COLLABORATIVE MODEL MANAGEMENT IN DEPARTMENTAL COMPUTING[1]

SOON-YOUNG HUH

Graduate School of Management, Korea Advanced Institute of Science and Technology
207-32 Cheongryangri-dong, Dongdaemun-gu, Seoul, Korea
syhuh@green.kaist.ac.kr

Q B. CHUNG

Villanova University, 800 Lancaster Avenue, Villanova, PA 19085, USA
qchung@email.villanova.edu

HYUNG-MIN KIM

Graduate School of Management, Korea Advanced Institute of Science and Technology
207-32 Cheongryangri-dong, Dongdaemun-gu, Seoul, Korea
syhuh@green.kaist.ac.kr

ABSTRACT

Today's increasing connectivity between separate entities within an organization has encouraged us to revisit the old promises of departmental computing and propose a new framework that can expand its potentials. When we view departmental computing from a distributed group decision support systems (DGDSS) perspective, it becomes evident that not only corporate data but also decision models should be an integral part of managerial decision making. In this paper, we present a framework for collaborative model management systems (MMS), which coordinates the changes made to shared decision models in the modelbase so that the net effect of the changes can be systematically propagated across multiple departments by automatically updating the model views. Details of the proposed model change propagation mechanism are discussed along with a realistic departmental computing scenario.

Keywords: Model management systems, Departmental computing, Mathematical models, Object-oriented database management system, View synchronization.

RÉSUMÉ

Le fait que nous découvrons aujourd hui des rapports de plus en plus évidents entre des entités séparées au sein d'une organisation nous incite à remettre en question les vieux poncifs de "departmental computing" et à poser les jalons vers une plus grande ouverture. Si nous considérons le "departmental computing" du point de vue des "distributed group decision support systems (DGDSS)", il devient évident que non seulement les données de l'entreprise mais aussi les modèles de décision devraient faire partie intégrante du processus de décision gestionnaire. Dans cet article nous présentons un nouveau cadre o pourraient fonctionner des "model management systems (MMS)" de collaboration. Ce cadre permettrait une coordination des changements apportés aux modèles de décision partagée du "model base", de sorte que les effets de ces changements seraient systématiquement repercutés sur une multiplicité de départements par une remise à jour automatique des "model views". Cet article offre une analyse détaillée du mécanisme de propagation du changement de modèle que nous proposons, de même qu un scénario réaliste de "departmental computing".

1. INTRODUCTION

Departmental computing refers to the networked computing environment maintained by the end-user departments. By virtue of being decentralized, it is believed to be more effective. Departmental computing began to be understood as an organizational decision

[1]Recd. Sept. '99, Revd. May '00, Acc. June '00. INFOR vol. 38, no. 4, Nov. 2000

support mechanism when Euske and Dolk [1990] reconceptualized it from the perspective of distributed group decision support systems (DGDSS). Similar views have been expressed earlier in a more general context. Gorry and Scott Morton [1971] maintain that the quality of decisions can be enhanced by incorporating data and decision models. Also, through an empirical study, Alter [1977] shows that there are two categories of decision support systems (DSS), namely data-oriented DSS and model-oriented DSS.

Recent adoption of the Internet and intranets in managerial decision support sheds new lights on the applicability of departmental computing by facilitating dynamic collaboration between departments along the value chain of an organization. Those geographically dispersed departments are in need of coordination and cooperation so that they can perform separate but related functions as if they were in the proximity [Martin 1996]. Therefore, it is desirable to develop systems that can support multiple departments to perform collectively. In such systems, flexibility and agility must be warranted because of their dynamic interactions.

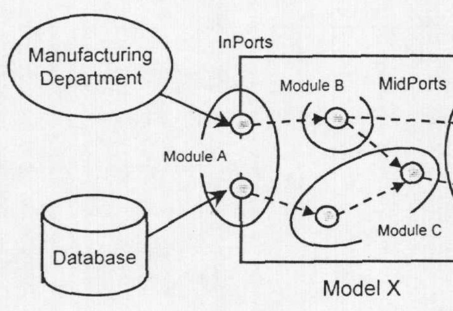
As business problems become more complicated and require high-precision solutions, mathematical models prove to be effective decision tools. Consequently, model management systems (MMS) are increasingly in demand where multiple departments share large-scale mathematical models. Coordination and sharing of decision models takes place on an on-going basis, and shared datasets for common models enhance collective understanding of the business problems. In such an environment, making changes in shared models is inevitable.

In a typical departmental computing model management scenario, corporate models are maintained in a central modelbase server, while individual departmental views are realized in client systems. Therefore, changes made to a shared model can cause inconsistency between the model revised by a department and the views rendered to other departments that have been derived from the original model. This seemingly inconsequential anomaly can do serious harms, such as undermining the validity of the shared understanding of the problem, impaired communications among the participating departments, and ultimately, the questionable efficacy of the departmental computing with distributed MMS. Thus, providing individual departments with synchronized views of the models as well as maintaining consistency becomes a key factor. While a systematic model change propagation mechanism is essential to distributed MMS [Ellis and Gibbs 1989; Huh and Rosenberg 1996], very little attention has been paid to model change and view update and synchronization in a collaborative model management environment.

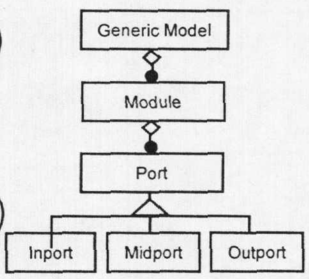
The purpose of this paper is to propose a collaborative model management framework for coordinating model change and automatic view update in departmental computing environments. We employ the generic model concepts [Huh 1993; Huh and Chung 1995] as the underlying framework to uniformly accommodate diverse mathematical models in collaborative MMS. With this framework, multiple model views on the shared models can be readily represented. In order to establish a mechanism of view synchronization, dependency management constructs and processes have been devised. Model change coordination and view update mechanisms are described in detail to synchronize departmental views when changes occur in the shared model. The proposed framework adopts the object-oriented database management system (ODBMS) for combining the structural constructs and change propagation mechanism in a single formalism.

2. REQUIREMENTS OF COLLABORATIVE MODEL MANAGEMENT SYSTEMS

Examination of model management literature reveals that a variety of methods have



(a) Architecture of a Generic Model



(b) Constructs and Relationships

Figure 1: The Basic Structure of Generic Model Concepts

been developed to represent algebraic decision models. Analytical modeling languages such as AMPL [Fourer, et al. 1990], GAMS [Bisschop and Meeraus (1982)], IFPS/-OPTIMUM [Roy, et al. 1986], PAM [Welch 1987], and SML [Geoffrion 1992] are some of the representation schemes most widely used by the practitioners, probably due to the formality of their algebraic notations.

