December 1995

A Deductive Object-Oriented Approach to Model Management

Vilas Wuwongse  
Asian Institute of Technology

Somsak Klumklai  
Asian Institute of Technology

Follow this and additional works at: http://aisel.aisnet.org/pacis1995

Recommended Citation
http://aisel.aisnet.org/pacis1995/80

This material is brought to you by the Pacific Asia Conference on Information Systems (PACIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in PACIS 1995 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
A Deductive Object-Oriented Approach to Model Management

Vilas Wuwongse and Somsak klumklai-

Computer Science Program, School of Advanced Technologies, Asian Institute of Technology
P.O. Box 2754, Bangkok 10501, Thailand, E-mail: vw@cs.ait.ac.th

Several approaches to the representation of model schema have been proposed in model management. However, none of the current approaches can support all main objectives of model management (redundancy reduction, elimination of inconsistencies, model sharing with multi-users, ease of model use and standard enforcement). A deductive object-oriented framework with inheritance mechanism is proposed for model management. Conversion algorithms from major current model representations to this new framework are presented in order to support all these frameworks.

1 Introduction

Model management is one of the three components of a decision support system (DSS). Models are instruments which transform data into information and thus may aid decision making. Just as data management attends to the systematic treatment of data representing a valuable resource to an organization, model management performs the same role with respect to models.

Model management comprises two important parts: a conceptual structure for their representation, and a set of high level manipulative operations for their access, retrieval and updating within a model base system. The major aims of model management are: Reduction of redundancy of models - they can proliferate in the same manner as application programs in the decentralized data case; Elimination of inconsistencies in model management due to redundancies; Sharing of models with multi-users - model management should admit sharing of implementation of functions for decision models; Encouragement of reuse of existing models in new applications; Ease of use of models by decision makers, and, finally, Enforcement of standards, e.g., standards of specification of decision models, comprehensibility and model interchange between systems.

Various frameworks for model management have been proposed in recent years. Such frameworks have been based on the relational database theory [3,5], on the entity-relationship approach [4,6], on the model abstraction method [7,12], on the structured modeling language [10,19], on io-mapping [15,16], and on the object-oriented approach [13,18,22]. A model has different views in these frameworks. Each representation in these frameworks can capture well characteristics of decision models and support solutions of the problems of model management. However, none of these frameworks can support all of the main objectives of model management (redundancy reduction, elimination of inconsistencies, model sharing with multi-users, ease of model use and standard enforcement). A new model framework is needed to meet all these needs.

This paper proposed a new model representation by means of a deductive object-oriented framework. Conversion Algorithms of model representation have been proposed in this framework. Within this framework, a model is conceived to be a class. With an inheritance mechanism, a new model can be built by making use of existing models which thus act as a superclass. As another advantage, such a deductive mechanism permits synthesis of new models from existing ones and provides to represent a model integration. Based on this framework, certain useful inheritance rules can be derived which assist in the reduction of redundancy in the design of a centralized model management system. Such deductive rules can improve the compactness of model management systems. The representations of models in this approach also improve a user's understanding of a model's potential. The conversion algorithms can be employed to integrate the current model frameworks into an unique framework.

In the next section, a brief overview of major components of the deductive object-oriented paradigm are presented. Then a new framework to
model representation has been proposed in the section 3. The components of model representation will be defined precisely within this framework. The inheritance rules are also devised to specify relationships between models in the model base. In section 4, conversion algorithms are proposed to support all major current approaches in model management.

2 Deductive Object-oriented paradigm

At the present time, programming languages are the product of developments that started in the 1950s. Numerous language concepts have been invented, tested and improved by being incorporated in successive languages. Among the present languages, there are two successful languages: The "Object-Oriented Language" and the "Logic Programming Language".

An object-oriented paradigm is based on the concepts of objects and classes of objects. When grouping data around objects, it is natural to try and tap into the potential of such representations by making use of class hierarchies, inheritance, typing etc. The inheritance mechanism encourages an application developer to reuse existing classes (objects) during construction of new applications.

