December 1993

Object-Oriented Design of Model Management Systems

Jian Ma
University of NSW

Vilas Wuwongse
Asian Institute of Technology

Object-Oriented Design of Model Management Systems

Jian Ma, School of Computer Science and Engineering, University of NSW, Australia

Vilas Wuwongse, Division of Computer Science, Asian institute of Technology, Thailand

Abstract

Model Management System (MMS) is a major component of a decision support system (DSS). Currently, several approaches have been proposed for the conceptual design of MMSs. However, these methods neglect to specifying the inheritance relationship among decision models. As a result, redundancy and inconsistency arise in MMSs. The objectives of this paper are to propose an object-oriented (OO) framework for the conceptual representation of decision models and to discuss the use of inheritance theory in OO research for the design of MMSs.

I. Introduction

By definition, a DSS is a software system designed to support semistructured or unstructured decisions in order to improve the effectiveness of decision-making (Blanning, 1986). One of the most commonly accepted frameworks for DSS (Sprague, 1989) is to classify it as comprising of three sets of capabilities: data management capability, model management capability, and the management capability for an interface between users and the system.

Data management is thought of as a well established research area because of the maturing technology in database modeling, database design and database management systems (DBMS). However, model management is a relatively new field of research. Several approaches have been proposed for the specification of decision models at the conceptual level, e.g. the relational approach (Blanning, 1989), which views a decision model as a virtual relation between input and output domains; the entity-relationship (E-R) approach (Chen, 1988), which treats a decision model as an entity comprising a number of attributes, and an interface among entities as a relationship. These two approaches make use of the existing methods, but fail to wholly support the reuse of decision models; whereas the model abstraction approach (Dolk, 1986) adapts the concept of data abstraction in modern programming languages and encapsulates data objects, procedures and assertions within a model abstraction. It separates the function specifications from their implementations in a decision model, but the relationships among decision models are not well specified. The structured modeling approach (Dolk, 1988) employs a hierarchically organized, partitioned and attributed acyclic graph to represent relationships in decision models. However, the resultant graph does not correspond to a real decision problem. The graph-based approach (Liang, 1986) views a model as a graph consisting of nodes representing data attributes and edges representing the functions that connect the input data to their associated outputs. The graph-based approach abstracts well the properties of a decision model at the conceptual level and is convenient to use. But the relationship between two models are not well specified. MMSs have been designed and implemented based on these methods, but they lack the design theory to discover the inheritance relationship for the reuse of decision models. As a result, redundancy and inconsistency arise in MMSs.

This paper presents an OO framework for the conceptual specification of decision models and discusses the use of inheritance theory in OO research for the conceptual design of MMSs. The proposed OO approach for model management emphasizes the concepts of data abstraction, information hiding and inheritance. It makes it possible to apply inheritance theory for the conceptual design in order to reduce redundancy and avoid inconsistency.

II. An Object-oriented Framework for Model Management Systems

In the technical literature (Meyer, 1988; Danforth and Tomlinson, 1988), several definitions are given of what OO means. In this paper, only the concepts of data abstraction, information hiding and inheritance are considered as essential characteristics.

Since the objective of this paper is to design MMSs at a conceptual level, λ-calculus (Lloyd, 1986; Barendregt, 1984) is used as a formal language to specify attributes and functions of a decision model. Typed λ-calculus is a formal system for the study of functions, their definitions and applications.

The basic typed λ-calculus has the syntax:

$$\begin{array}{ll} e ::= e(e) & \text{/* application} \\ \quad \lambda x{:}\tau.e(x){:}\tau & \text{/* abstraction} \\ \quad x{:}\tau & \text{/* typed variable} \\ \quad c{:}\tau & \text{/* typed constant} \end{array} \qquad (1)$$

where τ is a type with the syntax:

$$\tau ::= \text{INTEGER/BOOLEAN/CHARACTER/REAL} \\ /..../\tau \times \tau/\tau \rightarrow \tau/(\tau). \qquad (2)$$

Typed expression (thereafter abbreviated as expression) e is a typed function used to represent basic operations in a decision model. For simplicity, we use $f:\tau \rightarrow \tau$; $f=\lambda x.e(x)$ to represent function abstraction $f=\lambda x:\tau.e(x):\tau$. As a shorthand, we also allow "e' where $x:\tau= e$" to represent function abstraction $\lambda x:\tau.(e')(e)$.