Artificial intelligence (AI) methods have also been extensively used for model representation and reasoning, e.g., first-order predicate calculus [Dutta and Basu 1984], graph [Liang 1986], frames [Chung and O'Keefe 1993; Liang 1993], and analogical reasoning [Mannino, et al. 1990]. Stemming from the data modeling techniques, relational database approaches [Blanning 1985; Dolk 1988] have evolved to define and manipulate models using set-based relations.

With an emphasis on model execution rather than model storage and retrieval, a system-oriented framework [Muhanna and Pick 1994], object-oriented programming language (OOPL) approaches [Le Claire and Sharda 1990; Lenard 1987; Muhanna 1994], and object-oriented database management systems (ODBMS) [Huh 1993] have been introduced.

Given the variety of formalisms for model management, collaborative MMS should accommodate various departmental views on shared models. This is especially important for departmental computing using distributed MMS since an identical model may be understood and represented differently depending on the department's interests, roles, needs, and the user's modeling skills. For instance, while Procurement Department wants to know if the vendors shipments are coming in on schedule, Marketing Department may be interested in the delivery schedule of the customers orders. Likewise, Logistics Department may find it important to stay informed about the current and expected inventory volume, and yet another department, say, Manufacturing, may want to know if the product assembly is on schedule. Individual departments interact with the shared mathematical models on the basis of individual user views by creating, deleting, examining, or modifying the model components and datasets such as product demand parameters, material cost, and inventory levels.

3. MODELBASE CONSTRUCTION USING SYSTEMS APPROACH

Mathematical programming models have hierarchically decomposable structures consisting of multiple algebraic representations including objective function, constraints, parameters, decision variables, constant values, and index sets [Fourer, et al. 1990; Geoffrion 1987; Geoffrion 1992]. In generic model concepts, a model is characterized by

Given	P	a set of products
	$T > 0$	the number of production and sales periods
	$P\_\mathrm{Cost}_p \geq 0$	$p \in P$: production cost per unit of product p
	$S\_\mathrm{Cost}_p \geq 0$	$p \in P$: storage cost per unit of product p
	$P\_\mathrm{Capa}_{p,t1} \geq 0$	$p \in P,\ t1 \in \{1,\ldots,T\}$: production capacity of product p in period $t1$
	$\mathrm{Demand}_{p,t2} \geq 0$	$p \in P,\ t2 \in \{1,\ldots,T\}$: demand of product p in period $t2$
Define	$Q_{p,t1,t2} \geq 0$	$p \in P,\ t1 \in \{1,\ldots,T\},\ t2 \in \{1,\ldots,T\},\ t2 \geq t1$: units of product p manufactured in period $t1$ and sold in period $t2$.
Minimize	$\sum_{p\in P,\ t1\in\{1,\ldots,T\},\ t2\in\{1,\ldots,T\}} (P\_\mathrm{Cost}_p + (t2-t1)S\_\mathrm{Cost}_p)Q_{p,t1,t2}$	$t2 \geq t1$: total cost over all periods for all products considering production cost and storage cost of each product
subject to	$\sum_{t2\in\{1,\ldots,T\}} Q_{p,t1,t2} \leq P\_\mathrm{Capa}_{p,t1}$	$p \in P,\ t1 \in \{1,\ldots,T\},\ t2 \geq t1$: total units of product p manufactured in period $t1$ must not exceed the production capacity of product p in period $t1$
	$\sum_{t1\in\{1,\ldots,T\}} Q_{p,t1,t2} = \mathrm{Demand}_{p,t2}$	$p \in P,\ t2 \in \{1,\ldots,T\},\ t2 \geq t1$: total units of product p sold in period $t2$ must be equal to the demand of product p in period $t2$

Figure 2: Algebraic Formulation of a Production and Sales Scheduling Problem

a **generic model type**. A mathematical model is represented as an abstraction of a problem. A generic model is composed of three types of ports: **inports**, **outports**, and **midports**. Ports have a set of attributes and operations to describe the information pertaining to algebraic expressions and data values. An inport accepts input datasets. An outport produces computational result. A midport carries the rest of the data such as the intermediate computation status or the model constraints. Ports are grouped into modules, which are characterized by the **module type**. Modules can be integrated to form a generic model, which is at the highest level in the generic model hierarchy.

Figure 1(a) shows the external interface of the generic model concepts. Model X has two inports (in Module A) which receive input datasets from Manufacturing Department and the corporate database. Two outports (in Module D) provide model execution results to Marketing Department and Logistics Department. As such, a generic model can represent business decision-making processes encompassing multiple departments, where individual departments collaborate with each other by reciprocating datasets and computation results via shared models.

The constructs of a generic model and their relationships are represented by a hierarchical structure as shown in Figure 1(b) using Object Modeling Technique (OMT) [Rumbaugh, et al. 1991]. Brief descriptions of the OMT notations are as the following: The notation uses boxes and lines to depict classes of objects and their relationships. Among the relationships, the inheritance relationship is denoted by a triangle on the line, and the aggregation relationship is represented by a diamond. Multiplicity of association is represented by a black circle at the endpoint of a line.

To illustrate the proposed framework, let us take a simple and yet fairly realistic business scenario. Shown as Figure 2 is a model of potential production and sales scheduling problems whose objective is to minimize the production and storage cost, while operating within the production capacity and satisfying the market demand.

$$P = \{\text{ski, snowboard}\}$$
$$T = 3$$

Production and storage cost (P_Cost_p, S_Cost_p)

Cost / Product	Production cost	Storage cost
Ski	10	3
Snowboard	8	2

Production capacity ($P\_Capa_{p,t1}$)

Period / Product	1	2	3
Ski	2000	2000	3000
Snowboard	1500	1500	2000

Forecasted demand ($Demand_{p,t2}$)

Period / Product	1	2	3
Ski	1500	1000	3500
Snowboard	1200	800	2500

Figure 3a Example Instance of a Production and Sales Scheduling Problem Defined in Figure 2: Input Datasets with Two Products and Three Periods

This general model can be instantiated when specific data can be provided to describe the particulars. Figure 3 (a) presents a collection of input datasets for a simple case where two products (ski and snowboard) and three periods (1, 2, and 3) are under consideration. Using the given data, a linear programming model can be instantiated as shown in Figure 3 (b).

As shown, the model structure is deliberately and explicitly separated from the datasets so that classifying the ports can become straightforward. The ports associated with the provided datasets become inports, such as product type (P) and period (T). Also, those that are instantiated by parameters become inports, such as production cost (P_Cost), storage cost (S_Cost), production capacity (P_Capa), and demand (Demand). On the other hand, the ports that produce output become outports, such as production and sales quantities in specific period (Q). Finally, the rest of the model components become midports encompassing all the constraints and functions such as production capacity, demand constraints, and overall cost computing function (the objective function). These ports are then aggregated into specific groups such as sets, parameters, variables, objective, and constraints to form modules, which in turn are aggregated into a generic model. In this example, the resulting model is a production and sales scheduling model. Figure 4 represents the structuring of the model as in the form of model components stored in the modelbase.

