An Object-oriented Approach to Model Management

Vilas WUWONGSE **Jian MA**

Division of Computer Science, Asian Institute of Technology
G.P.O.Box 2754, Bangkok 10501, THAILAND

Abstract

The goal of this research is to propose an object-oriented framework for the specification, storage and manipulation of decision models in conceptual analysis of model management. The framework consists of two major parts: a conceptual structure, within which a decision model is represented in the form of a class and an inheritance mechanism enables a new class to be specified by making use of properties in existing ones; and manipulation operations, which is a set of high-level operators for the retrieval and update operations in a model management system. Based on this framework, the inheritance rules have been derived from function specialization within classes. As a result, an inheritance rule checking system has been designed and implemented to find out the inheritance relationship between two classes.

I. Introduction

Current research [1, 6] in model management of decision support systems (DSS) has shown that the decision model is a kind of information resource which should be efficiently managed like data. The main purposes for model management are:

1) to reduce redundancy: In early DSSs, each application had its own implementation of functions or procedures for decision models; this led to considerably redundancy in stored functions as well as multiplying the task of the model builder.

2) to avoid inconsistency: Inconsistency in model management is caused by redundancy, which allows different versions of function for the same model to exist in a centralized model management system (MMS). Thus, ambiguous results may be generated when executing a function.

3) to share models with multiusers: Different users may have various requirements. A MMS should allow not only existing applications to share the implementation of functions for decision models, but also new applications to be built by making use of the existing ones.

4) to allow decision makers to use models easily: Related implementations of functions for a decision model should be grouped together, so that decision makers can use them easily.

5) to enforce standards: Standards for the specification of decision models are desirable as an aid for model sharing, understandability and model interchange between systems.

Several conceptual frameworks have been proposed for the representation of decision models in order to meet the requirements for model management. These include the relational framework, which views a model as a virtual relation between input and output domains [1]; the entity-relationship (E-R) approach, which treats a model as an entity comprising a number of attributes, and an interface among models as a relationship [2]; the model abstraction approach, which adopts the concept of data abstraction in programming languages and encapsulates related operations on a common data structure within a model [6]; and the structured modeling method, which employs a hierarchically organized, partitioned, and attributed acyclic graph to represent relationships among model elements (e.g. entities, attributes and functions) [8]. The advantages/disadvantages of these approaches are compared in Table 1.

Table 1: Comparison of conceptual frameworks for model management.

	Relational	E-R	Model abstraction	Structured modeling
Reduce redundancy	P.S.	P.S.	N.S.	W.S.
Avoid inconsistency	P.S.	P.S.	N.S.	W.S.
Share models	P.S.	P.S.	N.S.	W.S.
Used easily	W.S.	W.S.	W.S.	N.S.
Enforce standard	W.S.	W.S.	W.S.	N.S.

Notes: W.S.: wholely support; P.S.: partially support; N.S.: not support.

The relational, E-R and model abstraction approaches do not wholly support model sharing, hence redundancy exists and inconsistency is not avoided. Structured modeling method specifies the relationship among models in great detail, but the resulting graph does not correspond to a real decision model. It is hard to use and difficult to understand.

The purpose of this paper is to propose a conceptual framework, based on the object-oriented paradigm, for model specification, storage and manipulation. The proposed framework is in accordance with model abstraction approach in the sense that related functions are grouped together on a common data structure, forming a class. With the extension of the inheritance mechanism, a new class can be built by making use of the function properties in its superclass. Based on the framework, some useful inheritance rules can be derived to help reduce redundancy in designing a centralized model management system.

II. Proposed Object-oriented Framework for Model Management

2.1 Domains, Functions and λ-calculus

For our purposes, a domain is a set of values with certain properties. A domain may include a special symbol $\perp$ (read "bottom") to represent the meaning of undefinedness. Primitive domains are those widely used in the computation (e.g. INTEGER, REAL, CHARACTER and BOOL). Composite domains are those constructed by primitive domains using such operations as UNION, INTERSECTION, DIFFERENCE and PRODUCT (defined in the usual manner). Type is a synonym of domain used to protect an underlying representation from arbitrary or unintended use. Type and domain are used interchangeablely in this context.

