An Object-Oriented Decision Support System: A Case of Inventory Management

Houn-Gee Chen
Institute of Technology Management,
National Tsing Hua University

Eldon Li.
Department of Information Management,
Yuan-Ze University

Abstract

This paper addresses an object-oriented methodology in designing decision support systems (DSS). An analysis of computational needs at various levels of an organization defines the data objects of the DSS. Using a two-tier approach, we first devise high-level object constructs in supporting decision analysis. With the message-passing mechanism, a higher-level object (solver) can be composed with certain problem-solving capability. Such composability provides further problem-solving powers. A hierarchical inventory management system and a company-wide order processing system are implemented using our framework.

Keywords: Object-Oriented Design, Decision Support Systems, Inventory Management, Order Processing Management

物件導向決策支援系統之設計：以存貨管理為例

陳鴻基
清華大學科技管理研究所

李有仁
元智大學資管系

摘要

根據物件導向的技術，本研究探討物件導向決策支援系統的設計，應用兩階層物件概念，高階物件支援決策分析，透過訊息傳遞機制，由低階的物件依情境進行資訊的搜尋與求解。二階層的物件效果提供多種求解組合，達到決策支援效果。應用此架構實際解決存貨管理和訂單處理之問題。

關鍵字：物件導向技術、決策支援系統、存貨管理、訂單處理

1. INTRODUCTION

Object-oriented methodology (OOM) which represents entities of the problem domain by object classes has created considerable interest in recent years. Its popularity stems from the fact that the process of the system is invoked by sending messages to the objects, whereas structural details of these objects are kept "hidden" from the user. The approach has been found to enhance the correspondence between problem areas and their semantic representation [32], allowing the user to control the levels of detail and aggregate information during decision-making [1,5,11,25,30,43]. Successful applications in database management [23,26,34], simulation modeling [7,37,34], and artificial intelligence and expert systems [33,41] have been reported. In general, objects have been found to be natural metaphors for model-building in that each is a capsule of state and behavior. They enable the nonprogrammer to solve unstructured problems more effectively by applying their job-specific experience and their own heuristics.

Despite a natural mapping of objects and problem-solving [32], no comprehensive framework has been proposed for developing an object-oriented decision support system (DSS). Instead, DSS researchers have proposed frameworks based on predicate calculus [4,8,13,15], relational approach [3], structured modeling [12,19] and rule-based expert systems [42] for DSS design. Most of them [14,16,21,40,42] have not become the on-line, real-time tools originally envisioned, as managers are found to still rely heavily on staff intermediaries for their computer analyses. The reluctance to use DSSs is partly due to: 1) those DSSs being designed primarily to solve problems on an ad hoc basis while decision support needs, in fact, exist at all levels of management [31,38,40] and across all functional areas [31,40]; 2) a lack of flexible and easy-to-use design to support an effective problem-solving process [14,16,40]. We argue that for the success of DSS, it is essential to support various degrees of data and model *abstraction* by allowing all levels of managers to incorporate his/her intuition, judgement, and/or past experience into the decision-analysis process. Object-orientation could be a significant factor in the success of such a DSS which involves users with diverse computer backgrounds.

Such DSS, however, must map the problem-solving process in a organization using many models and data types at various levels. We note that in the organizational computing, each level (department) performs a set of optimizations and trade-offs to determine the guidelines, boundaries and constraints within which the lower levels (or other departments) must operate. In performing this analysis each department considers only a subset of variables, parameterizing the remaining variables at a level set by other departments. This interdependence of departments for accurate and timely information provides the key motivation for the integrated DSS in this paper. Such a DSS can significantly simplify the job of coordination by incorporating into the system the decision

made by each department. Other departments are then made to operate under the constraints set by these decisions.

The first step in DSS construction is to identify the data sources (data objects) and analyze the interrelationship to represent them in an integrated manner. The hierarcgical nature of the organization leads to a data object hierarchy, proposed in this paper, which is connected by the dual relations of aggregation and inheritance. Each object in this case is formed by an aggregation of the lower-level data objects and the lower-level objects in turn inherit specific information from the aggregated (parent) object. This hierarchy represents a structural view of the information flow within the organization.

Computational models are also defined as objects according to the nature of decision-making at each level. As in object-oriented programming, computational aspects of the models are hidden within model objects and are activated by simple messages enhancing the user's ability to manipulate them during decision-making. We achieve this by using a high level of abstraction of data and models called "deferred classes" [30]. Using this approach, only the required attributes of an object class are defined at each tier of design while deferring the additional attributes for implementation to lower-level classes. Objects that define detailed procedural calculations are delegated to the implementation level and hidden from the user.

Model classification and abstraction allows us to develop "general purpose" model-solvers for a class of models without considering the details of each model in a class. To illustrate our approach, we consider a class of models that use tables and graphs for the interactive solution of problems with the help of the user [2]. In this case, the user interface, as a communication medium, becomes an important component for problem resolution, and the object-oriented design allows us to use a very high-level abstraction of data and models (system-tier objects) that can be used in the interface design. Our DSS building approach starts with the identification of the data and models used in the organization. These data and models are then classified and abstracted to define the system-tier objects which are used in designing the overall "architecture" of the system and in user interface design. Second-tier objects are the implementation-tier objects and, as descendants of the system-tier objects, define the implementation of computational details, which are deferred until specific application objects are considered.

In our design, an object resembles a generic entity in that each is a capsule of conditions and behavior. Objects are activated by message-passing. Sending a message results in a change to the object and/or a triggering of other messages. Working with object classes, a end user can manipulate objects by grouping them in a way that will support problem-solving. A task is then resolved by message execution among objects for information computation and analysis. Problem-solving can be viewed as coordinated

efforts among those objects. An integration of such objects represents a specific function/behavior and defines a problem-solving agent (a new object class). This newly defined object will become a problem solver. As more insight into the problem is gained, new objects will be composed to support additional requirements. This process will continue until the final decision is made. We argue that such composibility and reusability (via generic building blocks) are essential for the success of DSS.

The paper is organized as follows. Section 2 reviews related studies and addresses key features of the object-oriented methodology. Section 3 devises an object-oriented DSS model. Applying such model, section 4 illustrates a simple application--inventory management. A multi-department application (e.g.,order processing system) is also presented in section 5 which is followed by a conclusion section.

2. REVIEW OF RELATED STUDIES

In reviewing recent DSS packages [36,22,18,17], it has been noted [40,42] that a substantial amount of user initiative, such as the preparation of input for the execution of the model as well as the output format for a more interpretable form, is needed to fully and successfully utilize these packages. The lack of *effective* interface support has further caused managers to become reluctant to accept and use DSSs [9,16,21]. A continued development of user-friendly and flexible capabilities has been delineated as a major thrust of DSS research [9,14,29].