A logic programming paradigm is based on the notion that a program is a set of axioms, or rules, defining relationships between objects. A computation of a logic program is a deductive of consequences of the program. A program defines a set of consequences, which is its meaning. The art of logic programming is constructing concise and elegant program that have the desired meaning.

One of the main problems with an object-oriented paradigm is the lack of logical semantics which traditionally were important for database programming languages. On the other hand, a logic programming paradigm uses a flat data model and does not support data abstraction. It therefore can be expected that a combination of the two paradigms will yield a new, powerful paradigm.

Several attempts have been made to combine the two paradigms: Wild LIFE [11], Frame logic or F-logic [11], Noodle [20] and QUIXOTE [24]. In the author's opinion, none have been entirely successful. A discussion of important aspects of the deductive object-oriented paradigm follows.

There are 2 important components in a deductive object-oriented paradigm: Object space (or universe) and Object relationships.

The object space is a set of objects with the following properties:
(i) Component: An object contains several components which are elements of the same object space.
(ii) Specialization: A combination of substitution and inheritance determine the mapping from one object to another object in the same object space.
(iii) Class: An object belongs to a certain class (type or sort) among partially order of classes.

Object relationships are relations among objects in an object space; they are represented by definite clauses.

There are four types of abstraction which are used in a deductive object-oriented paradigm: generalization, aggregation, association and classification. These abstractions can be applied to the object space, since generalization can be realized by using Specialization (second property) and Class (third property); classification and aggregation can be realized by using Content (first property); association can be regarded as a relation of objects in the object space, represented by deductive rules. The notion of a deductive object-oriented paradigm can be illustrated in a formal way by the following definition.

Definition 2.1 An Object space is a tuple \((O,C,S,T,S,X,f)\), where:
- \(O\) is a set of objects.
- \(C\) is a mapping function from a particular object on to its components which are also objects, \(C: O \rightarrow \text{POW}(O)\), where \(\text{POW}(O)\) denotes the power set of objects (O).
- \(S\) is a partial mapping function from a particular object on to its type (T);
- \(S: O \rightarrow \text{POW}(T)\).
- \((T,S)\) is a partially ordered set of object types in the object space.
- \(X\) is a set of object variables, \(X \subseteq O\), such that \(\forall X \in X, Cx = \emptyset\).
- \(f\) is a specialization mapping from a particular object into another object, which could be substitution and/or inheritance;
- \(f: (X \rightarrow O) \rightarrow (O \rightarrow O)\).
Specialization comprises two mechanisms: Substitution and inheritance. Substitution is a mapping from an object form on to another object form. On the other hand, inheritance is a mechanism for addition the set of components to an object which some components are obtained from its super-classes.

In order to build such an object space, the theory of ordered sort universe can be employed [23]. Based on this theory, one creates an object space, to start with, by selection of an ontology specifying all the possible forms of objects of interest. Then a fixed point of the ontology operation will determine an object space consisting of objects exemplifying the specified forms.

**Definition 2.2** A definite clause of a set of objects is the set of objects connected by an IMPLICATION connective, the antecedent part of which contains zero or more objects connected by AND connectives and the consequence part of which has an object, i.e., \( O_h \leftarrow O_1, O_2, \ldots, O_n \), where \( n \geq 0 \).

**Definition 2.3** An Object relationship is a connection among objects. An Object relationship can be represented by definite clauses, structured from objects in an object space.

**Definition 2.4** A deductive object-oriented program is a set of object relationships.

Since an object space can be regarded as a specialization system [2], a deductive object-oriented program could be considered as a generalized logic program (GLP) over a specialization system. Therefore, all the theoretical properties of GLPs can be applied to deductive object-oriented programs, e.g., a declarative semantics of a deductive object-oriented program is the minimal subset of \( O \) which makes all object relationships in the program to be true.

3 A Proposed Deductive Object-Oriented framework for model management

A deductive object-oriented paradigm has been used to represent decision models. In order to specify the relationship among each model class, inheritance rule between model classes has been described.