Minimize

$$10Q_{ski,11} + 13Q_{ski,12} + 16Q_{ski,13} + 10Q_{ski,22} + 13Q_{ski,23} + 10Q_{ski,33} + 8Q_{snow,11} + 10Q_{snow,12} + 12Q_{snow,13} + 8Q_{snow,22} + 10Q_{snow,23} + 8Q_{snow,33}$$

Subject to:

$$\begin{array}{lcr}
Q_{ski,11} + Q_{ski,12} + Q_{ski,13} & \leq & 2000 \\
Q_{ski,22} + Q_{ski,23} & \leq & 2000 \\
Q_{ski,33} & \leq & 3000 \\
Q_{snow,11} + Q_{snow,12} + Q_{snow,13} & \leq & 1500 \\
Q_{snow,22} + Q_{snow,23} & \leq & 1500 \\
Q_{snow,33} & \leq & 2000 \\
Q_{ski,11} & = & 1500 \\
Q_{ski,12} + Q_{ski,22} & = & 1000 \\
Q_{ski,13} + Q_{ski,23} + Q_{ski,33} & = & 3500 \\
Q_{snow,11} & = & 1200 \\
Q_{snow,12} + Q_{snow,22} & = & 700 \\
Q_{snow,13} + Q_{snow,23} + Q_{snow,33} & = & 2500
\end{array}$$

Figure 3b Example Instance of a Production and Sales Scheduling Problem Defined in Figure 2: Linear Programming Model Instantiated by the Input Datasets

As mentioned earlier, the mathematical models shared by different departments can be stored and managed in a corporate modelbase server using generic model concepts with object-oriented database constructs. Once a mathematical model is created and stored in a modelbase, it is accessed and manipulated by individual departments through unique departmental views to reflect the departmental information needs and presentation preferences. Model views can be categorized into two types: model structure views and model instance views. Model structure views pertain to the model structure, as shown in Figure 4, and are usually used by skilled modelers to build a new model or to modify the structure of an existing model. In contrast, model instance views show the data contents of the model ports, including input datasets, computational results, and intermediate computing status of the model. The functional departments such as manufacturing, marketing, and logistics, are most likely to use the model instance views to work with their operational data.

Let us take Figure 5 to illustrate the model structure views. Presented here are two views (algebraic view and genus graph view [Geoffrion 1987]) for the problem described in Figure 2. The algebraic view is literally the model structure view represented by algebraic expressions, while the genus graph view is an acyclic directed graph showing the dependency relations among the genus identified through structured modeling [Geoffrion 1987; Geoffrion 1992]. Additional model structure views such as Block-Schematic representation, Activity-Constraint Graph, Netform Graph [Greenberg and Murphy 1995; Muhanna and Pick 1994] can be used to represent equivalent model structures depending on the user department preference and familiarity.

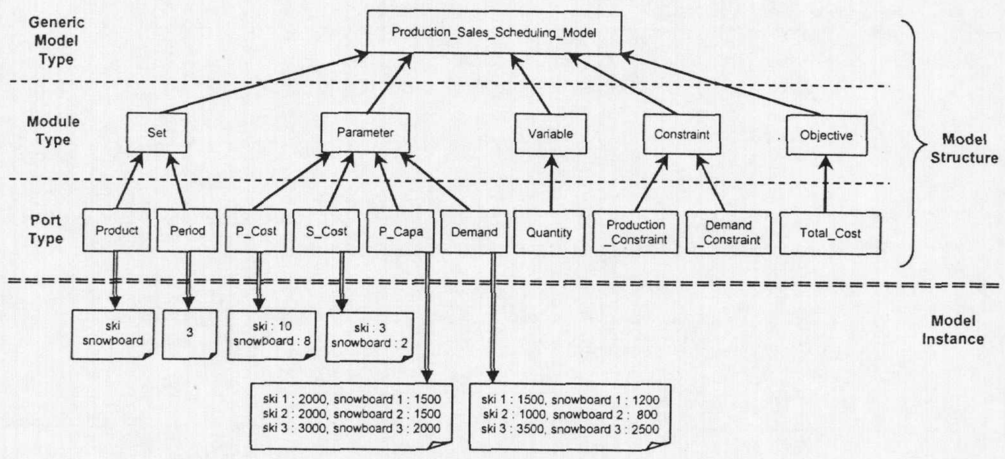
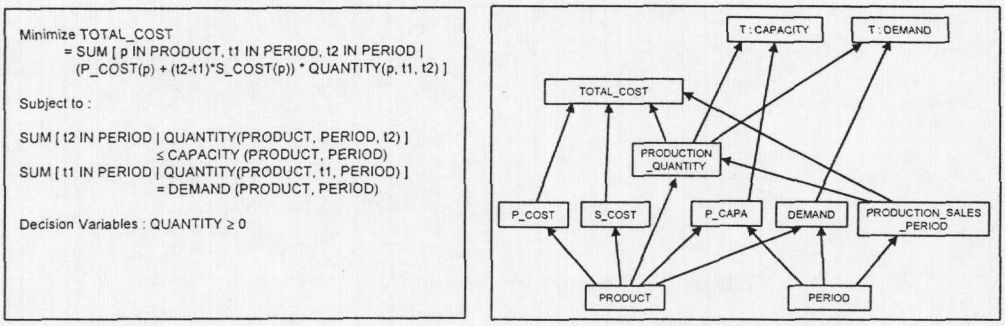


Figure 4. A Generic Model Representation of Production and Sales Scheduling Model


```
Minimize TOTAL_COST
        = SUM [ p IN PRODUCT, t1 IN PERIOD, t2 IN PERIOD |
           (P_COST(p) + (t2-t1)*S_COST(p)) * QUANTITY(p, t1, t2) ]

Subject to :

SUM [ t2 IN PERIOD | QUANTITY(PRODUCT, PERIOD, t2) ]
                        ≤ CAPACITY (PRODUCT, PERIOD)
SUM [ t1 IN PERIOD | QUANTITY(PRODUCT, t1, PERIOD) ]
                        = DEMAND (PRODUCT, PERIOD)

Decision Variables : QUANTITY ≥ 0
```

(a) Algebraic View (b) Genus Graph View

Figure 5: Examples of Model Structure Views of Production and Sales Scheduling Model

4. CHANGES IN THE MODEL AND THEIR EFFECTS ON DEPARTMENTAL VIEWS

A typical mathematical programming model stored in the modelbase undergoes its decision support life cycle. During the intelligence and design phases, it will change continuously since problem identification, model formulation and subsequent refinements may require iterative and incremental model definition [Sprague 1980]. Also, along the institutional use of the model, it may change or improve continually in its structure in response to the dynamic business environment. If the developer of the model in Figure 2 wants to add a new element (transportation cost, for instance), the existing model structure should be changed accordingly as described below.