With the object-oriented methodology (OOM), a system consists of a set of objects linked in certain way. The definition of objects is based on a collection of instances of abstract knowledge of the problem as well as a classification of relevant entities in an application environment [34]. An object is defined by a set of attributes which characterize the identity of the object and a set of methods representing its behavior. Objects are then accessed through a "defined" interface and details are hidden from other objects. Because of this "protection", objects can be updated independently without affecting others. Objects can be abstracted over a range of dimensions such that only the essential features of objects, relative to the problem domain, are explicitly presented and defined at each abstraction level. An object will inherit properties from higher-level objects.

Objects communicate with each other by sending or receiving messages. These messages invoke methods inside the receiving object and cause changes on the receiving object. Upon the completion of a message, additional messages may be triggered for further actions. This feature provides a way to relate objects for information processing [27,34]. A message may invoke a different response depending on the receiving object. For example, a message to invoke the method "Make_a_decision" would result in different responses from "Family" object (to determine the order cost) and "Item" object (to

determine the order quantity). With this polymorphism, methods can exhibit a similarity in behavior despite a difference in the manner in which this behavior is carried out.

In fact, each object's attribute can denote a state variable. The set of state variables will then define the object's state. Any state change is characterized by a change in the state variables and is caused by message execution. A state change could be simply an updating of an attribute while, in other instances, it could trigger a sequence of messages affecting objects.

3. Object-Oriented Model

In our model, an object X is defined by a tuple <A,M,C,R>. Here, $A=\{A_1,..,A_i,..,A_m\}$ is a set of state variables defining the object's attributes. $M=\{M_1,..,M_i,..,M_n\}$ is a set of methods that will be invoked by message-passing. Allowed states of objects are determined by the constraints (C). These constraints define the feasible state domain as well as prior conditions in executing methods. An object can only be in states consistent with its constraints (laws). An object that is in an unlawful state will change its state to a lawful state by firing appropriate messages. As an example, consider the method Plot of object Graph. The constraint "data is needed prior to plotting a graph" will ensure the filling of data prior to plotting a graph.

There are several ways to associate objects (or object class). Objects can be related together via a simple "parent-child" relationship in one instance while being linked via complex causual effects in others. Three relationships (R) are defined in this model.

1) Generalization-specialization (G-S)

This relationship groups objects on the basis of certain common behavior. The high-level objects define the generalization while each low-level object represents a specialization. The objects in the lower level inherit properties from higher-level objects. Continuously applying this relationship will result in a hierarchy of objects with each level defining specific abstraction of the objects.

2) Assembly

Assembly is the "part-whole" relationship in which objects representing the components are associated with an object representing the final assembly. This relationship will ensure the existence of all components objects prior to assembling the final object. Consider the object *Resource Manager*. The "part-whole" relationship will assure that all the components objects (Machine, Worker, Material) be available before the Resource Manager can proceed the manufacturing of customers' orders.

3) Association

This relationship allows objects to be associated in a new dimension via certain

decision-analysis concerns. Consider objects Graph and Model. An association relationship will enable the Graph object to plot the computational results of the Model object. Note that the plot method of Graph object is invoked by Model object. The assembly relationship can be viewed as a special association relationship. If two objects are tightly bound by a part-whole relationship, an assembly relationship will be defined. However, if two objects are usually considered independently, then the linkage will be an association relationship.

The above relationships resemble three major information flows in an organization. The G-S relationship supports the information flow in mapping the organizational hierarchy. The association relationship allows the information to flow across the departmental boundary. The assembly relationship imposes additional restrictions on the association relationship.

Following the linkages, messages can flow between objects. The general form of the message is: Send Object(B) Method(M). This represents a message to invoke method M on object B (receiving object). With the message, the attributes of the initiating object become accessible to the receiving object. In carrying out the message, a state change may place the affected object in an "unlawful" state and trigger new messages for further state changes. The message processing and triggering mechanism provides a way to model the dynamics of objects. The present study emphasizes the use of *objects* dedicated to represent system entities and *message-passing* dedicated to defining the dynamics of the system in an attempt to support the decision-making. We argues that a decision-maker be able to manipulate various objects (system-tier objects) via such message-passing mechanism. Different levels of analysis will then be carried out in supporting problem-solving.

We now discuss the object-oriented DSS. Our design is based on an analysis of computational needs and data requirements.

4. AN OBJECT-ORIENTED DSS

A typical DSS decision-making [20] always involves a large number of decision variables and constraints which are well beyond the scope of current information technology. The result has been a "divide-and-conquer" approach to problem-solving which operates primarily through a decomposition of the organizational responsibility [19]. These decompositions often result in a hierarchical structure where each level performs a set of optimizations and trade-offs to determine the guidelines, boundaries and constraints within which the lower levels must operate. Each level, in fact, considers only a subset of variables and parameterizes the remaining set of variables at an upper level. This decomposition process may continue until a level becomes clear enough to require no

further breakdown. The subproblems in this level are called problem primitives. The final solution is attained by composing solutions from problem primitives.

In our object-oriented DSS, primitive objects are defined to represent detailed computational procedure and specific data values. They are classified as implementation-tier objects and hidden from the user. However, to facilitate the use of models and data, two system-tier objects are defined so that computational results and data are communicated with the user via DOH (data object hierarchy) and Model, respectively, in a more understandable manner.

4.1 Data Object Hierarchy (DOH)

Under DOH, relevant data objects indicating model variables and parameters are defined at each level to represent various data abstractions of the database. Various data objects are connected by G-S relationships such that upper-level objects represent a generation of data while lower-level objects, in turn, inherit information from their parent objects and denote data specialization. Within this hierarchy, the relationships of data attributes and their roles in decision-making can be clearly defined. For example, a typical DOH in inventory management (Fig.1) indicates that a top-level manager determines the holding and stockout costs within which the middle and low-level managers must operate. Similarly, the middle manager determines the order cost which influences the low-level manager's decision. This hierarchy shows the inheritance relationship between various objects in the DSS, and their role in the organizational decision-making.