3.1 Model representation by means of a deductive object-oriented paradigm

On the basis of a deductive object-oriented paradigm, a model universe can be represented by an object space and object relationships. An object space is a tuple \((O,C,S,(T,s),X,f)\) in which \( O \) is a set of objects in forms of interest. A decision model is viewed as a class \((M)\) in the object space which each class is a set of functions. A function in each model class can be specified by \( \lambda \)-expressions. A model object \((O)\) is an instance of a model class \((M)\) in the object space denoting a special case of the model class. In addition, a model object uses all the functions which are specified in its model class. According to specialization, a combination of substitution and inheritance, each model class can inherit function properties from its model subclasses. The object relationship is employed in order to cope with aggregation of model classes for a representation of the model integration.

3.1.1 The object space in a model representation

Classes and objects are important components in the object space. In order to define the class and object for a model representation in a deductive object-oriented paradigm, it should be extended from the object-oriented conceptual model representation [22].

**Definition 3.1** A model class \((M)\) in the object space is a conceptual schema specifying a decision model. Based on a set of common data structure, it groups relevant functional expressions together in a fixed set. A model class \((M)\) can be represented as follows (Fig. 1): \[ M (input \Rightarrow (in\_attr_1 = e_1, in\_attr_2 = e_2, \ldots, in\_attr_m = e_m), output \Rightarrow (out\_attr_1 = e_{m+1}, out\_attr_2 = e_{m+2}, \ldots, out\_attr_{n-m} = e_n)) \]

**Fig. 1** General representation of a model in a model base

where \( M \) is the global name of the model class, \( e_i (1 \leq i \leq n, m < n) \) is a functional expression local to \( M \), \( in\_attr_i (1 \leq i \leq m) \) is an input attribute name of a functional expression, \( out\_attr_i (1 \leq i \leq n-m) \) is an output attribute name of a functional expression, \( \Rightarrow \) denotes a mapping from attribute name to its set value, \( \Rightarrow \)
denotes a mapping from attribute name to its attribute value.

In this representation of a decision model, a model class is composed of two sets of functional expressions, i.e., input set and output set. The input and output sets are the group of functional expressions which play the role of input and output, respectively, of a model class. The model class is the realization of the concepts of data abstraction. It possesses the following properties:

- There are no duplicate expressions in a model class. A combination of expressions in a class has the uniqueness property. This property can be uniquely identified by the class name (model name).
- Expressions of a model class are disordered. An expression of a model class can be a functional expression or a model class.

The functional expression, which is used to explain the computational procedure of a model, can be represented by a lambda expression in the typed $\lambda$-calculus (Lloyd, 1986) with the syntax:

\[ e ::= e(e) \quad \text{/* application, }\]
\[ \lambda x.e : t \quad \text{/* abstraction, }\]
\[ x : t \quad \text{/* variable, }\]
\[ c : t \quad \text{/* constant, }\]

\[ \tau ::= \text{Integer} | \text{Real} | \text{Boolean} \ldots \quad \text{/* type}\]

where typed expression $e$ (typed function) is used to represent basic operations in a decision model.

The typed expression $(e)$ and attribute names (in_attr, out_attr) are components of object (O) in the object space. The type of a function expression $(e)$ and a model name $(M)$ are components of object types $(T)$ in the object space.

In its purest form, the typed $\lambda$-calculus does not have built-in functions (such as ADD and composite types (as array). Since the present aims are practical, the pure typed $\lambda$-calculus with a suitable collection of such built-in functions has been extended to support composite types:

- Composite types of arrays. Scalars, vectors and matrices are arrays with different dimensions. Thus, a n-ary real vector has the type $R^n$ and a n x n-ary real matrix $R^{n x n}$ or $R^n$. The integer vector and matrix, respectively. $R^n$ is the type for the transposition of the vector with type $R^n$.

- The arithmetic functions ADD, SUB, MULT, DIV are used to represent addition, subtraction, multiplication and division on scalars, MADD, MSUB, MMLT, MDIV, MINV, MTNS, MINT are functions for the addition, subtraction, multiplication, division, inversion, transposition and floor respectively, on matrices.