A function f is a direct mapping from domain A to codomain B, written $f:A\rightarrow B$. In general notation, $y=f(x)$ means $\langle x,y\rangle:f$, where $x:A$ and $y:B$.

In a mathematical representation of decision making, there is an assumption that any decision model can be abstracted and represented in the form of functions [7].

EXAMPLE 2.1: In a production line, the Total Cost (TC) is linearly decided by the Production Volume (PV):

$$TC = a + b \times PV, \tag{2.1}$$

where a and b are constants. If we use f and x to represent TC and PV, respectively, Equation 2.1 becomes:

$$f(x) = a + bx \tag{2.2}$$

Functions can be classified by their mappings. Some classifications are:

identity: $f:A\rightarrow A$ is the *identity* function for A iff for all $x:A$, $f(x)=x$.

composite: for functions $f:B\rightarrow C$ and $g:A\rightarrow B$, where g is *total*, then the function $(fog)(x) = f(g(x))$ is called *composite* function of f and g, which denotes a mapping $(A\rightarrow B)\rightarrow C$.

λ-calculus [11] is a formal system for the study of functions, their definitions and applications. It captures exactly those functions that can be computed by mechanical or electronic means, and has shown to be equivalent to Turing machine. We use λ-calculus as a formal language to specify functions in the framework.

The basic λ-calculus has the syntax:

$$e ::= e(e) \quad \text{/* application,} \tag{2.3}$$

$$\lambda x.e(x) \quad \text{/* abstraction,} \tag{2.4}$$

$$\lambda x \quad \text{/* variable,} \tag{2.5}$$

$$\lambda c \quad \text{/* constant,} \tag{2.6}$$

where expression e refers to a function abstraction, which represents a mapping from input to output domains.

EXAMPLE 2.2: In Equation 2.2, x is a variable, a+bx is an expression in which x is free, and must either be defined globally to the expression or be undefined:

$$f=\lambda x.(a+bx) \tag{2.7}$$

where x is bounded by the λx and (a+bx) is an abstraction.

2.2 Conceptual Structure of an Object-oriented Framework

2.2.1 Class and Object

A typed expression e is a typed function used to represent basic operations in a decision model. It is described by typed λ-calculus and has the following syntax:

$$e ::= e(e) \quad \text{/* application,} \tag{2.8}$$

$$\lambda x{:}\tau.e(x){:}\tau \quad \text{/* abstraction,} \tag{2.9}$$

$$\lambda x{:}\tau \quad \text{/* typed variable,} \tag{2.10}$$

$$\lambda c{:}\tau \quad \text{/* typed constant,} \tag{2.11}$$

where τ is a type with the syntax:

$$\tau ::= \text{INTEGER/BOOLEAN/CHARACTER/REAL}$$

$$/..../\tau \times \tau/\tau \rightarrow \tau/(\tau) \tag{2.12}$$

Typed expressions (abbreviated as expressions, thereafter) are the basic units used to describe a decision model. For simplicity, we use $f{:}\tau \rightarrow \tau$, $f=\lambda x.e(x)$ to represent function abstraction $f=\lambda x{:}\tau.e(x){:}\tau$.

EXAMPLE 2.3: With the addition of types, Equation 2.7 becomes:

f:REAL→REAL;
f=λx.(a+bx) (2.13)

As a shorthand, we allow e' where x:τ= e to represent function abstraction $\lambda x{:}\tau.(e')(e)$.

DEFINITION 2.1: A class is a conceptual schema specifying a decision model. Based on a set of common data structures, it groups relevant expressions together in a fixed set:

$$C = \{e_1, e_2, \ldots, e_n\} \tag{2.14}$$

where C is the name of the class and each $e_i (i \leq i \leq n)$ is an expression.

Class is the realization of the data abstraction concept. In Equation 2.14, expressions are mutually exclusive and orders among them are not important.