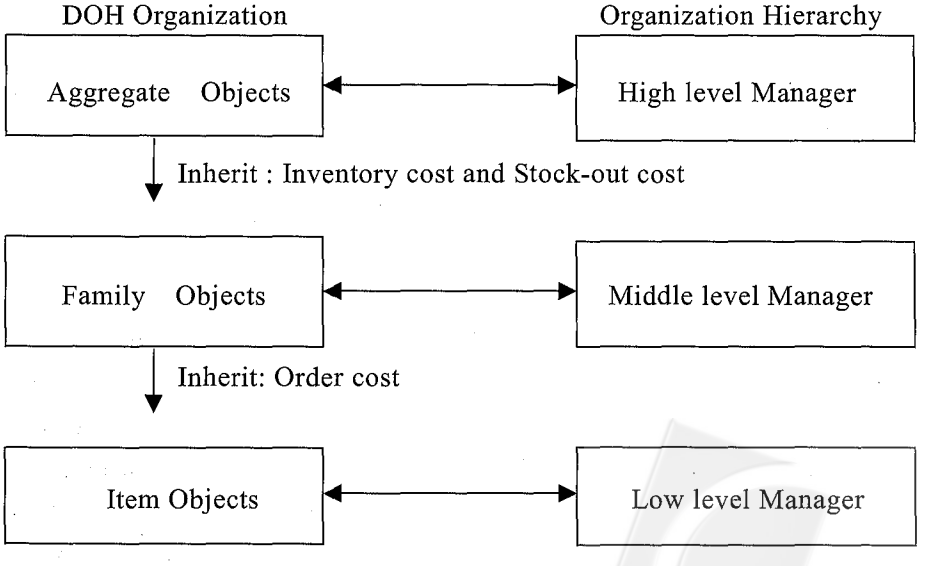


Figure 1: DOH and Organization Hierarchy

4.2 Model Object

Each data object in the previous section can be analyzed by one or more mathematical models in the model base. We assume that the purpose of analysis is to determine the desired value of an attribute, which is specified as the decision variable. The central feature of each model is an output function which allows us to compute the value of a performance index (dependent variable) for a given value of an independent variable. The independent variable is either the decision variable or a surrogate which has a simple mathematical relationship with the decision variable.

Note that computation with a model proceeds when the model is applied to a data object. In this case the parameters of the models are instantiated by the attributes of the data object. These attributes are either directly available within the data object or inheritened from the parent object.

For decision analysis, data objects must be analyzed by model. A model allows users to compute the value of a performance index (dependent variable) for a given value of an independent variable. The independent variable is either the decision variable or a surrogate which has a simple mathematical relationship with the decision variable. To facilitate the use of various computational models, an abstraction (Model) is defined. This system-level Model object (Table 1) is connected with primitive model objects via G-S relationships. Various computational procedures can then be invoked by using the Output method. For example, if Order_Qty is an implementation-level model object, then it can be invoked by sending the message *Output* with the parameter *Order_Qty*. In addition, Model and DOH objects are associated that the data objects provide the parameters for model execution.

Table 1: Model Object

```
+-------------------------------------------------------------------+
| Object: Model                                                     |
| Attributes:                                                       |
|   x   {represents a value of the independent variable}            |
|   a:  a set of parameters                                         |
| Methods:                                                          |
|   Output(x)  {compute the value of the performance index}         |
|   Change(x)  {update the decision variable}                       |
| Constraints:                                                      |
|   c: a range of value {define valid data ranges for x}            |
| Relationships:                                                    |
|   Association: Model_Solver                                       |
|   G-S:   implementation-level model objects                       |
+-------------------------------------------------------------------+
```

4.3 Model_Solver Objects - Worksheet and Graph

The Model object specified above permits the user to compute the value of the performance index for a given value of the decision variable. While this may be useful in examining the impact of a specific decision, it does not allow the user to determine the desirable level of a decision variable for a given parameter set. This can be achieved by developing a solver for the models [2].

We assume that a decision is made by examining the trade-off between the dependent and independent variables rather than for optimization. One way to analyze such a trad-off is to generate a table (Worksheet) that displays the values of dependent variable (y) for selected values of independent variable (x). A graphical plot (Graph) between x and y may also be generated. Interaction between Graph and Worksheet ensures that selected points on the graph are also displayed on the worksheet.

Once a sufficient number of points are generated, a model is "solved" by choosing a value for the independent variable from the table. This determines the new value of the decision variable, which is used to update the data objects.

To illustrate this process, we consider the use of an order quantity model for an inventory intem. After examining several values of order quantity (50, 150, 250, 100, 350), the user may then select an appropriate value of the order quantity which becomes the final decision.

Note that teh above solution procedure separates a model from the solver. The solver in this case uses the appropriate display mechanisms to display the nature of trade-off between dependent and independent variables to the user. It must also incorporate a procedure for selecting from the table a specific value of the independenet variable which represents a solution to them. A decision is made by examining the trade-off between the dependent and independent variables rather than a single optimal solution. Since their features are common to all models in the DSS, they can be abstracted to a high-level object.

To support such analysis, another system-level object Model_Solver is defined. This Model_Solver object consists of objects Graph and Worksheet, allowing the values of dependent variable (Y) for selected values of independent variable (X) to be displayed in a graphic or tabular form, respectively (Table 2). They are connected via an assembly relationship (Figure 2). The constraints set in both objects ensures the existence of arrays X and Y prior to invoking any methods. Additional features are defined to ensure that selected points on the graph can be displayed on the worksheet for detailed comparison; similarly, selected points from the worksheet can be depicted on the graph for better abstraction. Thus, once a sufficient number of points are generated, a model is "solved" by choosing a value for the independent variable from the presentation. This will then determine the new value of the decision variable. To illustrate such a process, we consider the use of an order quantity model. After examining several values of order

quantity (50, 150, 250, 100, 350) (Fig. 3), the user may finally make a decision by selecting an appropriate order quantity (e.g., 150) from the presentation.

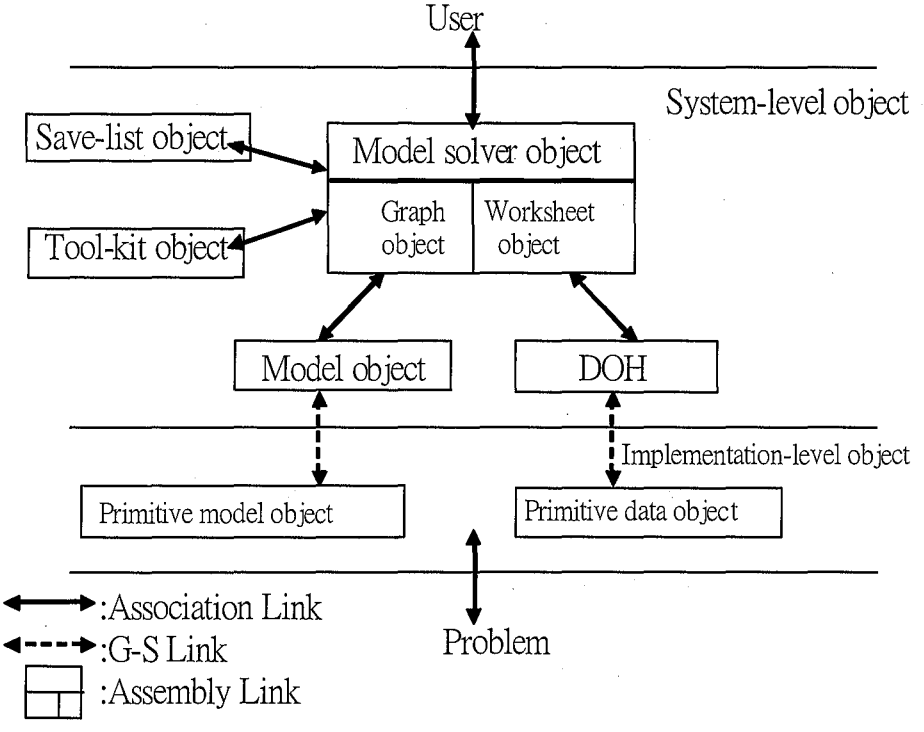
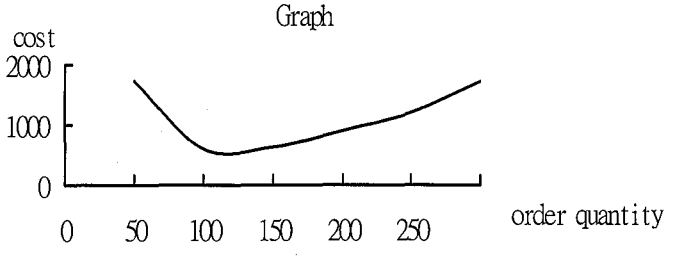