In its purest form, typed λ-calculus does not have built-in functions (e.g., +) and composite types (e.g., ARRAY). But our intentions are practical, so we extend the typed λ-calculus to support arithmetical operations on the array type, i.e., A^{-1}, A^{T}, and A+B stand for the inversion, transposition and addition of the matrix respectively.

EXAMPLE 1: A typical linear decision model (or equation) is defined as:

$$f:\mathrm{REAL}\rightarrow\mathrm{REAL}; \quad f=\lambda x.(a+bx). \tag{3}$$

where x is a variable bounded by the λx with type REAL, a,b are constants, and λx.(a+bx) is a function abstraction with type REAL.

In order to allow various users to use MMSs conveniently, relevant functions of a decision model should be specified together based on a common data structure. This ensures the quality of decision model specification. In the literature of object-orientation (Cardelli and Wegner, 1985), record structure has been used to specify the concept of data abstraction and information hiding, where each field is an expression represented in typed λ-calculus.Thus:

DEFINITION 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:

$$C = \{e_1, e_2, \ldots, e_n\}, \tag{4}$$

where C is the global name of the class and each $e_i (1 \le i \le n)$ is an expression local to C.

Class is the realization of data abstraction and information hiding concepts, within which expressions are unordered and not duplicated due to the fact that a class corresponds to a set.

EXAMPLE 2: Linear programming (LP) (Gass, 1985) is a basic decision model in operations research, where the simplex method algorithm is commonly used to compute its optimal solutions. In practice, sensitivity analysis always follows the optimal result. If we use GOAL to represent the function that performs the simplex method algorithm, SOLUTIONs to display solution reports and RANGE to do the sensitivity analysis, then the LP model is specified in the following class schema:

```
LP={        c:R_n;
                /* coefficient vector c
        A:R_n^n;
                /* constraint matrix A
        b:R_n;
                /*  resource vector b
        GOAL:R_n → REAL;
                GOAL=MIN{c^T*X where X:R_n=A^-1*b};
                /* use simplex method algorithm
                /* to solve the problem
        SOLUTIONs:R_n;
                SOLUTIONs=X;
                /* print standard solution report
        RANGE:R_n→R_n;
                RANGE=λX.X*A^-1*b)
                /* to test the range of X, while
                /* still keep the optimal result
        }.
```

where c, A and b are the cost coefficient vector, constraint matrix and resource vector respectively. The vectors have a n-ary REAL type R_n and the matrix n×n-ary REAL type R_n^n.

In this OO framework, a class schema specifies a general decision model, but its instance is represented by an object. An object denotes a special case of the model class. Being 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.

In OO programming, incremental modification is a technique to integrate a modifying function with an existing one by the operator "⊙". In typed λ-calculus, it is represented by:

$$sc = \lambda p, \lambda x.m \odot p(x) = \lambda p, \lambda x.\ m(p(x)) \tag{5}$$

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.

DEFINITION 2: If some expressions in a class $C'=\{e'_1, e'_2, \ldots, e'_m\}$ are either direct copies or incremental modifications of the corresponding expressions in a class $C=\{e_1, e_2, \ldots, e_n\}$ $(m \ge n)$, then C' is a subclass of C and C is a superclass of C'.

DEFINITION 3: Inheritance is a mechanism to allow a new class to be defined by inheriting function properties from existing ones. Formally, if $C'=\{e'_1, e'_2, \ldots, e'_m\}$ is a subclass of a set of classes $\{C_1, C_2, \ldots, C_n\}$ (where $C_i = \{e_{i1}, e_{i2}, \ldots, e_{ir_i}\}$, for any i in $1 \le i \le n$), then it can be denoted by:

$$C' = \{e'_{j1}, e'_{j2}, \ldots, e'_{jk}\} \quad \textit{inherit from } C_1, \ldots, C_n. \tag{6}$$

where each c'_{ji} ($1 \le j \le m$) is either an incremental modification of the corresponding expression in class C_i ($1 \le i \le n$) or an expression which never appears in its superclasses.