EXAMPLE 2.4: Linear programming (LP) [7] is a basic model in operations research or management science (OR/MS), where the simplex method algorithm is commonly used to compute its optimal solutions. In practice, sensitivity analysis always follows the optimal result. Let c, A and b be the cost coefficient vector, constraint matrix and resource vector, respectively, where the vectors have a n-ary REAL type R_n and the matrix R_n^n. LP is then specified in the following class schema:

LP={ c:R_n;
c=λc;
/* input coefficient vector c
A:R_n^n;
A=λA;
/* input constraint matrix A
b:R_n;
b=λb;
/* input resource vector b
GOAL:$(R_n^n \times R_n \rightarrow R_n) \rightarrow$ REAL;
GOAL=min { cX where X:R_n=A^{-1}×b};
/* use simplex method algorithm to solve the problem
SOLUTIONs:R_n;
SOLUTIONs=λX;
/* print standard solution report
RANGE:$R_n \rightarrow R_n$;
RANGE=λX.(X)(A^{-1}·b) where GOAL*=cX;
/* to test the range of X, while still keep the optimal result
}.

In this object-oriented framework, a class schema specifies a general decision model, but its instance is represented by an object.

DEFINITION 2.2: An object is an instance of a class, denoting a special case of the model class. Once defined, it uses all the functions in that class. Formally, if an object O is an instance of class C, it is denoted by O:C.

EXAMPLE 2.5: To decide what is the best percentage of investing one million US$ in projects AA and BB is an object, INVEST, belonging to class LP. Object INVEST is specified by INVEST:LP. It uses all functions defined in class LP, e.g. INVEST.GOAL is to execute the simplex method algorithm to get optimal solutions.

2.2.2 Inheritance

In model management, incremental modification is a technique to compose a modifying function with an existing one. In typed λ-calculus, it is represented by:

$$sc = \lambda p, \lambda x.mop(x) = \lambda p, \lambda x.\ m(p(x)) \tag{2.15}$$

where sc, p and m are new, existing and modifying functions, respectively. If p(x) has function type $\tau \rightarrow \tau$, sc must have the composite function type $(\tau \rightarrow \tau) \rightarrow \tau$.

In some cases, m may be the identity function, i.e. m=$\lambda y.y$, resulting in sc being the same as p. In other words, sc is a direct copy of p. However, in most cases, m is an expression acting on the values returned by p.

EXAMPLE 2.6: In the integer programming model [7], one way to get the optimal result is to use the simplex method algorithm to obtain the real optimal result and then execute the branch&bound algorithm to reach the integer optimal solution. Therefore, GOAL is an incremental modification for GOAL in class LP:

$$\text{GOAL}:((R_n^n \times R_n \rightarrow R_n) \rightarrow IR_n) \rightarrow \text{REAL};$$

GOAL=min{ cX where $X:IR_n=A^{-1} \cdot b$ }.

DEFINITION 2.3: Inheritance is a mechanism to allow a new class to be defined by inheriting properties from existing ones. If $SC = \{sc_1, sc_2, ..., sc_m\}$ is a subclass of $P = \{p_1, p_2, ..., p_n\}$, SC should have the same or more numbers of expressions than P ($m \geq n$) and the expressions in SC are either direct copies or incremental modifications of the corresponding expressions in P. Formally, SC is denoted by:

$$SC = \{sc_1, sc_2, ..., sc_m\}$$
$$\textit{inherit from}\ P., \tag{2.16}$$

where each expression sc_i is either a copy or an incremental modification of expression p_i.

EXAMPLE 2.7: In the multi-objective linear programming model [12], each objective function has a weight. One possible approach to determine the weights for each objective function is to use the fuzzy delphi mathematics (FDM) [9] method, which can be implemented in the class FDM below.