Figure 2: Inventory manager



Worksheet

Order quantity	Cost
50	1038
150	700
250	1030
300	690
350	1720

Figure 3: Graph and worksheet object in inventory manager

Object Graph plots a line graph using arrays X and Y, and labels it with text while object Worksheet displays X and Y in a tabular manner. Once a decision is made (e.g., Make_a_decision), the affected data attributes in the DOH and database will be updated accordingly. With such objects, users can directly manipulate data and models without encountering unnecessary details allowing them to perform their analysis in an interactive manner.

Table 2: Model_Solver Object -- Graph and Worksheet

Model_Solver object
Object : Worksheet
Attributes:
xvar {represents the current value of the independent variable}
X,Y: array{X is the independent variable,Y is the dependent variable}
c: pointer's location in the table
Method:
Display_table {display X,Y in a tabular form}
Add_value(x,y) {add a new value (x,y) to the arrays}
Make_a_decision(xvar){update the data attributes in DOH and database}
Constraints:
If X and Y are empty then invoke Output
Relationship:
Association: Save_List, Tool_Kit
Object : Graph
Attributes:
xvar {represents the current value of the independent variable}
X,Y: array{X is the independent variable,Y is the dependent variable}
Xlabel, Ylabel: string
row, col: current location of cursor
Method:
Plot{plot a graph using values in arrays X and Y,joining each pair of adjacent points by a straight line}
Label {label the axis with Xlabel and Ylabel}
Add_value(x,y) {add a new value (x,y) to the arrays}
Make_a_decision(xvar){update the data attributes in DOH and database}
Constraints:
If X and Y are empty then invoke Output
Relationship:
Association: Save_List, Tool_Kit

4.4 Save_List Object

We have so far considered a single instance of Model being applied to a data object. It is, however, possible to create different application of Model for the same data object by changing parameter values or using a different type of models. Interactive decision-making, in this case, is facilitated by the ability of the decision-maker to change (temporarily) the parameter values of a model and then study the impact of such changes on the graph and worksheet images. The decision-maker should also be able to store and retrieve these

images without having to recreate them every time they are required. This is achieved by saving these images and their parameters in a list (Savel_List) and retrieving them by using commands such as Undo (moving up the list) and Redo (moving down the list). Table 3 summaries object Save_List.

Table 3: Save_List Object

Object : Save_List {store the graph and worksheet image of a model's instance (a trial)}
Attributes:
a: a list of image
Method:
Save {save a trial}
Retrieve {retrieve an existing trial}
Undo {moving up the list}
Redo {moving down the list}
Constraints:
If *a* is empty then display the current image
Relationship:
Association: Model_Solver

4.5 Tool_kit Object

To enhance the data and model manipulation, the Tool_Kit object is defined and incorporated into this DSS design. It includes a set of objects supporting various user interface activities such as file management, editing, and presentation management. These objects are associated with Model_Solver such that their methods can be invoked by data and model objects for additional support. Many object-oriented languages provide such support in their object library. Some of them are defined as follows:

View object {to navigate the Graph and Worksheet for detailed examination}
Edit object {to change, add, delete attributes}
File object {to handle the file input and output}
Window object {to perform windowing techniques}

4.6 Integration of Objects -- A Problem-Solving Agent

Our design allows the user to develop a DSS application without considering the

details of the model's computations. It also supports interactive decision analysis by allowing the user to develop solutions in an incremental manner, from initial experimentation to complete solutions. This approach is particularly appealing where a system's structure is not well defined. Using message-passing and triggering, the model's parameters and decisions can now be updated by working at a high-level abstraction to support decision analysis. The next section addresses an inventory management application.

5. INVENTORY MANAGEMENT APPLICATION

5.1 The Problem

In an ordinary three-level inventory management system [6], the lowest level deals primarily with decisions (e.g., order quantity) regarding single-item inventory management. Although analytical models exist for this type of problem, managerial intervention however is often required due to vendor requirements, transportation constraints or storage restrictions. At the middle level, the manager is instead concerned about policies affecting a group of items (a family). The interaction effects of an individual item's policy are examined and often a trade-off analysis [28, 35] is conducted that exchanges some measure of capital investment in inventory with a family's operating expenses (e.g., order cost). This decision is then passed down and it affects the decisions at lower levels. At the top level, corporate management examines the capital investment required to maintain an inventory level for better customer service. The decision is made by examining the trade-off regarding stockout cost (or holding cost) and the total capital investment [35].

We apply our design framework for this inventory management problem. Our focus is on the decision-making across management hierarchy. Key object definitions are addressed next. More details are reported in [10].

5.2 Data Object Hierarchy

For this inventory management, the uppermost echelon of the DOH (see Fig. 1) defines object Aggregate concerning the aggregate inventory decisions for all stock-keeping items. Important attributes are holding_cost and stockout_cost. At the middle level, object Family is defined. Analysis is performed to determine the order_cost for a group of items. The lowest level deals with the decisions regarding an inventory item and the object Item is defined with attributes such as order_quantity and demand. Although various data abstractions are defined at each level, the inheritance mechanism supports the data integrity. Any analysis at the middle management level is based on the inherited

values of holding cost and stockout cost set at the top level. Similarly, order quantity decisions are affected by the holding cost from the top level and the order cost from the middle level.

A database is derived (Table 4) according to this DOH hierarchy. Each table defines the object's attributes for a particular level. The up index defines the linkage with higher level objects and supports the information inheritance. The down index stores a pointer directed to the first lower-level object at another table. The linking index then locates the next same-group item from the table. For example, using the down index of the family object and the linking index of the item object, we see that family A consists of items 1, 2, and 3 and family B includes items 4 and 5.