In this paper, every expression is defined to be local to a class. Therefore the same expression name may appear in more than one class. In order to avoid this ambiguity in the inheritance case, the expression name should be explicitly distinguished by concatenating the class name with "." and the expression (e.g. C.e) when necessary.

EXAMPLE 3: In the multi-objective linear programming model (MOLP), each objective function has a weight. One possible approach to determine the weights for each objective function is to use the fuzzy statistical method (FSM), which can be implemented in the class FSM below.

```
FSM={   Rwt:$R_n^m$;
            /* mxn-ary real matrix of
            /* the raw weights
        W:$R_n^m \rightarrow R_n$;
            W=$\lambda Rwt.\left(\left(\sum_{j=1}^{m} Rwt_{ij}\right)/m\right)$;

            /* evaluate proper weights for
            /* each objective function
     };
```

where Rwt is a matrix storing raw weights and W is a function to evaluate raw weights for different objective functions.

With the proper weights for corresponding objective functions, MOLP model can then be specified by inheriting function properties from class LP and FSM:

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

where cc is a new function for the input of cost coefficient matrix and wc assigns weights to the corresponding cost coefficient vector.

Inheritance specifies the relationship for the reuse of decision models, which helps to reduce redundancy and avoid inconsistency among classes in designing a MMS.

III. An Object-Oriented Design Theory for Model Management Systems

In the proposed OO framework, a decision model is specified in the form of a class and the inheritance mechanism allows a new class to be specified by making use of properties in existing classes. However, in designing a MMS using OO approach, we still don't know what classes are needed and what the inheritance relationships among them will be. This chapter answers the question in two separate sections: section 3.1 proposes the normalization of a single class; and section 3.2 presents the inheritance theory for the specialization functions between two classes.

3.1 Normalization of a class

This section first reviews conversion rules (Lloyd, 1986), so as to build the design theory for an individual class design.

The simplification of λ-expressions is governed by the rules of conversion (CONV). There are three conversion rules, namely α, β, η, as they are defined below:

η-Conversion (CONVη) allows any expression to be treated as a function:

$$\eta{:}-\quad e \equiv \lambda x.e(x) \tag{7}$$

where x is not free in e to prevent variable capture. An application of η-conversion from right to left simplifies an expression and is called an η-reduction (REDη).

EXAMPLE 4: A linear expression can be treated as a function by η-conversion rule:

$$a+bx \equiv \lambda x.\mathrm{ADD}(a, \mathrm{MULT}(b, x))$$

α-Conversion (CONVα) is a rule for the systematic renaming of formal parameters:

$$\alpha{:}-\text{ if y is not free in e, } \lambda x.e \equiv \lambda y.e[y/x] \tag{8}$$

where y must not be free in e to prevent variable capture. Parameter names in an expression are 'dummy' names and can be changed consistently.

EXAMPLE 5: If we rename x by y in example 3.1, the expression becomes:

$$f=\lambda x.\mathrm{ADD}(a, \mathrm{MULT}(b, x))=\lambda y.\mathrm{ADD}(a, \mathrm{MULT}(b, y))$$

β-Conversion (CONVβ) is the rule for function application. It defines the substitution of an actual parameter for the formal parameter.

$$\beta{:}-\ ((\lambda x.e)a) \equiv e[a/x] \tag{9}$$

EXAMPLE 6: If we let the value of x be 10, the above expression becomes:

$$f=\lambda x.\mathrm{ADD}(a, \mathrm{MULT}(b, x))10$$
$$=\mathrm{ADD}(a, \mathrm{MULT}(b, 10))$$

where a and b are constant coefficients.

The application of β-conversion from left to right simplifies an expression and is called β-reduction (REDβ).

The conversion rules allow expressions to be reduced (RED) or evaluated. In general,

DEFINITION 4: An expression that can not be reduced further is in normal form. Formally, if e CONV e' and e' is in normal form, then e' is a normal form of e.