As shown in Figure 6 (a), a new variable, T_Cost_p, representing the transportation cost for each product, should be added. Also, the objective function should be modified as well to include the transportation cost. When the underlying structure changes, the dependent structure views should be modified to reflect the structural changes. This is depicted in Figure 6 (b). Once the model structure has been developed and stored in a modelbase, a specific dataset may be applied through its model instance views, and the model can be executed using a solver algorithm like the Simplex method.

The departmental instance views may take different forms depending on various

Given ....

$\mathbf{T\_Cost_p \geq 0}$ $p \in P$: transportation cost per unit of product p

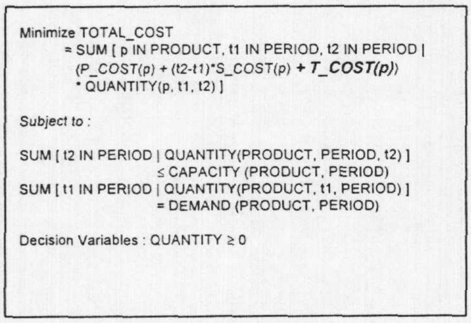
....

Minimize $\sum_{p \in P,\, t1 \in \{1,\ldots T\},\, t2 \in \{1,\ldots,T\}} (P\_Cost_p + (t2 - t1)\, S\_Cost_p + \mathbf{T\_Cost_p})\, X_{p\, t1\, t2}$

$t2 \geq t1$: total cost over all periods for all products considering production cost, storage cost, and **transportation cost** of each product

....

(a) An Example of a Structural Change on the Problem Defined in Figure 2

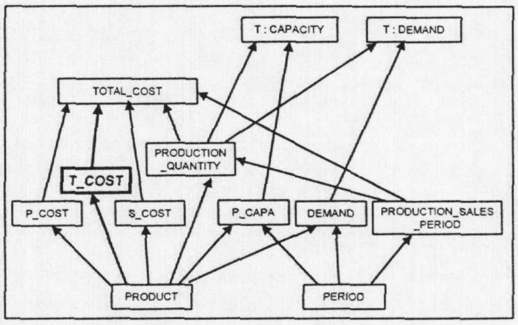


```
Minimize TOTAL_COST
       = SUM [ p IN PRODUCT, t1 IN PERIOD, t2 IN PERIOD |
         (P_COST(p) + (t2-t1)*S_COST(p) + T_COST(p))
         * QUANTITY(p, t1, t2) ]

Subject to :

SUM [ t2 IN PERIOD | QUANTITY(PRODUCT, PERIOD, t2) ]
                    ≤ CAPACITY (PRODUCT, PERIOD)
SUM [ t1 IN PERIOD | QUANTITY(PRODUCT, t1, PERIOD) ]
                    = DEMAND (PRODUCT, PERIOD)

Decision Variables : QUANTITY ≥ 0
```



(b) Reflection of a Model Structural Change on User Views

Figure 6: Structural Change of a Model and its Reflection on User Views

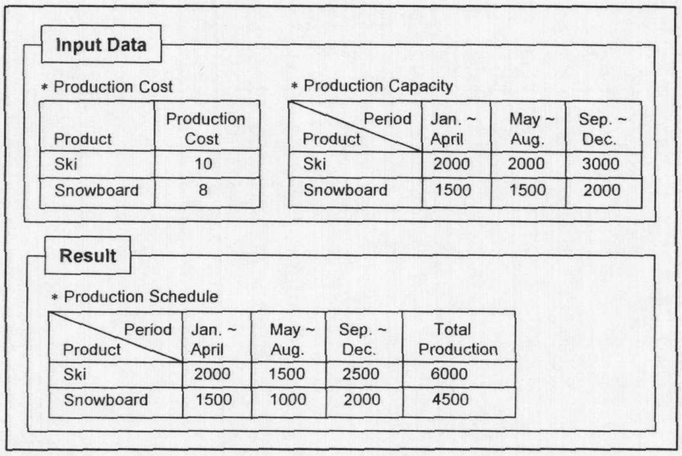
Effects / Locus of Change	Model Views	
	Model Instance View	Model Structure View
At Instance Level	Change	No Change
At Structure Level	Change	Change

Table 1. Effects of model change

factors described earlier. Presented in Figure 7 are two examples of model instance views for the model presented in Figure 2.

Figure 7 (a) provides the model instance view employed Manufacturing Department. The input data containing the production cost and production capacity per product type is inserted through the departmental view, and the production schedule over the production periods per product type is generated from model execution. Figure 7 (b) presents a different model instance view for both Marketing and Logistics departments. Through this view, the market demand forecast and the storage cost are entered, and a product delivery and sales schedule are obtained.

This simple and practical example illustrates how models can be shared and manipulated by multiple operational departments through their departmental instance views. In such collaborative MMS environments, when a portion of a model instance (e.g., input data) is altered by one user department, such a change should be known to other



Input Data

* Production Cost

Product	Production Cost
Ski	10
Snowboard	8

* Production Capacity

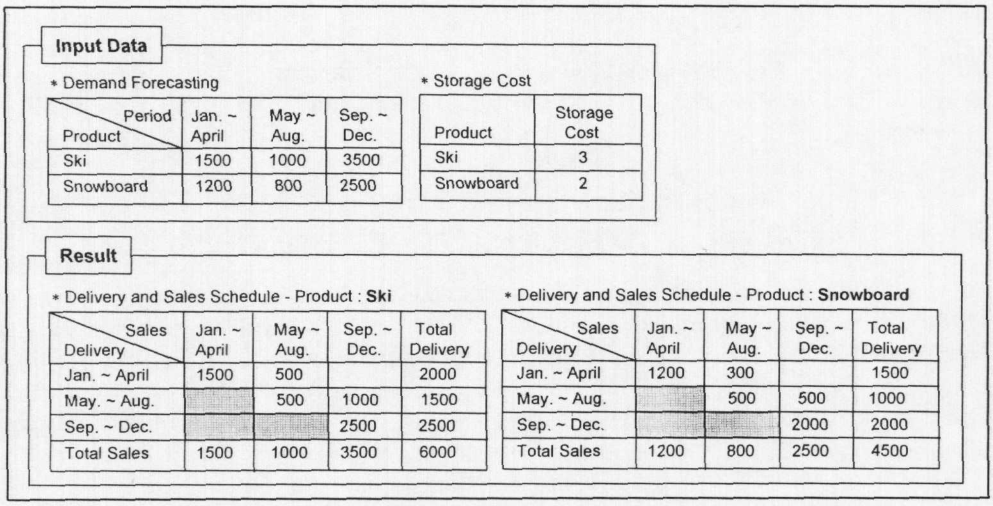
Period Product	Jan. ~ April	May ~ Aug.	Sep. ~ Dec.
Ski	2000	2000	3000
Snowboard	1500	1500	2000

Result

* Production Schedule

Period Product	Jan. ~ April	May ~ Aug.	Sep. ~ Dec.	Total Production
Ski	2000	1500	2500	6000
Snowboard	1500	1000	2000	4500