FDM={ Rwt:R_n^m;

Rwt=λRwt;

/* input n×m-ary real matrix of the raw weights

W:$R_n^m \rightarrow R_n^{-1}$;

W=$\lambda Rwt.\left(\left(\sum_{j=1}^{m} Rwt_{ij}\right)/m\right)$;

/* evaluate proper weights for each objective function

};

With the proper weights for corresponding objective functions, MOLP model can then be specified in the class MOLP:

MOLP={ b:R_n;

A:R_n^n;

cc:R_n^m;

cc=λcc;

wc:$R_m^{-1} \times R_n^m \rightarrow R_n^{-1}$;

wc=$\lambda W, \lambda cc$ (W×cc);

/* to combine the weight with each objective function

GOAL:$(R_n^n \times R_n) \rightarrow$ REAL;

GOAL=min { (wcX where X:R_n=A^{-1}b) (wc/c)};

/* use simplex method algorithm to solve the problem

SOLUTIONs:R_n;

SOLUTIONs=λX;

/* print standard solution report

RANGE:$R_n \rightarrow R_n$;

RANGE=λX.(X)(A^{-1}·b) where GOAL*=cX;

/* to test the range of X, while still keep the optimal result

},

where b, A, GOAL, SOLUTIONs and RANGE are direct copies of corresponding expressions, and cc is an incremental modification of c in class LP. w is a direct copy of expression from class FDM, and wc is a new function abstraction to combine weights with corresponding objective functions. Hence, class MOLP can be abbreviated by specifying it to inherit function properties from class LP and FDM:

MOLP = { wc:$R_m^{-1} \times R_n^m \rightarrow R_n^{-1}$;
wc=$\lambda W, \lambda cc$.(W×cc);
/* to combine the weight with each objective function
}
inherit from: FDM, LP.

The above example shows a new class can be built by inheriting function abstractions from existing ones. In the next example of inventory control model, we show a different variety of inheritance to allow a new class to be incrementally modified from existing classes.

EXAMPLE 2.8: 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 [12], we assume that the demand is known and steady. The notations used are:

Let Q = lot size per order,
Q^* = optimal lot size per order,
R = units required (demand) per unit time,
C_0 = 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:

$$C(Q) = C_0 \frac{R}{Q} + C_h \frac{Q}{2} \tag{2.17}$$

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

$$Q^* = \sqrt{\frac{2RC_0}{C_h}} \tag{2.18}$$

$$C(Q^*) = \sqrt{2RC_0C_h} = C_hQ^* \tag{2.19}$$

The simplest optimal lot size model (INVS) can be specified in class INVS:

```
INVS={ Q:REAL;
         Q=λQ;
       R:REAL;
         R=λR;
       Q*:REAL→REAL;
         Q*=λR.(SQRT(2*R*C0/Ch));
       C(Q*):REAL→REAL;
         C(Q*)=λR.(SQRT(2*R*C0*Ch));
       }.
```

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

Let R'=maximum production possible per unit time (R'>R), and $v = 1 - R/R'$, the solution is then,

$$Q^* = \sqrt{\frac{2RC_0}{C_h v}} \tag{2.20}$$

$$C(Q^*) = \sqrt{2RC_0C_h v} = C_h v Q^* \tag{2.21}$$

The improved inventory control model is then specified in class INVUR:

```
INVUR={ Q*:(REAL→REAL)×REAL→REAL;
          Q*=λv.(INVS.Q*/SQRT(v));
        C(Q*):(REAL→REAL)×REAL→REAL;
          C(Q*)=λv.(INVS.C(Q*)*SQRT(v));
      }
inherit from INVS.
```

Inheritance helps to specify the relationship for the reuse of decision models, which improves the quality of class schema. Class and inheritance together constitute the object-oriented framework for conceptual analysis in model management.

2.3 Manipulation Operations in the Object-oriented Framework

The proposed object-oriented framework is a conceptual framework for the representation of decision models. In order to allow users to manipulate the classes at an abstract level, while neglecting detailed implementation of functions within classes, and to provide an efficient minimum subset of the language that can express enough things to make a model management system useful, a set of manipulation operations for the framework needs to be defined.