However, it is not always possible to reduce an expression to a normal form. In practice, there are many possible reduction sequences that can be applied to an expression to reduce it to the normal form. This raises the possibility that two different sequences may lead to different normal forms for the same expression. Fortunately, there is a theorem, the first Church Rosser Theorem, which guarantees that if an expression reduces to two normal forms, they must be inter-convertible using an α-conversion. Formally,

THEOREM 1 (Church Rosser Theorem I): If e_1 CONV e_2, then there exists an expression e such that e_1 RED e and e_2 RED e.

One of the consequences of the first Church Rosser Theorem is that the normal form of an expression is unique. However, if we reduce the expressions in an arbitrary order, we may produce a nonterminating sequence of reductions. Luckily, there is a second Church Rosser Theorem which tells us that if we always reduce the leftmost expression first, the reduction sequence will terminate with a normal form, if it exists.

THEOREM 2 (Church Rosser Theorem II): If e RED e' and e' is in normal form, then there exists a reduction sequence from e to e', which involves successively reducing the leftmost expressions.

This order of reduction is called the normal order reduction. Examples and detailed proofs of Theorems 1 and 2 can be found in Barendregt's (1984) book.

On the basis of the above theorems for the reduction of expressions, we can now develop the normalization theory for an individual class design.

DEFINITION 5: Expression e_1 is equivalent to e_2 if and only if there exists a normal form e, such that e_1 CONV e and e_2 CONV e.

If e_2 CONV e, by Theorem 3.2, there exists a leftmost reduction sequence for e_2 CONV e such that e CONV e_2 (conversions are reversible). Therefore,

$$e_1 \text{ CONV } e \text{ CONV } e_2.$$

In other words, the statement e_1 is equivalent to e_2 means that e_1 can be reduced to e_2 in some finite number of steps by the conversion rules.

DEFINITION 6: A class is in its first normal form if and only if each of its expressions is in normal form and its two expressions are not equivalent.

In proposed OO framework, a decision model is represented in the form of a class which groups relevant expressions together on a common data structure. Thus the simplification of a class depends on both the reduction of expressions and the elimination of redundant expressions within a class.

If a class is in its first normal form, then no redundancy or inconsistency exists within a class, which can be used as a result to specify a decision model.

The first normal form of a class emphasizes the simplification of expressions within a class. However, it does not consider the specification of relationships for reuse between classes.

3.2 Inheritance Rules between Classes

If a class is in its first normal form, then it is the efficient form for the specification of a decision model. Now suppose every class is in its first normal form (by leftmost reduction), one of the major relationships for the reuse of decision models is governed by inheritance rules for specialization functions between classes.

It will be recalled that in the proposed OO framework, most of the expressions are typed function abstractions with the form: $f:\rho \rightarrow \tau$, for $x:\rho$ and $e(x):\tau$. Within this functional framework, some rules for functions can be derived:

DEFINITION 7: If there exist domains $\rho \supset \rho'$ and the function $f:\rho \rightarrow \tau$, then $f:\rho' \rightarrow \tau$ is a specialized function of $f:\rho \rightarrow \tau$.

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

DEFINITION 8: If there exist domains $\tau \supset \tau'$ and the function $f:\rho \rightarrow \tau$, such that $f:\rho \rightarrow \tau'$ is meaningful, then $f:\rho \rightarrow \tau'$ is a specialized function of $f:\rho \rightarrow \tau$.

The condition that $f:\rho \rightarrow \tau'$ is meaningful means that for any value $x \in \rho$, there exists $f(x) \in \tau'$. Then the mathematical explanation of the definition can be stated as: given any value $x \in \rho$, there exists $f(x) \in \tau' \subset \tau$. Thus $f:\rho \rightarrow \tau'$ shares the same function implementation with $f:\rho \rightarrow \tau$.

EXAMPLE 7: The stock cutting (SC) problem can be modelled in an integer programming model, 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 IPGOAL:$((R_n^n \times R_n \times R_n \rightarrow R_n) \rightarrow IR_n) \rightarrow$ REAL
in class IP:

```
IPGOAL:$(R_n \rightarrow IR_n) \rightarrow$ INTEGER;
   IPGOAL=MIN {MMLT(MTNS(c), Y) where
          Y=MINT(X) where X=MMLT(MINV(A), b)}.
```

where IPGOAL 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.