Table 4: Organizational DOH (Inventory Management)

Aggregate object

Stock-out cost	Inventory cost	Down index	Up index	linking index
35	0.25	A	N/A	N/A

Family object

ID	Order cost	Down index	Up index	linking index
A	35	1	Aggregate	B
B	10	4	Aggregate	Nill

Item object

Ok	Description	Unit cost	Demand	Lead time	Family ID	Linking index	Down index
1	Screw	15	855	20	A	2	N/A
2	Bolt	40	415	17	A	3	N/A
3	Washer	25	3060	20	A	Nil	N/A
4	Spring	100	1902	23	B	4	N/A
5	Nail	65	1153	23	B	Nil	N/A

5.3 Implementation-level Model Objects

Detailed model formulas are defined by the objects at the implementation level. We assume X is the independent variable (decision variable) and Y is the corresponding dependent variable and consider the basic nature of the trade-off rather than the computation of an optimal value of the decision variable. Among the inventory management models (see [10] for details), the inventory cost model is defined to support the decision regarding a single item. The cycle stock model is applied at the family level. The stockouts trade-off and inventory-cost trade-off models are used at the aggregate level.

5.4 The User Interface

In designing the user interface, we assume that the user analyzes one inventory item at a time. the user, in this case, creates an instance of the Inevntory_Item class specifying the identification number of an inventory item. The data for this item is withdrawn and a model is applied to create graph and worksheet images. The user may keep on generating values in the worksheet or create new images by changing parameter values. Old images are stored in the working memory, and may be retrieved as desried. To facilitate analysis, four windows are created on the screen.

5.5 Implementation

With the aforementioned data and model object definitions, an object-oriented DSS for inventory management was implemented (Fig. 4). Window 1 presents the attributes of a corresponding object. Graph and Worksheet are displayed in Windows 2 and 3, respectively. Window 4 shows a menu for sending messages. "Graph" option allows user to work on the graphic representation while "Worksheet" option directs the user to the tabular representation. With the support of View object (not shown), the user can navigate the graph or worksheet for a detailed analysis. The "Edit" option updates the attributes and parameters by invoking Edit object. "Save_List" is for image saving and retrieval. The "Decision" option records the user's selection on the value of the independent variable and causes updating on data attributes.

"Graph" option allows new values to be generated from the Graph while "Worksheet" option generates new values within the Worksheet. "Decision" allows the user to select a value of the independent variable using the arrow on the Worksheet. The choice "Edit" allows a change of parameters/models for a new trial. It invokes a procedure to compute and display the graph for the new parameters and also display a blank worksheet for the user to work on. Other option are "Undo" or "Redo" as explained earlier. On "Exit", the X, Y arrays and images are saved for subsequent examination. Details are given in [10].

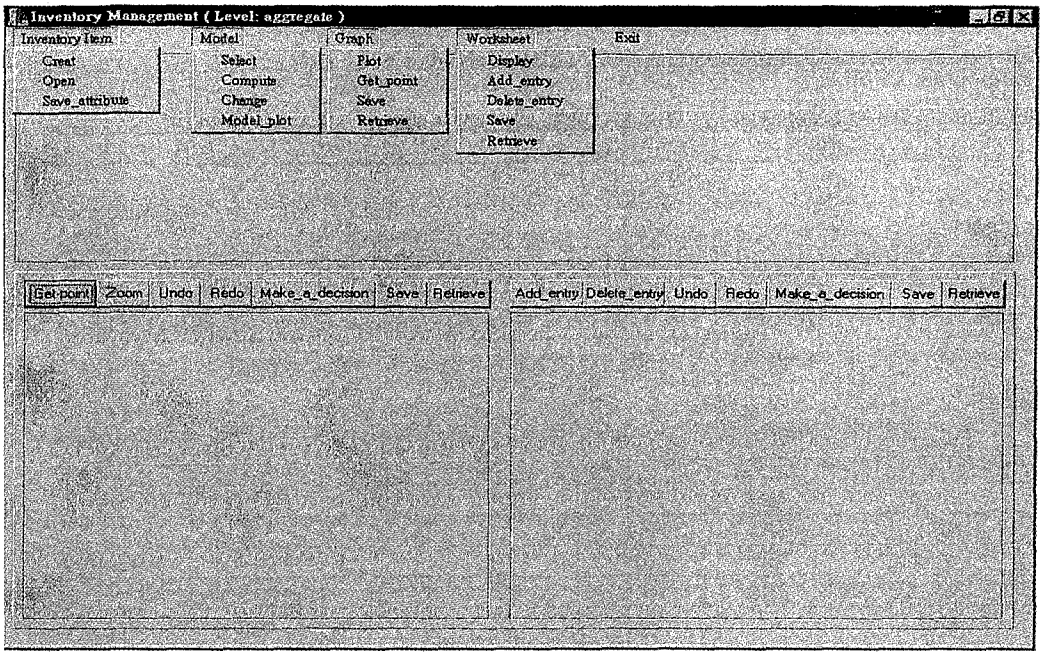


Figure 4: An object-oriented DSS for Inventory Management

In using this system, the user can start at any level of the data hierarchy. According to data objects, a model will be invoked via the Model_Solver object. Its graph and worksheet images are displayed accordingly. Working with the model-solver, the user may create many trials by changing the model's parameters. Old images are stored and can be retrieved as desired. After examining the possible alternatives, a decision is then made and corresponding data attributes are updated. Consider a simple illustration based on Table 4. The objective is to determine the stockout cost, which the user thinks is high. The user starts by selecting a data object and model. A graph and worksheet images of the model's computation are presented and saved as trial number 1. A further analysis sets the holding cost = .15 and the related images are kept as trial number 2. After several conducts, the user decides to reexamine trial number 1. A decision is then made by selecting X=10 from the Worksheet. The stockout cost (6.12) is calculated and updated automatically (Table 5). The process will continue as new requirements occur.

Table 5: Organizational DOH (Inventory Management)--updated

Aggregate object

Stock-out cost	Inventory cost	Down index	Up index	Linking index
6.12	0.25	A	N/A	N/A

Family object

ID	Order cost	Down index	Up index	Linking index
A	35	1	Aggregate	B
B	10	4	Aggregate	Nill

Item object

ID	Description	Unit cost	Demand	Lead time	Family ID	Linking index	Down index
1	Screw	15	855	20	A	2	N/A
2	Bolt	40	415	17	A	3	N/A
3	Washer	25	3060	20	A	Nil	N/A
4	Spring	100	1902	23	B	4	N/A
5	Nail	65	1153	23	B	Nil	N/A

5.6 An Inventory Manager

With our DSS, users can make changes on the parameters and formulate their own problem-solving procedures by sending proper messages in an interactive fashion. The system supports the process by furnishing pertinent information and coordinating the required communications. As a result, various objects are examined and coordinated in an attempt to solve the problem. The problem-solving is carried out via a sequence of message processing. To illustrate this, consider the following inventory changes.