Now recall that a class is a conceptual schema specifying a decision model. Which groups relevant expressions together on a common data structure in a fixed set. Therefore, the manipulation operations for the object-oriented framework should include two parts, namely:

1) the traditional set operations: UNION, INTERSECTION and DIFFERENCE;
2) the special operations: FIND, PROJECT, INSERT, DELETE and MODIFY. They are defined as follows:

FIND: to select class names or expression lists in a MMS, whenever some conditions are met:

FIND target-list WHERE condition

PROJECT: to form a new class with the extracts of expressions from a specified class:
PROJECT class-name WITH expr-list
INSERT: to create a new class in a model management system:
INSERT class-name WITH expr-list
DELETE: to delete a specified class in a model management system:
DELETE class-name
MODIFY: to update the expressions in a specified class:
MODIFY expr WITH expr IN class-name

Since a class corresponds to the concept of a fixed set, where each expression is an element in the set. The completeness of above operations is obvious due to the fact that they support the traditional set operations. Manipulation operations on the conceptual structure of the object-oriented framework is simpler than relational algebra in the relational database model. Their syntax can be defined as shown in Figure 2.1:

```
class-defn ::= DEFINE class AS class-name = { expr-list }
oprt ::= find / project / insert / delete / modify / infix-expr
find ::= FIND target-list WHERE condition
target-list ::= class-name / expr-list
project ::= PROJECT class-name WITH expr-list
insert ::= INSERT class-name WITH expr-list
delete ::= DELETE class-name
modify ::= MODIFY expr WITH expr IN class-name
expr-list ::= expr / expr-list, expr
expr ::= id : type
infix-expr ::= class infix-op class
infix-op ::= UNION / INTERSECT / DIFFERENCE
condition ::= condition op condition / class-name cmp-op constant
            / id cmp-op constant / type cmp-op constant
cmp-op ::= < / ≤ / = / > / ≥
op ::= AND / OR / NOT
```

Figure 2.1: A BNF grammar for manipulation operations in the object-oriented framework.

Even though a commercial MMS may include many other operations for a user friendly interface, the designed syntax can meet the minimum requirement for manipulation operations in object-oriented model management.

III. Design of Model Management

3.1 Inheritance Rules

An object-oriented framework consisting of a conceptual structure and its manipulation operations has been proposed for model management. However, one question still remains to be answered: given a set of decision models to be represented in an object-oriented framework, how we decide on what classes are needed and what the inheritance relationships among them are. This is the design problem of model management.

Now, recall the proposed object-oriented framework in Equation 2.14. Most of the expressions are typed function abstractions when modeling a decision model, which have the form: $f:\rho \rightarrow \tau$, for $x:\rho$ and $e(x):\tau$. Within this functional framework, some rules for functions can be derived:

RULE 1: If domains $\rho \supset \rho'$ and $f:\rho \rightarrow \tau$, then $f:\rho' \rightarrow \tau$ is a specialized function of $f:\rho \rightarrow \tau$. $f:\rho' \rightarrow \tau$ shares the same function implementation with $f:\rho \rightarrow \tau$.

EXAMPLE 3.1: In the simplest optimal lot size inventory model of Example 2.8, R is an integer, denoting the units required per unit time. Hence, the optimal lot size function becomes,

Q^*:INTEGER→REAL;
$Q^*=\lambda R.(SQRT(2*R*C_0/C_h))$;

which is a specialized function of Q^*:REAL→REAL in class INVS.

RULE 2: If domains $\tau \supset \tau'$, $f:\rho \rightarrow \tau$ and $f:\rho \rightarrow \tau'$ is meaningful, then $f:\rho \rightarrow \tau'$ is a specialized function of $f:\rho \rightarrow \tau$. $f:\rho \rightarrow \tau'$ shares the same function implementation with $f:\rho \rightarrow \tau$.