(a) Manufacturing Department's Model Instance View



Input Data

* Demand Forecasting

Period Product	Jan. ~ April	May ~ Aug.	Sep. ~ Dec.
Ski	1500	1000	3500
Snowboard	1200	800	2500

* Storage Cost

Product	Storage Cost
Ski	3
Snowboard	2

Result

* Delivery and Sales Schedule - Product : **Ski**

Sales Delivery	Jan. ~ April	May ~ Aug.	Sep. ~ Dec.	Total Delivery
Jan. ~ April	1500	500		2000
May. ~ Aug.	▨	500	1000	1500
Sep. ~ Dec.	▨	▨	2500	2500
Total Sales	1500	1000	3500	6000

* Delivery and Sales Schedule - Product : **Snowboard**

Sales Delivery	Jan. ~ April	May ~ Aug.	Sep. ~ Dec.	Total Delivery
Jan. ~ April	1200	300		1500
May. ~ Aug.	▨	500	500	1000
Sep. ~ Dec.	▨	▨	2000	2000
Total Sales	1200	800	2500	4500

(b) Marketing and Logistics Department's Model Instance View

Figure 7: Example of Instance Views of Production and Sales Scheduling Problem

departments that will use the model for their departmental decision making.

Changes in the model influence the model views differently depending on where the changes are made. If the change is made at the instance level, only the model instance views will change. On the other hand, the change made at the structure level will affect both the model structure views and the model instance views. Table 1 summarizes the relationships between the changes in the model and their effects on the model views.

Unlike the model instance changes, in the event of model structure changes, the changed model structure should be first propagated to the various departmental model structure views. Then, the changed model structure with the existing or new model instance should be re-calculated and the revised computation results should be noti-

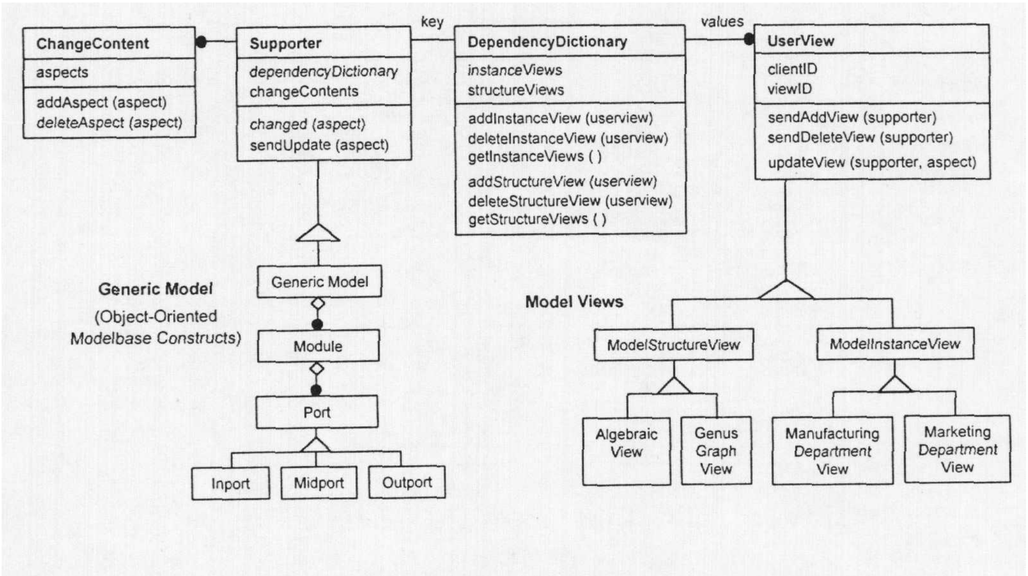


Figure 8: Object Model for Change Propagation in MMS

fied to the dependent model instance views. In the following sections, we propose a change propagation mechanism to support such requirements in departmental MMS environments.

5. OBJECT-ORIENTED DATA MODEL FOR CHANGE PROPAGATION MECHANISM

The mechanism of propagating changes from a shared model to its departmental views can be best represented by the object-oriented paradigm. A **supporter** is an object representing a class of models shared by multiple departments. The model in Figure 4 is an example of a supporter. Another object, **userview**, represents a class of departmental views of the supporter. The model structure views in Figure 5 and their model instance views in Figure 7 are examples of userviews of the supporter in Figure 4.

Changes made to a supporter should be reflected in the userviews, and the mechanism to propagate such changes is shown in Figure 8 using OMT. According to the OMT notations, each class is represented by three rows. The name of the class is in the top row, while the middle and bottom rows contain the class attributes and the operations, respectively. At the core of this mechanism is the **dependency dictionary** object, which defines and maintains the dependency relationships between the supporter and the userviews.

The **DependencyDictionary** class in Figure 8 connects the **Supporter** and **UserView** classes. Since both userviews (instance views and structure views) should be supported in the departmental computing environment, the DependencyDictionary class holds two attributes (the **instanceViews** attribute and the **structureViews** attribute). Defining and enforcing the dependency relationships between the supporter and its userviews is done by three types of manipulation operations: two addition operations (**addInstanceView()**, **addStructureView()**), two deletion operations (**deleteInstanceView()**, **deleteStructureView()**), and two retrieval operations (**getInstanceViews()**, **getStructureViews()**) on the userviews for each supporter.

The dependency relationship between a supporter and a userview is generated when a new userview is created by a department at its local client. More specifically, when a new departmental view is created, the **sendAddView()** operator of the UserView class is invoked to send a message to a supporter informing that a new dependency relationship has been generated. Likewise, when a userview is removed, the **sendDeleteView()** operator is invoked in order to inform the supporter of the removal of the dependency relationship. Since the departmental view is created in a client location, the UserView class needs to have two key attributes (**clientID** and **viewID**) to identify the client, which is the owner of the userview, and the view itself.

Meanwhile, when a change is made to a supporter, the **changed()** operator in the Supporter class is invoked, which triggers the **sendUpdate()** operator to send an update request message to all the registered userviews. Upon receiving the requests, the relevant userviews execute the **updateView()** operator in the UserView class.

As in any multi-user environment, collision of simultaneous attempts to modify the same supporter by more than one department should be avoided. That is, the change operations should be performed within a transaction guard to ensure such features as consistency, permanence, serializability, and recovery [Lamb, et al. 1991]. In what we termed transaction management, the change requests are propagated to userviews only when the transaction reaches a "commit" state, i.e., when the most recent operation on a supporter has been completed successfully. To accomplish this, we use the "transaction" feature of ODBMS. In support of change request, the **ChangeContent** class is additionally provided to register the contents involved in the change. The detailed contents of the change are kept through the **aspects** attribute in the ChangeContent class and the ChangeContent object is linked to the supporter by the **changeContents** attribute in the Supporter class. When the transaction is completed, the ChangeContent objects are either propagated to the userviews or removed in its entirety depending on the final status of the transaction, i.e., commit or abort.