- Logical functions (such as AND, OR, NOT, EQU, NEQ, LT, GT, etc.) and constants (TRUE, FALSE).

- Conditional functions

  \[ f : \text{boolean} \to \tau \]

  \[ f = \begin{cases} 
  e_1 & \text{if } e_l = \text{TRUE} \\
  e_2 & \text{if } e_l = \text{FALSE}
  \end{cases} \]

- Self-defined functions. For example, MIN is a function which finds the smallest number within a set. For two variables,

\[ \text{MIN}_2 = \lambda x_1, x_2. \text{IF } x_1 < x_2, x_1, x_2 \]

Example 1 An inventory plays a vital role in the operation of any business or enterprise. It should be efficiently managed, so that the total cost can be reduced. In the following inventory control model [21], it has been assumed that the demand is known and steady. The following notation is used:

Let $Q = \text{lot size per order},$

\[ Q^* = \text{optimal lot size per order}, \]

$R =$ units required (demand) per unit time,

$C_o =$ cost of ordering or setup per order placed,

$C_h =$ cost of holding a unit of inventory per unit time,

$C(Q) =$ total relevant cost (ordering + holding) per unit time for lot size $Q$.

For a simplest optimal lot size model, the total relevant cost is decided by:

\[ C(Q) = C_o \frac{R}{Q} + C_h \frac{Q}{2} \]

Then, the optimal lot size and the optimal relevant cost are determined by:

\[ Q^* = \sqrt{\frac{2RC_o}{C_h}} \]

\[ C(Q^*) = \sqrt{2RC_oC_h} = C_s Q^* \]

In reality, the history of the inventory should be considered, whence the problem refers to the
optimal lot size model with uniform replenishment:

Let \( R' = \frac{\text{maximum production possible per unit time}}{R} \), and \( v = \frac{1}{R'} \), the solution is then,

\[
Q^* = \frac{\sqrt{2RC_o}}{C_s v}
\]

\[
C(Q^*) = \sqrt{2RC_oC_s v} = C_s v Q^*
\]

By employing this new model representation, the simplest optimal lot size model can be specified in a model class "invs", and the improved inventory control model is specified in a model class "invar" as follows:

The model classes of inventory control models:

\[
\text{invs(input => (cost_of_order = Co:real, cost_of_holding = Ch:real, unit_require = R:real), output => (opt_lot_size = }\lambda R\text{.sqrt(mult(2,mult(R,div(Co,Ch))))), opt_total_rel_cost = }\lambda R\text{.sqrt(mult(2,mult(R,mult(Co,Ch)))) })
\]

\[
\text{invar(input => (factor = V:real), output => (opt_lot_size = Q:real, opt_total_rel_cost = CQ:real, imp_opt_lot_size = }\lambda Q,\lambda V\text{.div(Q,sqrt(V))), imp_opt_total_rel_cost = }\lambda CQ,\lambda V\text{.mult(CQ,sqrt(V))) })
\]

The inheritance rule of these model classes:

\[\text{invar} \leq \text{invs}\]

This new representation of these models shows that the improved inventory control model (INVAR) can inherit properties, i.e., the input set \{cost_of_order, cost_of_holding, unit_require\} and the output set \{opt_lot_size, opt_total_rel_cost\} from the simplest optimal lot size model (INVS). Inheritance helps to specify the relationship among the models for the reuse of the decision models, thus improving the quality of the class scheme and encouraging reduction of redundancy in the model base.

**Definition 3.2** An object (O) in the object space is an instance of a model class (M), denoting a special case of the model class. It uses all the functions in that class.

**Example 2** Each unit of the Model 777 electric shaver of the Little Shaver Company requires an on-off switch, which is manufactured at the company's plant. In the coming years 100,000 units of the Model 777 are to be produced, at a constant rate. For each production run of the switch, there is a $40 setup cost. The switch production capacity would be 200,000 per year, if switches were being produced on a continuous basis. The annual holding cost per unit is considered to be 20 percent of the item cost, which for this switch, is $0.25.