EXAMPLE 3.2: In the stock cutting problem model [12], since c, A and b are integer coefficient vector, integer constraint matrix and integer resource vector, respectively. The solution function is then a specialized function abstraction of $GOAL:((R_n^n \times R_n \rightarrow R_n) \rightarrow IR_n) \rightarrow$ REAL in class IP:

$GOAL:((R_n^n \times R_n \rightarrow R_n) \rightarrow IR_n) \rightarrow$ INTEGER;
GOAL=min{ cX where $X:IR_n=A^{-1} \times b$ },

where GOAL first performs the simplex method to get the real optimal results, and then executes branch&bound method to reach integer optimal result. The final optimal value has the type integer.

RULE 3: If domains $\tau \supset \tau', \rho \supset \rho'$, $f:\rho \rightarrow \tau$ and $f:\rho' \rightarrow \tau'$ is meaningful, then $f:\rho' \rightarrow \tau'$ is a specialized function of $f:\rho \rightarrow \tau$. $f:\rho' \rightarrow \tau'$ shares the same function implementation with $f:\rho \rightarrow \tau$.

EXAMPLE 3.3: In Example 2.3, a,b are integer constants, let the production volume be an integer, then Equation 2.13 becomes:

f:INTEGER→INTEGER;
f=λx.(a+bx)

which denotes a mapping, INTEGER→INTEGER, and is a specialized function of f:REAL→REAL in Equation 2.13.

Then, an inheritance relationship among classes in model management can be derived:

RULE 4: For a given class C = { e_1, e_2, ..., e_n }, and a new class C' = { e'_1, e'_2, ..., e'_m }, where m≥n. If e'_i ($\exists i(1 \le i \le n)$) is a specialized function of corresponding e_i in class C, C' is then a subclass of C.

EXAMPLE 3.4: The integer programming (IP) model can be specified as:

IP = { c:R_n;
A:R_n^n;
b:R_n;
GOAL:$((R_n^2 \times R_n \to R_n) \to IR_n) \to$ REAL;
GOAL=min { cX where X:IR_n=A^{-1}b};
/* combine simplex method with branch&bound algorithm
SOLUTIONs:R_n;
SOLUTIONs=λX;
/* print standard solution report
RANGE:$IR_n \to IR_n$;
RANGE=λX.(X)(A^{-1}·b) where GOAL*=cX;
/* to test the range of X, while still keep the optimal result
},

where IP.SOLUTIONS and IP.RANGE are specialized functions of LP.SOLUTIONS and LP.RANGE, respectively. IP.GOAL is an incremental modification of LP.GOAL in class LP. Hence, IP is a subclass of LP, which can be abbreviated by:

IP={ GOAL:$((R_n^n \times R_n \to R_n) \to IR_n) \to$ REAL }
inherit from LP,

where GOAL is a composite function, which can not be simply inherited.

Different from the normalization theory in database design, the proposed inheritance rules are not sufficient for conceptual design in model management. This is because the decision model is much more complicated than data and its abstraction specification is determined by practical problem semantics. Hence, only some simple relationships for specialized functions can be automatically inferred by inheritance rules. At functional level, the type checking rules [4] for typed λ-calculus can also be used.

3.2 An Inheritance Rule Checking System

An inheritance rule checking system (IRCS) consisting of a lexical analyzer, a syntax analyzer and an inheritance rule checker has been designed and implemented in PASCAL to check the correctness of the framework and the applicability of inheritance rules for conceptual analysis in model management.

A lexical analyzer is a programmer scanner, which recognizes every element in the specification of decision models, returns a token when it is valid in the object-oriented framework, or otherwise, reports a lexical error. Some basic elements in the framework include identifiers (e.g. names of classes, functions and variables), operators, types and separators.