Since the proposed class definitions are generic, the Supporter and UserView classes can be inherited down to generic model and departmental model views. In addition, the operations can be customized to suit collaborative decision making across departments because they are declared in such a way that they can be customized by the participating departments. Figure 8 shows an inheritance hierarchy of the Supporter and UserView classes using OMT. The generic model structure in Figure 1 and its example structure in Figure 4 inherit the properties defined in the Supporter class, while its dependent views (structure and instance) inherit the properties of the UserView class. Both subclasses can be augmented with additional structural attributes and functional operations. Thus, while the superclass handles generic change propagation tasks, the focus of the subclass is on the collaborative model management tasks.

6. AUTOMATIC MODEL VIEW UPDATE IN DEPARTMENTAL COMPUTING

The change propagation mechanism described thus far has set the stage for our next, more detailed discussion. Propagation of model changes using dependency relationships takes place at two distinct levels. At the lower level, an **internal dependency dictionary** maintains the dependency relationships between a supporter and its userviews within a client process. At the upper level, an **external dependency dictionary** manages the dependency relationships between a supporter on the server and its dependent client processes containing the departmental model views. This concept is depicted as Figure 9.

In this diagram, a server containing the modelbase is placed in the middle, while a cluster representing a client system such as Manufacturing Department or Marketing Department is on either side. The shaded elements (e.g., A and B) represent supporters,

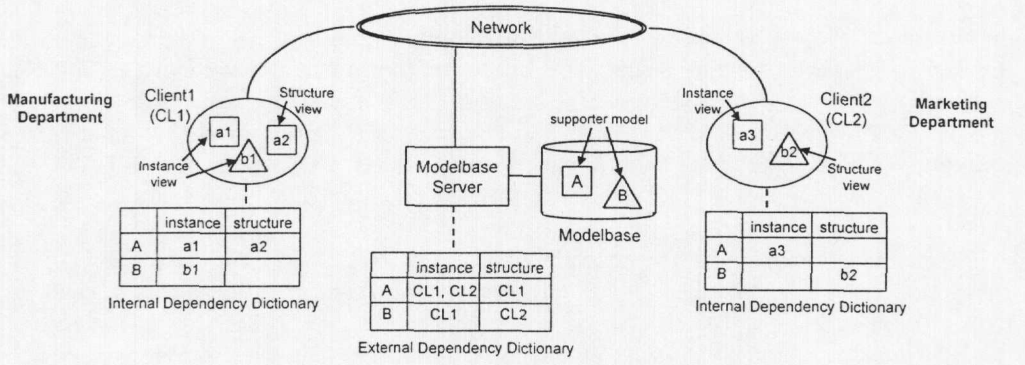


Figure 9: Internal and External Dependency Dictionary

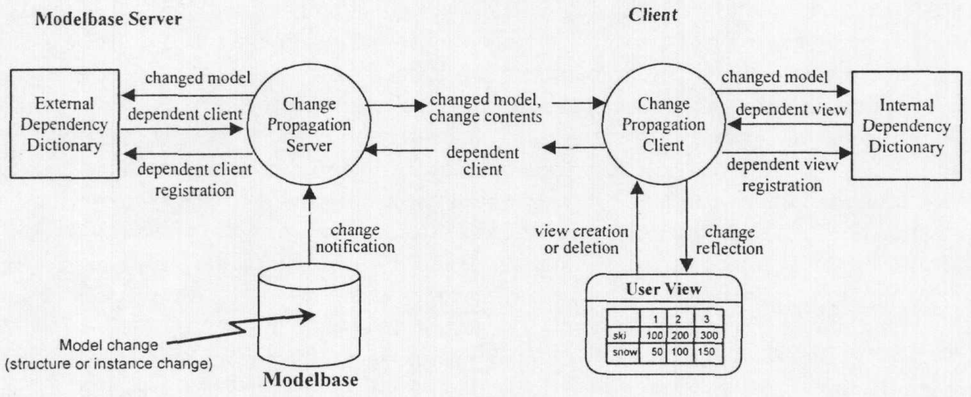

Modelbase Server
Client
External Dependency Dictionary
changed model
dependent client
dependent client registration
Change Propagation Server
changed model, change contents
dependent client
Change Propagation Client
changed model
dependent view
dependent view registration
Internal Dependency Dictionary
change notification
view creation or deletion
change reflection
User View
Model change (structure or instance change)
Modelbase

Figure 10: Change Propagation Server and Change Propagation Client

or the models stored in the modelbase. The transparent elements in an oval (e.g., a1, a3, and b1) represent the userviews created by client departments and stored at client sites (e.g., CL1). For example, the diagram shows that the model instance userviews (a1, a3) for supporter A are created in both CL1 and CL2, whereas a model structure view (a2) for A is created in CL1 only. When the modelbase server consults with the external dependency dictionary, it recognizes that model instance views have been created in clients CL1 and CL2, and a model structure view in CL1. Meanwhile, CL1 has two model views of supporter A, namely a1 (model instance view) and a2 (model structure view). Likewise, CL2 has one model view of A, namely a3 (model instance view). Similar information is maintained internally within each client system.

Maintaining the dependency relationships in two separate dependency dictionaries enables the networked subsystems to function independently so that the network load required to propagate changes can become significantly less than managing the whole dependency information on the server. The benefit is even greater as the number of departmental clients increases since the number of userviews will become substantially larger, subsequent to the growth of the number of clients.

The two objects in Figure 10, **Change Propagation Server** and **Change Propagation Client**, embody this approach to maintain the dependency relationships in two levels. As the names indicate, Change Propagation Server is on the server side while Change Propagation Client is on the client side. Their primary task is to support