From the problem data, one can know that

- unit require (R) = 100,000
- maximum production possible per unit time (R') = 200,000
- cost of order (Co) = $40
- cost of holding (Ch) = 0.20 * 0.25 = $0.05
- uniform replenishment factor (v) = 1 - (R/R')

\[= 1 - (100,000/200,000) = 0.5\]

The objects of the invs and invar model in this example can be given as follows:

\[
\text{invs(input => (cost_of_order = 40, cost_of_holding = 0.05, unit_require = 100000), output => (opt_lot_size = 12649.11), opt_total_rel_cost = 632.45 ))}
\]

\[
\text{invar(input => (factor = 0.5), output => (opt_lot_size = 12649.11, opt_total_rel_cost = 632.45, imp_opt_lot_size = 17889.54, imp_opt_total_rel_cost = 447.21))}
\]

In this case, a user of inventory control model want to know the optimal lot size and the associated total relevant inventory cost. Thus the objects of these models are the solution of these models.
3.1.2 The object relationship in a model representation

Relationships between models are represented by object relationships, defined by deductive rules. The object relationship is a kind of definite clause or rule that concerns about objects instead of predicates, e.g.,

\[ M_1(\text{input} \rightarrow (\text{in\_attr}_1 = E_1, \text{in\_attr}_2 = E_2, \ldots, \text{in\_attr}_{m_1} = E_{m_1}), \]
\[ \text{output} \rightarrow (\text{out\_attr}_1 = E_{m_1+1}, \text{out\_attr}_2 = E_{m_1+2}, \ldots, \text{out\_attr}_{n_m} = E_n) \]

\[ M_2(\text{input} \rightarrow (\text{in\_attr}_1 = E_1, \text{in\_attr}_2 = E_2, \ldots, \text{in\_attr}_{m_2} = E_{m_2}), \]
\[ \text{output} \rightarrow (\text{out\_attr}_1 = E_{m_2+1}, \text{out\_attr}_2 = E_{m_2+2}, \ldots, \text{out\_attr}_{n_2} = E_{n_2}) \]

\[ M_3(\text{input} \rightarrow (\text{in\_attr}_1 = E_1, \text{in\_attr}_2 = E_2, \ldots, \text{in\_attr} = E_m), \]
\[ \text{output} \rightarrow (\text{out\_attr}_1 = E_{m+1}, \text{out\_attr}_2 = E_{m+2}, \ldots, \text{out\_attr}_{n_3} = E_{n_3}) \]

\[ M_n(\text{input} \rightarrow (\text{in\_attr}_1 = E_1, \text{in\_attr}_2 = E_{j+1}, \ldots, \text{in\_attr}_{n_n} = E_{mn}), \]
\[ \text{output} \rightarrow (\text{out\_attr}_1 = E_{mn+1}, \text{out\_attr}_2 = E_{mn+2}, \ldots, \text{out\_attr}_{n_n} = E_{n_n}) \]

where \( M_i \) ( \( 1 \leq i \leq n \)) are model objects of the model classes. The goal of clauses \( M_1 \) represents an integrated model and each sub-goal specified attributes of the integrated model. Common variables inside sub-goal objects show interconnections among the model objects.

3.2 Inheritance Rule between Classes

Most of the expressions in a model are typed function abstractions which in the case of modeling of a decision model have the form: \( f: \rho \to \tau \), for \( x: \rho \) and \( e(x):\tau \). Within this functional framework, some function rules can be derived:

**Definition 3.3** If there exist domains \( \rho \subseteq \rho \) and the function \( f: \rho \to \tau \), then \( f: \rho' \to \tau \) is a specialized function of \( f: \rho \to \tau \).

In mathematics, a function is defined as a mapping from a domain on to a range, when, for any value \( x \in \rho' \subseteq \rho \), there exists \( f(x) \in \tau \). Thus, the specialized function \( f: \rho' \to \tau \) shares the same function implementation with \( f: \rho \to \tau \).

