., N. KATS, "A Model Management System to Support Policy Analysis", *Decision Support Systems*, Vol.2, No.1, March, 1986, pp. 55 - 63.

13. OSCAR, N., "A Survey of Object-Oriented Concepts", Object-Oriented Concepts, Databases, and Applcations, 1989, pp. 3 - 21.

14. SHAW, M.J. et al., "Applying Machine Learning To Model Management In Decision Support Systems", *Decision Support Systems*, Vol.4, No.3, September, 1988, pp. 285 - 305.

15. Suh, E.H., Suh, C.H., and Le Claire, B.P., "Object-Oriented Structured Modeling for an O-O DSS", *Working Paper Series # 89-08, Pohang Institute of Science & Technology*, September, 1989.