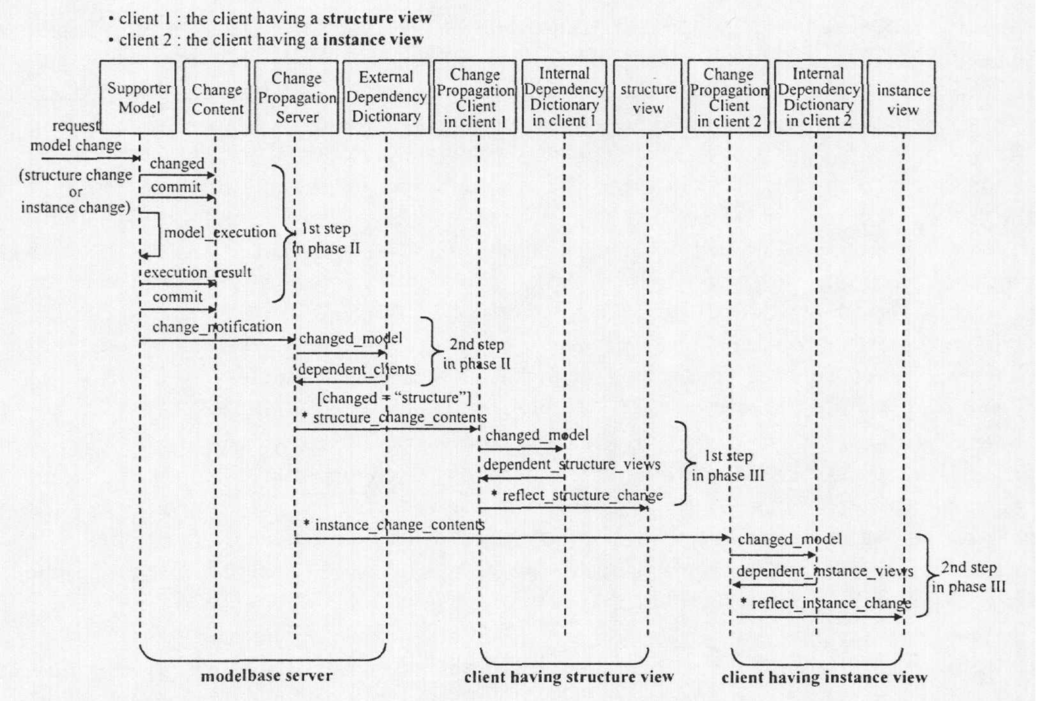

• client 1 : the client having a structure view
• client 2 : the client having a instance view
Supporter Model
Change Content
Change Propagation Server
External Dependency Dictionary
Change Propagation Client in client 1
Internal Dependency Dictionary in client 1
structure view
Change Propagation Client in client 2
Internal Dependency Dictionary in client 2
instance view
request model change (structure change or instance change)
changed
commit
model_execution
execution_result
commit
1st step in phase II
change_notification
changed_model
dependent_clients
2nd step in phase II
[changed = "structure"]
* structure_change_contents
changed_model
dependent_structure_views
* reflect_structure_change
1st step in phase III
* instance_change_contents
changed_model
dependent_instance_views
* reflect_instance_change
2nd step in phase III
modelbase server
client having structure view
client having instance view

Figure 11: A Sequence Diagram on a Change Propagation Process

communications between the server and the clients based on the information contained in each of the dictionaries. As shown in Figure 10, the named arrows represent the messages of the event occurring between the components. Change Propagation Server looks up the external dependency dictionary to find out which clients contain the userviews in various departments, while Change Propagation Client uses its internal dependency dictionary to stay informed about the views created in the client.

Described below is how the Change Propagation Server and the Change Propagation Client interact with each other to play their respective roles when a change is encountered. The process takes place in three phases. During Phase I, the communication link is established between a server and its clients. In addition, the dependency relationships between the supporter model and its dependent userviews as well as the relationship between the supporter and its dependent client processes are registered in the internal dependency dictionary of each client and the external dependency dictionary.

Described here is the change propagation process that takes place during Phase II (at the server) and Phase III (at the client). The **sequence diagram** of UML (Unified Modeling Language) [Booch, et al. 1999] in Figure 11 is helpful in describing the sequence of messages as well as the objects exchanging the messages. In the diagram, an object is shown as a box at the top of a dashed vertical line called the object's lifeline. Each message is drawn as an arrow between the lifelines of two objects, and time progresses downward. To represent a condition behind a message, a special notation is used such as [changed = "structure"]. In that case, the message is passed only if the condition is met. An asterisk is used to denote that a message is sent many times to multiple receiver objects.

Phase II deals with the change propagation process in the modelbase server. First, when the model structure or model instance changes, the content of the change is registered in the ChangeContent object linked to the changed model. Only after the entire transaction has been completed, the next change propagation process can start. Otherwise (i.e., the abort status), the change is not propagated to the departmental userviews. After the update operations are concluded, the model is executed again now with the changed structure or changed instance. The new model execution causes a change at the instance of the supporter model's outports, and the change contents of the outports instances are added to the ChangeContent object as well. In the next step of this phase, after the model change and execution processes have been terminated without any error, the changed supporter sends a message to the Change Propagation Server, which initiates the concrete change propagation steps. Upon receiving the message, the Change Propagation Server checks its external dependency dictionary to identify the clients that contain the userviews of the changed model.

During Phase III, the change propagation process takes place in the client. Different process may be invoked depending on the locus of the change in the supporter model: i.e., structure or instance. In the case of model instance change, the change propagates only to the instance views in various departments. On the other hand, if the change is in the model structure, the propagation process first updates the structure views, and then the revised model execution results are propagated to the instance views.

The first step in Phase III is propagation of the structure change to the Change Propagation Client with the structure views. Upon receiving a structural change message, the Change Propagation Client refers to its internal dependency dictionary to find the specific structure views created in the client. Then, the change contents are reflected on the views appropriately according to the types of the views. The second step involves propagation of the instance change (including the model execution results) to the instance views. If the model's instance change is propagated to the Change Propagation Client in the client containing the instance views, the Change Propagation Client uses its internal dependency dictionary, and the identified instance views are updated adequately to reflect the instance change, completing the whole process.

7. ADVANTAGES OF THE PROPOSED PROPAGATION MECHANISM

The proposed framework for the model change propagation mechanism is uniquely beneficial in several respects. In this section, we present the advantages of the proposed approach compared to other MMS methodologies which tend to neglect the need for synchronization of model changes across different user entities that a modelbase serves.

In this framework, the generic model concepts play the key role, which enables various mathematical models to be represented in a uniform way, by way of an object-oriented database management system (ODBMS) that combines the modeling constructs and change propagation mechanism in a single formalism. The ODBMS approach enriches the change propagation and view update mechanisms with the built-in concurrency handling and transaction management capabilities, ensuring a reliable MMS architecture for collaborative computing environment.

Furthermore, the models stored in the modelbase can be concurrently accessed and manipulated by way of the individual department's userviews. Departmental userviews are further grouped into instance views and structure views. The instance view shows the input datasets and model computation results, while the structure view shows the model schema. Through these views, individual departments may revise their portion of the model using their datasets, execute the model, and even modify the structure of the model.

Next, the proposed method supports the autonomy of user departments modeling activities. Changes in the model instance or in the model structure can be initiated by any user department as necessary. Such changes are immediately and automatically propagated to other department. When a userview is created in a department, a corresponding dependency relationship between the departmental view and the shared model is registered for subsequent management. With the changes at the model instance level, only the dependent model instance views are updated. However, when the changes are at the model structure level, not only the dependent model instance views but also dependent model structure views should be updated.

Finally, by differentiating the superclass objects (responsible for the change propagation mechanism) and the subclass objects (responsible for the model representation in the departmental views), view updates can be handled differently depending on the contents and presentation styles of the dependent departmental views. Thus, the superclass objects perform the core collaborative computing functions including client-server communication, dynamic dependency relationship management between the shared models and their dependent user views, and coordination of the changes. The subclass objects operate on the departmental views so that their model views can be an extension of the shared model to accommodate their own uniqueness and preferences.