**Example 3** In the simplest optimal lot size inventory model (example 3.1), if \( R \) is an integer, denoting the number of units required per unit time. Hence, the optimal lot size function \( Q^* \) becomes,

\( Q^*:\text{INTEGER} \to \text{REAL}; \)
\( Q^*=\lambda \sqrt{2\cdot \text{multi}(R,\text{div}(\text{Co},\text{Ch})))}; \)

Since \( \text{INTEGER} \subseteq \text{REAL} \), \( Q^*:\text{INTEGER} \to \text{REAL} \) is a specialized function of \( Q^*:\text{REAL} \to \text{REAL} \) in the model class "inv".

**Definition 3.4** If there exists the domains \( \tau' \subseteq \tau \) and the function \( f: \rho \to \tau \), such that \( f: \rho \to \tau' \) is meaningful, then \( f: \rho \to \tau' \) is a specialized function of \( f: \rho \to \tau \).

**Example 4** In the stock cutting (SC) problem model [21], since \( c \), \( A \) and \( b \) are an integer coefficient vector, an integer constraint matrix and an integer resource vector, respectively, the solution function is a specialized function abstraction of

\( \text{IPGOAL}: \)
\( ((R_0 \times R_s \times R_u \to R_s) \to IR_s \to \text{REAL}) \) in

the model class \( \text{IP} \):

\( \text{IPGOAL:}(R_s \to IR_s) \to \text{INTEGER}; \)

\( \text{IPGOAL:MIN(MMLT(MTNS(e),Y) where} \)
\( Y = \text{MINT(2)}, \)
\( \text{and X = MMLT(MINV(A),b)} \}

\( \text{IPGOAL} \) first performs the simplex method to get the real optimal results, and then executes the branch & bound method to reach the integer optimal result. The final optimal value has the type integer.

**Definition 3.5** If there exist domain \( \tau' \subseteq \tau \), \( \rho' \subseteq \rho \) and the function \( f: \rho \to \tau \), such that \( f: \rho' \to \tau' \) is meaningful, then \( f: \rho' \to \tau' \) is a specialization function of \( f: \rho \to \tau \).

**Example 5** For a given function \( f: \text{REAL} \to \text{REAL}; f = \lambda x.\text{add}(a,\text{multi}(b,x)) \), if \( a \) and \( b \) are integer constants and the production volume is integer, then this function becomes:

337
f:INTEGER → INTEGER;
  f(x) = λx.add(a,mult(b,x))

which denotes a mapping: INTEGER → INTEGER, and is a specialized function of
f:REAL → REAL.

The inheritance relationship between two classes can be derived from the inheritance rules for their specialized functions.

**Definition 3.6** For a given class M and a new class M' which are defined as follows:

M (input => (in_attr1 = e1, ... , in_attrm = em),
    output => (out_attr1 = e_{m+1}, ..., out_attrn-m = en ))

M' (input => (in_attr1 = e'1, ... , in_attrp = e'p),
    output => (out_attr1 = e'_{p+1}, ..., out_attrq-p = e'_q ))

where q ≥ n and p ≥ m if an expression e'_i (for some i in (1 ≤ i ≤ p)) is a specialized function of the corresponding expression e_i in Class M, then M' is a subclass of M, which inherits the function from its super-class M. One can write

M' ≤ M,

M' (input => (in_attr1 = e'1, ... , in_attrk = e'k),
    output => (out_attr1 = e'_{k+1}, ..., out_attr_{k+j} = e'_j ))

where e'_j (1 ≤ j ≤ q) is not a specialized function of its corresponding expression in the super-class M.