A syntax analyzer is a program grammar checker, which accepts model specification if it follows the grammar rules. For type checking purposes, we introduce ρ, σ, β as intermediate type variables, and modify grammar rules of Equation 2.12 to reduce ambiguity:

$$\tau ::= \rho / \tau \rightarrow \rho$$
$$\rho ::= \sigma / \rho \times \sigma$$
$$\sigma ::= \beta / (\tau)$$
$$\beta ::= \text{INTEGER} / \text{BOOL} / .... / R_n / IR_n / R_n^n \qquad (3.1)$$

With the grammar rules for the proposed framework (e.g. Equations 2.8 through 2.14, and 3.1), a syntax analyzer is implemented, the result being stored in the following data structure:

```
eclass, tclass: ARRAY[1..n] of RECORD
                    fname: string;
                    dmn: string;
                    cdmn: string;
                    status: string
                END;
```

where eclass, tclass, fname, dmn and cdmn stands for existing class, target class, function name, function domain and function codomain, respectively. status is a character bit to record the inheritance relationship between classes. status = 0, 1 and 2 means the function in eclass is nothing, a direct copy and a specialized function of corresponding function in tclass, respectively.

An inheritance rule checker is then a separate program to check if there exists an inheritance relationship between two classes. Its algorithm can be specified in Fig. 3.1.

```
i:=1;
WHILE eclass[i].fname<>' ' DO BEGIN
  j:=1;
  WHILE tclass[j].fname<>' ' DO BEGIN
     IF eclass[i].fname=tclass[j].fname THEN
        IF eclass[i].dmn=tclass[j].dmn THEN
           IF (eclass[i].cdm=tclass[j].cdm) and (eclass[i].status=0) THEN BEGIN
              eclass[i].status:=1;
              tclass[j].status:=1 END
           ELSE IF (eclass[i].cdm⊃tclass[j].cdm) and (eclass[i].status=0) THEN BEGIN
                 eclass[i].status:=2;
                 tclass[j].status:=2 END
        ELSE IF (eclass[i].dmn⊃tclass[j].cdm) and (eclass[i].status=0) THEN BEGIN
              eclass[i].status:=2;
              tclass[j].status:=2 END;
     j:=j+1
  END;
  i:=i+1
END;
```

Fig. 3.1: An algorithm for inheritance rule checking

IV. Conclusions

A formal object-oriented framework consisting of a conceptual structure and its manipulation operations has been proposed for model management in DSS. The conceptual structure which emphasizes data abstraction, information hiding and inheritance provides many advantages over current approaches to model management.

Since function abstractions are the only elements used in the proposed framework, it is possible to analyze and propose the rules for specialized functions and inheritance rules for classes in order to reduce redundancy and avoid inconsistency in the conceptual design of model management.

Based on the proposed framework and inheritance rules, an automatic inheritance rule checking system has been designed and implemented to check the correctness of model specification and to discover the relationship between model classes.

The proposed framework aims at specifying decision models in operations research or management science; however, it may also be useful for more general disciplines.

References

[1] 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., 1988, 156-169.

[2] R. W. Blanning: "An Entity-Relationship Approach to Model Management." *Decision Support Systems*, Vol. 2, North-Holland, 1986, 65-72.

[3] L. Cardelli: *Semantics of Multiple Inheritance.* Research Paper, 1988.

[4] L. Cardelli & P.Wegner: "Understanding types, Data Abstraction and Polymorphism." *Computing Survey*, vol. 17, no. 4, December 1985, 471-522.

[5] C. J. Date: *An Introduction to Database Systems*, 4th edition, Addison-Wesley Publishing Co. Inc. 1986.

[6] D. R. Dolk: "Data as Models: An approach to Implementing Model Management." *Decision Support Systems*, Vol. 2, North-Holland, 1986, 73-80.

[7] S. I. Gass: *Decision Making, Models and Algorithms.* John Willey & Sons, Inc. 1985.

[8] A. M. Geoferion: "An Introduction to Structured Modeling." *Management Science*, Vol. 33, No. 5, May, 1987, 547-588.

[9] A. Kaufmann & M. M. Gupta: *Fuzzy Mathematical Models in Engineering and Management Science*, North-Holland, 1988.

[10] H. R. Lewis & C. H. Papadimitriou: *Elements of the Theory of Computation.* Prentice-Hall, Inc., 1981.

[11] A. Lloyd: *A practical introduction to denotational semantics.* Cambridge University Press, 1986.

[12] R. E. Trueman: *Quantitative Methods for Decision Making in Business.* CBC College Publishing, 1981.