(1) a change on the unit cost of *item i* (C_i) in *family 1*
(2) a demand change for *item j* (D_j) in *family 2*
(3) a creation of *family 3* which includes *items r* and *s*

To make appropriate decisions that will accommodate these changes, the user first updates the data attributes (e.g., unit cost and demand). Next, a new family is created and items r and s are added via Editor object. To determine new policies for those affected items and family, an analysis is then conducted. First, item i's inventory cost is examined by sending a message "Plot" with the updated unit cost. When a selection regarding the order quantity is made, its impact on the family and aggregate level can be examined immediately by sending "Plot" messages to appropriate data objects. After several trials (a sequence of message processing), a final decision is made and the database is updated accordingly. Similarly, new policies for item j and family 3 can be determined in the same manner.

An integration of such objects and their linkages, however, defines a problem solver assisting the user in determining proper inventory policies. Such a solver is defined by a composition of objects and can be viewed as a higher-level object (e.g., manager). Combining this inventory manager and other managers, an even higher-level object can be defined for the support of company-wide decision-making. This is addressed in the next section.

6. AN ORDER PROCESSING APPLICATION

6.1 The Problem

Most organizational functions are not produced by a single department that is acting alone. Instead, each function is manifested by an interaction of several different departments. Consider the order processing function. Orders are received and submitted to the Order_Processing department where the customer's credit is verified and orders are validated for accuracy. If there is stock to fill an approved order, the customer is notified and the shipment is arranged. Otherwise, production is scheduled. If production is overcommitted, "delay" messages will be sent to customers. A new purchase order is issued if raw materials are running short.

This function is realized through the coordinated activities of the order processing, warehousing, manufacturing, purchasing, shipping, and accounting departments. A proper DSS should, however, integrate the operations of disparate departments into a single processing function to support problem-solving.

6.2 A DSS Design

Consider a DSS design (Fig. 5) for order processing using our framework. There are two major managers handling the customer orders. The Order Processor (a manager) handles the order tracking, handling, and shipping. The Manufacturing manager plans and schedules the resources (e.g., machines, workers, materials). Objects are assigned to manage specific production resources. The Machine manager performs the machine scheduling, tracks the machine loading and updates the machine status. Various scheduling algorithms are defined at the implementation level to support the Machine manager. The Material manager reviews inventory policies, conducts the material requirement planning/analysis, and places raw material orders. This is supported by inventory models and MRP models in performing the required analysis. Upon receiving the raw material requests, the Purchasing manager places the orders after proper analysis. The Human_Resource manager schedules workers for production runs. Each manager, in fact, is an object with a specific problem-solving capability. With the message-passing mechanism, they are integrated, enabling them to produce proper responses to the

customers regarding the status of the orders. Details of each manager's function are addressed below and summarized in Tables 6-9.

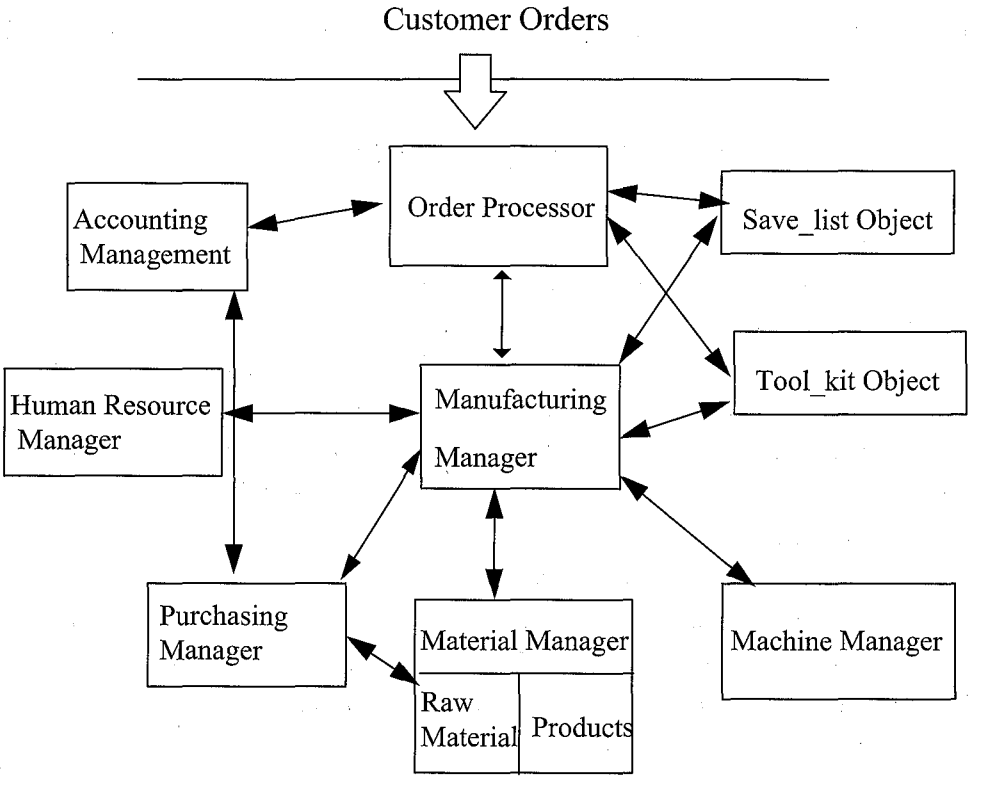


Figure 5: An Order Processing Decision Support System

Order_Processor
Attributes
a: a list of filled orders*
b: a list of committed orders*
c: a list of uncommitted orders*
d: customers' info
Methods
Open_an_account { open an account and check for credit }
trigger: Send Accounting Credit_Check
Fill_an_order { fill an order and arrange shipment }
trigger: Send Material Take_out
trigger: Send Order_Processor Ship
Ship_an order { ship the order and notify the account receivable }
trigger: Send Accounting Bill_customer
Constraints
For each order(x,y,i,d)
if in_stock(x)>=y then
Send Order _Processor Fill_an_order { fill an order with stock }
else
Send Manufacturing Draw_procuction_order
Relationships
Association: Manufacturing, Accounting, Save_List, Tool_Kit
Order* = { (x,y,i,d) |x ε X,y: order quantity; i: received date; d: delivery date }
X: set of products

Accounting Manager
Attributes
a: customer's info
b: payment due
Methods
Bill_customer
Check_Credit
Update_Credit
Draw_a_payment
Relationship
Association: Order Processor

Table 6: Order_Processor and Accounting Managers

Material Manager
Attributes
a: inventory items info
b: in_stock of inventory items
c: review policy of inventory items
d: replenishment level of inventory items
Methods:
Put_on {stock augmentation}
Take_out {stock diminution}
Out_of_stock
trigger: Send Purchasing Place_order
Make_a_decision {determine an appropriate inventory policy}
Create_a_report {create inventory reports}
Constraints:
If in_stock<replenishment level then
Send Material Out_of_stock
Relationships
G-S: Inventory models, material requirement planning models
Assembly: Raw Materials, Products
Association: Manufacturing