**Remark** The declaration M' ≤ M can be read as "M' is a subclass of M".

**Definition 3.7** Multiple inheritance is a mechanism to allow a model class to be defined by inheriting function properties from more than one superclasses. Formally, if there is a model class M which is defined as follows:

M' (input => (in_attr1 = e'1, ... , in_attrm = e'_m),
    output => (out_attr1 = e'_{m+1}, ..., out_attrn-m = e'_n ))

And model class M' is a subclass of a set of classes (M_1, M_2, ..., M_n) where

M_i (input => (in_attr1 = e_i1, ... , in_attrm = e_im),
    output => (out_attr1 = e_{im+1}, ..., out_attrn-im = e_{in} ))

for any i in 1 ≤ i ≤ n then it can be denoted that

M' (input => (in_attr1 = e'_{j1}, ... , in_attrm = e'_jm),
    output => (out_attr1 = e'_{jm+1}, ..., out_attrn-m = e'_jn))

which each e'_{jl} (1 ≤ jl ≤ m) is either an incremental modification of the corresponding expression in model class T_i (1 ≤ i ≤ n) or expression which never appears in its superclasses.

**Example 6** The integer programming (IP) model can be specified as follows:

IP(input => (coefficient_vector = c; R_n, constant_matrix = A; R_n^a, resource_vector = b; R_n, output => (ipgoal=λc; λA; λ)), b.MIN(MM.LT(MTNS(c), MINT(MM.LT(MINV(A),b)))))), solutions = λA; λ). b.MIN(MTNS(A),b))).

with ipgoal:(R_n → IR_n) → Real and solutions: R_n.

The linear programming (LP) model can be specified as follows:

LP(input => (coefficient_vector = c; R_n, constant_matrix = A; R_n^a, resource_vector = b; R_n, output => (goal=λc; λA; λ), b.MIN(MM.LT(MTNS(c), MM.LT(MINV(A),b)))), solutions = λA; λ). b.MIN(MINV(A),b))).

with goal: R_n → Real and solutions: R_n.

Obviously, a solution in IP is a specialized function of solutions in LP, and ipgoal is an incremental modification of goal in LP. Hence, IP is a sub-class of LP, which can be specified by inheritance of function properties from LP.

Inheritance rules help to discover specialized functions in order to reduce the redundancy between two classes.
4 Conversion of Model Representation

Conversion algorithms of model representation are now presented. This will help to show that current approaches to model representation are special cases of the deductive object-oriented approach. Conversion descriptions of each approach have been described as follows, and the conversion algorithms have been given in Appendix.

4.1 The Relational Approach

The relational model is composed of a virtual relation (R) or a subset of the Cartesian product of a set of domains corresponding to input and output attributes; a set of key attributes; a set of non-key attributes; a set of tuples of relation (R) and join operation of relation (R).

1. **Key attribute**: Group a set of attributes which can be a candidate key of a virtual relation (R) as an input set of a model class (C).
2. **Non-key attribute**: Group a set of non-key attributes of a virtual relation (R) as an output set of a model class (C).
3. **Virtual relation (R) or Table**: Define a virtual relation (R) or table as a model class (C) which composes of parameters (function without arguments) playing the roles of inputs or outputs to a model.
4. **Tuple**: Define a tuple of a virtual relation (R) as an object (O) which belongs to a model class (C).
5. **Join**: The join of models in a model set is represented by deductive rules in an object relationship.

**Remark**
- A **superkey** is a set of one or more attributes, which, when taken collectively, allow us to identify uniquely an entity or relation.
- A **candidate key** is a subset of the attribute of a superkey that is also a superkey and not reducible to another superkey.

4.2 The Entity-relationship Approach

The ER model is composed of entities, attributes and relationships.

1. **Entities**: Define an entity in ER-diagram as a model class which has relationships with other entities. The model class will has its own attributes which are represented by a set of functions.
2. **Attributes**: Define attribute names of each entity as static property of an entity (model class). The attribute names can be classified in two set, i.e., input attribute set and output attribute set. In addition, attribute values which are computational attributes will become functional expressions of a model class which are defined by lambda expressions.
3. **Relationships**: A relationship in ER diagram is defined as an object relationship which is represented by a deductive rule.

4.3 The Structured Modeling Approach

Structured model is composed of five important components, i.e., elementary structure, generic structure, modular structure, model schema and elemental detail table.