8. SUMMARY AND CURRENT STATUS OF THE RESEARCH

In departmental computing, distributed MMS can help multiple departments share decision models to perform their tasks collaboratively while maintaining their own views of the shared models. The structure or instance of a model stored in the modelbase server is expected to change due to the dynamic nature of the business environment. Therefore, it is necessary to provide a mechanism to propagate the changes to the related departments so that their views of the modified model may be brought up to date and consistent. We described a collaborative MMS framework that facilitates the change propagation mechanism and provides synchronized and consistent views of evolving, shared models in departmental computing.

An Implementation of a prototype distributed MMS for collaborative modeling to support multi-view presentation and change coordination with automatic view update has proven the efficacy of the proposed framework. In this Windows NT-based MMS, the object types and the operations proposed in the mechanisms have been fully implemented without any conceptual distortion using C++.

REFERENCES

1. Allter, S. (1977), "A Taxonomy of Decision Support Systems", *Sloan Management Review, 19*, 1, pp. 39-56
2. Bisschop, J. and A. Meeraus (1982), On the Development of a General Algebraic Modeling System in a Strategic Planning Environment, *Mathematical Programming Study, 20*, pp. 1-29
3. Blanning, R. (1985), A Relational Framework for Join Implementation in Model Management, *Decision Support Systems, 1*, pp. 69-82
4. Booch, G., J. Rumbaugh, and I. Jacobson (1999), *The Unified Modeling Language, User Guide*, Addison Wesley
5. Chung, Q. B., and R. M. O'Keefe, (1993), Provisions for Naïve Users of Model Management Systems, *European Journal of Information Systems, 2*, pp. 225-236
6. Dolk, D. (1988), Model Management and Structured Modeling: The Role of an Information Resource Dictionary System, *Communications on ACM, 31*, pp. 704-718

7. Dutta, A. and A. Basu (1984), An artificial intelligence approach to model management in decision support systems, *IEEE Computer, 17*, 9, pp. 89-97

8. Ellis, E. and S. Gibbs (1989), Concurrency Control in Groupware Systems, *Proceedings of the International Conference on the Management of Data, ACM SIGMOD*, pp. 399-407

9. Euske, K. J. and D. R. Dolk, Control Strategies for Group Decision Support Systems: An End-User Computing Model, *European Journal of Operational Research, 46*, 2, pp. 247-259

10. Fourer, R., D. Gay, D., and B. Kernighan, (1990), A Mathematical Programming Language, *Management Science, 36*, 5, pp. 519-554

11. Geoffrion, A. (1987), An Introduction to Structured Modeling, *Management Science, 33*, pp. 547-588

12. Geoffrion, A. (1992), The SML Language for Structured Modeling: Level 1 and 2, *Operations Research, 40*, pp. 38-57

13. Greenberg, H. J. and F. H. Murphy (1995), Views of Mathematical Programming Models and Their Instances, *Decision Support Systems, 13*, pp. 3-34

14. Gorry, G. A. and M. S. Scott Morton (1971), "A Framework for Management Information Systems", *Sloan Management Review, 13*, 1, pp. 55-70

15. Huh, S.-Y. (1993), Modelbase Construction with Object-Oriented Constructs," *Decision Sciences, 24*, 409-434

16. Huh, S.-Y. and Q. B. Chung (1995), A Model Management Framework for Heterogeneous Algebraic Models: Object-oriented Database Management Systems Approach, *Omega, 23*, pp. 235-256

17. Huh, S.-Y. and D. A. Rosenberg (1996), Dependency Maintenance for Collaborative Computing Environment, *Journal of Systems and Software, 34*, 3, pp. 231-246

18. Lamb, C., G. Landis, J. Orenstein, and D. Weinreb (1991), The ObjectStore Database System, *Communications of the ACM, 34*, 10, pp. 50-63

19. Le Claire, B., and R. Sharda (1990), An Object-Oriented Architecture for Decision Support Systems, *Proceedings of the 1990 International Society for Decision Support Systems Conference*, pp. 567-586

20. Lenard, M. (1987), An Object-Oriented Approach to Model Management, *Proceedings of the 20th Hawaii International Conference on System Sciences*, pp. 509-515

21. Liang, T. (1986), A Graph-Based Approach to Model Management, *Proceedings of the 7th International Conference on Information Systems*, pp. 136-151

22. Liang, T. (1993), Analogical Reasoning and Case-based Learning in Model Management Systems, *Decision Support Systems, 10*, pp. 137-160.

Mannino, M., B. Greenberg, and S. Hong (1990), Model Libraries: Knowledge Representation and Reasoning, *ORSA Journal of Computer, 2*, pp. 287-301

23. Martin, J. (1996), *Cybercorp: The New Business Revolution, American Management Association*

24. Muhanna, W. (1994), SYMMS: A model management system that supports model reuse, sharing and integration, *Euopean journal of operational research*, 72,2, pp. 214-242.

25. Muhanna, W., and R. A. Pick (1994), Meta-modeling Concepts and Tools for Model Management: A Systems Approach, *Management Science, 40*, pp. 1093-1123

26. Roy, A., L. Lasdon, and J. Lordman (1986), Extending Planning Languages to Include Optimization Capabilities, *Management Science, 32*, 3, pp. 360-373

27. Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen (1991), *Object-Oriented Modeling and Design*, NJ: Prentice-Hall

28. Sprague, R. H. (1980), A Framework for the Development of DSS, *MIS Quarterly, 4*, **4**, pp. 1-26

29. Welch, J. (1987), PAM A Practitioner's Approach to Modeling, *Management Science, 33*, 5, pp. 610-625

Soon-Young Huh is Associate Professor of Management Information Systems at Graduate School of Management, Korea Advanced Institute of Science and Technology (KAIST), Seoul, Korea. He received a B.S. in electronics engineering from Seoul National University and a M.S. from KAIST. He holds a Ph.D. degree in information systems from University of California, Los Angeles. His current research interests are knowledge extraction using data mining technologies, web pattern analysis and recommendation systems, model management systems, and collaborative computing systems. His publications have appeared in Decision Sciences, Omega, Decision Support Systems, Journal of Database Management, International Journal of Intelligent Systems in Accounting, Finance and Management, Journal of Systems and Software, and others

Q B. Chung is an Assistant Professor of management information systems at Villanova University. He earned his Ph.D. in Management from Rensselaer Polytechnic Institute. The focus of his research is on enhancing the quality of managerial decision making through the use of operations research and management science models, combining the technical and quantitative aspects of problem solving with the awareness of socioeconomic issues surrounding information technology. He papers have appeared in academic journals including *Annals of Operations Research, Intelligent Systems in Accounting, Finance & Management, Omega, European Journal of Information Systems, Information & Management, Electronic Markets, Industrial Management & Data Systems*, and *Journal of Operational Research Society.*

Hyungmin Kim is a senior consultant of the management consulting services at PriceWaterhouseCoopers in Korea. He received his Ph.D. in management engineering from Korea Advanced Institute of Science and Technology (KAIST) in Seoul, Korea. His current research interests focus on collaborative model management systems and knowledge management and workflow systems in financial service area.