Purchasing Manager
Attributes
a: suppliers info
b: a list of filled purchasing orders
c: a list of unfilled purchasing orders
d: a set of evaluating criteria
Methods
Place_order{place an order for materials}
Solicit_a_ bid {solicit bids from supplier}
Make_a_decision {evaluate suppliers' bids and make a decision}
Receive {receive a replenishment, notify the receiving, and arrange the payment}
trigger: Send Material Put_on
trigger: Send Accounting Draw_a_payment
Constraints
Relationship
G-S: bid evaluation models
Association: Material

Table7: Material and Purchasing Managers

Machine Manager
Attributes
a: machine status
b: machine capacity
d: committed schedules
e: uncommitted schedules
f: machine maintenance schedules
Methods:
Draw_machine_schedule {schedule machines for a production order}
Add _machines
Maintain_machines
Constraints:
If maintenance schedule is up then
Send Machine Maintain_machines
Relationship
G-S: machines scheduling models
Association: Manufacturing

Human_Resource Manager
Attributes
a: personal info
b: skill of workforce
c: availability of workforce
d: committed schedules
e: uncommitted schedules
Methods:
Draw_workforce_schedule {schedule workers for production orders}
Add_workforce
Train_workforce
Constraints:
If new workforce is added then
Send Human_Resource Train_workforce
Relationship
G-S: worker scheduling models
Association: Manufacturing

Table8: Machine and Human_Resource Managers

Manufacturing Manager
Attributes
a: finished production orders
b: committed production orders
c: uncommitted production orders
d: capacity of resources
e: status of resources
f: aggregate plan
Methods
Draw_production_order
Schedule_a_production
trigger: Send Material Take_out
trigger: Send Machine Draw_machine_schedule
trigger: Send Human_Resource Draw_workforce_schedule
Finish_an_order
trigger: Send Material Put_on
trigger: Send Order_Processor Fill_a_order
Make_aggregate_plan
Constraints:
If machines, workers, materials are available then
Send Manufacturing Schedule_a_production
If future demand is high then
Send Machine Add_machines
Send Human_Resource Add_workforce
Send Material Make_a_decision {update inventory policy}
Relationship
Association: Order_Processor, Save_List, Tool_Kit, Machine,
Material, Human_Resource
G-S: aggregate planning and scheduling models, forecasting models

Table9: Manufacturing Manager

6.3 Order Processor

Upon receiving customer orders, the Order Processor triggers a sequence of messages: 1) create an account and check the customer's credit rating; 2) validate the order; 3) fill the order with the inventory if there is enough stock; 4) fill the order with additional productions. Message 1 further invokes methods for the Accounting manager to open an account and check the customer's credit. Message 4 will cause the Manufacturing manager to schedule the production. The status of customers' orders is continuously updated in the process. Once an order is filled, the method Make_a_shipment is invoked to arrange a shipment and the Accounting manager is informed.

6.4 Manufacturing Manager

The Manufacturing manager schedules the production and coordinates the resources necessary for supporting the schedule. Customer orders are scheduled based on the availability of resources and the nature of orders. It is associated with three resource managers (e.g., machine, material, human resource) such that various resource scheduling models can be called up to schedule a production run. However, various messages will be triggered to ensure the availability of resources prior to committing a production order. Upon the completion of a production order, the inventory is updated and the Order Processor is informed. If the aggregate plan indicates that the future demand is high, then more resources are requested.

6.5 Machine Manager

The Machine manager monitors the machine loading and is supported by a set of machine scheduling models. A new production schedule is drawn by examining the current production schedule (e.g, the committed schedules) as well as the nature of the order. The Manufacturing manager will be informed of this new schedule. If scheduled maintenance occurs, the machines will become unavailable.

6.6 Material Manager

The Material manager monitors the changes in inventory, reviews inventory policy, directs the reordering of raw material, and provides various inventory status reports. In addition to the aforementioned methods (section 5), methods Put_on and Take_out define the stock augmentation and diminution, respectively. The method Out_of_stock triggers a request to the Purchasing manager for raw material replenishment. Various inventory models are incorporated to support the decision.

6.7 Purchasing Manager

The Purchasing manager handles the activities necessary for acquiring goods and services from suppliers. It identifies the suppliers, solicits the bids, and places an order. The purchasing prices and delivery dates are determined accordingly. Upon receiving the raw material, the material and Accounting managers will be notified appropriately.

6.8 Human_Resource Manager

This manager handles various human resource management activities such as recruiting, staffing, training/development, skills inventory, and worker scheduling. In response to a new production request, it will make a proper worker schedule by invoking scheduling models. The Manufacturing manager will then be informed of this.

6.9 An Order_Processing Manager

With this DSS, the customer orders are first processed by an Order Processor. Upon receiving an order, the method Open_an_account is invoked to validate the order and open an account. An additional message is triggered for credit checking. If an order can't be filled by the in-stock inventory, a production order is created and the Manufacturing manager is notified. Otherwise, the products are taken out of the inventory via a Material manager. Once an order is filled, a shipment is arranged and the Accounting manager is notified. Upon receiving a production order, the Manufacturing manager schedules the production by coordinating various resource managers. The method Schedule_a_production invokes the scheduling of a production. This task is supported by implementation-level scheduling models. The completion of a production order (e.g., Finish_an_order) will notify the Order Processor of a shipment. If the predicted future demands are high, then additional resources are requested. Note that these predictions are further supported by a set of forecasting models. If the material is lower than the replenishment level, the *Out_of_stock* message is triggered and a purchase order is placed. The method Make_a_decision in Material manager helps determine appropriate inventory policies. The Purchasing manager solicits and evaluates bids. The orders are placed according to the outcome of the evaluation. The Material and Accounting managers are notified upon receiving the materials. Machine and Human_Resource managers are supported by machine scheduling and worker scheduling models, respectively. A production is scheduled only when all resources are available. Using message-passing, a group of managers now participates in decision-making and a final decision will be made by exchanging messages between them. Such a DSS composed of a group of independent components defines a unique problem-solving behavior. Note that new constraints can

always be added to each manager in order to define various problem-solving conditions (e.g., scheduling preference).

7. CONCLUSION

We have demonstrated a way to design DSSs for company-wide use. With the two-tier design, the user is directing the system through a high-level construct and the detailed computations are hidden. We analyze the hierarchical nature of organizational decisions and define the data-object-hierarchy data object to support various levels of *data abstraction*. The computational results and data attributes are communicated with users via high-level object constructs. The message-passing provides the mechanism to integrate various objects for further decision analysis.