1. **Elementary Structure**: Specify all of the definitional detail of a specific model instance. There are five types of elements, i.e., primitive, compound, attribute, function and test. The elementary structure is used to define components of model classes.
2. **Genus Structure**: Capture the natural familial grouping of elements. This is accomplished by partitioning all elements of a given type into genera. The genus structure is used to define the relationship among sub-models in the model set.
3. **Modular Structure**: Organize generic structure hierarchically by grouping genera into conceptual unit called modules according to commonality or semantic relatedness. The modular structure is used to classify sub-models of a model set. A model class is defined to represent each sub-model and a main model class is defined to represent a model set.
4. **Model Schema**: Express the generic and modular structure in text-based form. The model schema is used to define attribute name and attribute value of each model class.
5. **Elemental Detail Table**: Describe a particular instance of the general class of models represented by a schema. The elemental detail table is used to define objects corresponding to each model class in the model set.
4.4 The Model Abstraction Approach
The model abstraction is composed of abstract models, data objects, procedures and assertions.
1. Abstract models: An abstract model in this approach is defined as a model class in
deductive object-oriented approach.
2. Data objects: Define data objects of an
abstract model as attributes of a model class.
Data objects is used to specify type of attributes
of the model class.
3. Procedures: Regard procedures of an abstract
model as functional expressions of output
attributes of a model class. In addition,
parameters of the procedures is used to classify
the input or output attributes of the model
class.
4. Assertions: Assertions is used to specify
relationship among the data objects and
procedures. There are 2 purposes of assertions.
First purpose, an assertion (specified data
objects constraint of a abstract model) is
represented by a deductive rule in deductive
databases. Second purpose, an assertion
(specified model generalization) is represented
by an inheritance rule in object space.

4.5 The Object-oriented Approach
The object-oriented model [22] is composed of
model classes, a set of expressions and inheritance
rules.
1. Model classes: A model class of the object-
oriented approach is defined as a model class in
the deductive object-oriented approach.
2. Set of expressions: Group a set of expressions
which type of an expression is function
abstraction as an output set of model class. And
also group a set of expressions which type of an
expression is variable and constant as an input
set of model class.
3. Inheritance rules: Define inheritance rules
between classes as inheritance rules in the
object space.

Conclusions
New model representation for model
management systems are proposed in this paper.
By using this new representation, it is possible to
improve the conceptual model representation
to power to meet the main requirements of model
representation, i.e., reduction of redundancy of
models, elimination of inconsistencies, sharing of
models, reusing of existing models, eas of use of
models and standard of specification of models.
Conversion Algorithms from other major
representations to the proposed representation
have been developed to show that the new model
representation can support all existing model
representations.
The proposed framework has several
advantages. First, it could be employed as a
common representation for model bases. Second,
the proposed inheritance mechanism enables the
reuse of existing models. Thus redundancy and
inconsistency in a model base can be reduced. The
inference mechanism is employed to integrate
models in the model base. So the compactness of
the model base can be improved. Finally, the
proposed conversion algorithms can be employed
to integrate all major current frameworks into a
unique framework.

References
H. ALT-KACI, B. DUMANT, R. MEYER, A.
PODELSKI, and P.V. ROY, The basic data
structure, The Wild LIFE Handbook
(prepublication edition), Digital research
K. AKAMA, Generalized Logic Programs on
Specialization Systems and SLDA Resolution,
Technical Report HIER-LI-9002, Hokkaido
University, Japan (1990b).
R.W. BLANNING, A Relational FrameWork for Join
Implementation in Model Management Systems,
Decision Support Systems, North-Holland, Vol. 1,
R.W. BLANNING, An Entity-Relationship Approach
to Model Management, Decision Support Systems,
R.W. BLANNING, Model Management Systems,
Decision Support Systems: putting theory into
practice, 2nd edition, R.H. Sprague, JR & H.J.
Watson eds. Prentice-Hall, Inc., pp. 156-169
(1989).
Y.S. CHEN, An Entity-Relationship Approach to
Decision Support and Expert Systems, Decision
Support Systems, North-Holland, Vol. 4, pp. 225-
D.R. DOLK and B.R. KONSYNISKI, Knowledge
Representation for Model Management Systems,