This new approach is characterized by easily accessible primitive building blocks (e.g., an object), providing a mechanism that permits objects to be linked in order to develop problem solvers. Our design supports the step-wide modeling process such that the decision-maker can create his (her) own abstract type and map the problem domain into those objects, thus building a solver through message passing. Through a composition of objects, various higher-level objects are defined in supporting problem-solving. Once designed and built, a object (e.g., modular) can be reused in order to create another modular for defining a specific problem-solving ability. This composability is essential in supporting various decision-makings.

In contrast to a conventional programming approach, our object-oriented design encourages a decentralized style of decision- making by allowing objects to exchange messages among themselves. Such distribution and localization of knowledge also has its appealing in supporting distributed decision-making.

REFERENCE

1. Alasuvanto, J., E. Eloranta, M. Fuyuki, T. Kida, and I. Inoue, "Object Oriented Programming in Production Management - Two Pilot Systems," International Journal of Production Research, 26 (May 1989) 765-776.
2. Angehrn, Albert A. and Hans-Jakob LHthi, "Intelligent Decision Support Systems: A Visual Interactive Approach," Interfaces, 20:6, November-December 1990, 17-28.
3. Blanning, R.W., "A Relational Framework for Joint Implementation in Model Management Systems," Decision Support Systems, 1, 1985, 69-82.
4. Bonczek, R.H., C.W. Holsapple, "A Generalized Decision Support System Using Predicate Calculus and Network Database Management," Operations Research, 29(2),

1981, 263-281.
5. Booch, Grady, Object-Oriented Design and Applications, Benjamin/Cummings, 1991.
6. Brown, R. G., Material Management System, John Wiley, 1977, New York.
7. Burns, James R. and J. Darrell Morgeson, "An Object-Oriented World-View for Intelligent, Discrete, Next-event Simulation," Management Science, 34 (December 1988) 1425-1441.
8. Chakravarty, Amiya and Diptendu Sinha, "Knowledge Modularization for Adaptive Decision Modeling," ORSA Journal on Computing, 2(4), Fall, 1990, 312-324.
9. Carter, G. M. et al, Building Organizational Decision Support Systems, Academic Press, 1992
10. Chen, H. G. and D. Sinha, "An Inventory Decision Support System Using the Object-Oriented Approach," Computers and Operations Research, 23(2), 1996, pp.153-170
11. Cox, Brad and Andrew Novobilski, Object-Oriented Programming: An Evolutionary Approach, Addison-Wesley, 1991
12. Dijkstra, E.W., "Notes of Structured Programming," in Structured Programming, Dah, O.J., E.W. Dijkstra and C.A.R. Hoare (eds.), Academic Press, 1972.
13. Dutta, A. and A. Basu, "An Artificial Intelligence Approach to Model Management in Decision Support Systems," IEEE Computer 1984, 89-97.
14. Elam, J., J. Henderson, P. G. Keen, and B. Konsynski, "A Vision for Decision Support Systems," Special Report, University of Texas at Austin, 1986
15. Elam, Joyce and Benn Konsynski, "Using Artificial Intelligence Techniques to Enhance the Capabilities of Model Management Systems," Decision Sciences, 18 (3), 1987, Summer, 487-502.
16. Eom, Hyun B. and Sang Lee, "A Survey of Decision Support System Applications," (1971-April 1988), Interfaces, 20:3, May-June 1990, 65-79.
17. EXPRESS, Information Resources Inc., Walthan, MA
18. FOCUS, Information Builders, New York, NY
19. Geoffrion, A.M., "Introduction to Structured Modeling," Management Science, 33, 1987, 547-588.
20. Hax, A. and N. Majluf "Organization Design: A Survey and An Approach," Operations Research 29(3). 1981 May-June.
21. Hogue, Jack, "A Framework for the Examination of Management Involvement in Decision Support Systems," Journal of Management Information Systems, 4 (1), 1987, 96-110
22. IFPS, Comshare Inc., Ann Arbor, MI
23. Kim, Won and F. H. Lochovsky (eds.), Object-Oriented Concepts, Databases, and

Applications, ACM Press (1989), New York.
24. King, David, "Intelligent Support Systems:Art, Augmentation, and Agents," Decision Support Systems: Putting Theory into Practice, 3rd Edition (eds. R. Sprague and H. Watson), Prentice Hall, 1993
25. Lockemann, Peter "Object-Oriented Information Management," Decision Support Systems 5, 1989, 79-102.
26. Lorie, Raymond, W. Kim, D. McNabb, W. Plouffe, and A. Meier, "Supporting Complex Objects in a Relational System for Engineering Databases," Query Processing in Database Systems, 1985, Springer Verlag, 145-155.
27. Martin, James, Object-Oriented Analysis and Design, Prentice Hall, 1992
28. McClain J. O. and L. J. Thomas, Operations Management, 2nd edition, Prentice-Hall (1985), New Jersey.
29. Mitchell, C. M., "Design Strategies for Computer-Based Information Displays in Real-Time Control Systems," Human Factors, 25 (1983) pp. 353-369.
30. Myer, B., Object-oriented Software Construction, Prentice-Hall, 1988.
31. Nunamaker J. F., L. M. Applegate and B. R. Konsynski, "Computer-Aided Deliberation: Model Management and Group Decision Support," Operations Research, (November-December 1988) 849-863.
32. Quillian, R. "Semantic Memory," M. Minsky (editor), Semantic Information Processing, MIT Press, (1970), Cambridge, MA.
33. Ranch-Hindin, Wendy B., Artificial Intelligence in Business, Science, and Industry, Fundamentals, Vol. 1, Prentice-Hall, (1986), New Jersey.
34. Rumbaugh, J. M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, Object-Oriented Modeling and Design, Prentice Hall, 1991
35. Silver, E. A. and R. Peterson, Decision Systems for Inventory Management and Production Planning, 2nd edition, John Wiley (1985), New York.
36. SIMPLAN, Simplan System Inc., Chapel Hill, NC
37. Smalltalk-80, Fundamentals of the Smalltalk-80 Language. (1980).
38. Sprague, R. H. and E. D. Carlson, Building Effective Support Systems, Prentice-Hall, (1982), New Jersey.
39. Sprague, Ralph, "DSS in Context," Decision Support Systems, 3, 1987, pp.197-202
40. Sprague, Ralph and Hugh Watson, Decision Support Systems: Putting Theory into Practice, 3rd ed. Prentice Hall, 1993
41. Stefik, Mark and Daniel Bobrow, "Object-Oriented Programming: Themes and Variations," The AI Magazine, (Winter 1986) 40-62.
42. Turban, E. Decision Support and Expert System, 3rd ed. Macmillan, 1993
43. Yau, Stephen S. and Jeffery J. P. Tsai, "A Survey of Software Design Techniques," IEEE Transactions on Software Engineering, SE-12 (June 1986), 713-721.