DEFINITION 9: If there exist domains $\tau \supset \tau', \rho \supset \rho'$ and the function $f{:}\rho \rightarrow \tau$, such that $f{:}\rho' \rightarrow \tau'$ is meaningful, then $f{:}\rho' \rightarrow \tau'$ is a specialized function of $f{:}\rho \rightarrow \tau$.

This definition is in fact the combination of definitions 7 and 8. As a result, $f{:}\rho' \rightarrow \tau'$ shares the same function implementation with $f{:}\rho \rightarrow \tau$.

EXAMPLE 8: In example 1, a and b are integer constants. If we let the value of x be an integer, the expression becomes:

```
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 specialized functions. Therefore the definitions of inheritance can be further explained by:

DEFINITION 10: Given a class $C = \{ e_1, e_2, \ldots, e_n \}$, and a new class $C' = \{ e'_1, e'_2, \ldots, e'_m \}$, where $m \geq n$, if expression e'_i (for some i in $(1 \leq i \leq n)$) is a specialized function of the corresponding expression e_i in class C, C' is then a subclass of C, which inherits functions from its super-class C. Formally, we write

$C' = \{e'_{j1}, \ldots, e'_{jk}\}$
inherit from C.,

where e'_{jl} $(1 \leq jl \leq m)$ is not a specialized function of its corresponding expression in superclass C.

EXAMPLE 9: Since the computation of integer programming (IP) problem is to use simplex method and then branch&bound method for integer optimal result, it is the incremental modification of the above simplex method. Thus class IP can be specified by inheriting function properties from LP.

IP={ IPGOAL:$(R_n \rightarrow IR_n) \rightarrow$ REAL }
inherit from LP.,

where IPGOAL is a composite function, which calls the simplex method algorithm in class LP, and then executes the branch&bound method to generate integer optimal results.

Inheritance rules help to discover the specialized functions in order to reduce redundancy between two classes. Thus in the conceptual analysis of model management, if more than one decision model is concerned, we have

DEFINITION 11: A class is in second normal form if and only if it is in the first normal form, and it does not contain specialized functions in the existing class.

In contrast to the first normal form, the second normal form of a class emphasizes the reduction of specialized functions by inheriting from its superclasses.

IV Conclusion

An OO framework has been proposed and an OO design theory consisting of the normalization of a class and the inheritance rules between two classes has been presented. The proposed OO approach to model management emphasizes the concepts of data abstraction, information and inheritance. It provides a consistent and natural decomposition of decision models and promotes software reuse in the conceptual design of MMSs.

References

Aho, A. V., Sethi R. and Ullman J. D., 1986, Compilers: Principles, Techniques, and Tools, Addison-Wesley Publishing Company.

Barendregt, H. P., 1984, The Lambda Calculus: Its Syntax and Semantics, Horth-Holland Publishing Company, Amsterdam, New York, Oxford.

Blanning, R. W., 1986, An Entity-Relationship Approach to Model Management, Decision Support Systems, Vol. 2, North-Holland, pp. 65-72.

Blanning, R. W., 1989, 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.

Cardelli, L. and Wegner, P., 1985, Understanding types, Data Abstraction and Polymorphism, Computing Survey, vol. 17, no. 4, December, pp. 471-522.

Chen, Y. S., 1988, An Entity-Relationship Approach to Decision Support and Export Systems, Decision Support Systems, 4, North-Holland, pp. 225-234.

Danforth, S. and Tomlinson, C., 1988, Type Theories and Object-Oriented Programming, ACM Computing Survey, Vol. 20, No. 1, pp. 29-72.

Dolk, D. R., 1986, A Generalized Model Management System for Mathematical Programming, ACM Transactions on Mathematical Software, Vol. 12, No. 2. pp. 92-126.

Dolk, D. R., 1988, Model Management and Structured Modeling: The Role of an Information Resource Dictionary System, Communication of the ACM, Vol. 31. No. 6. pp. 704-718.

Gass, S. I., 1985, Decision Making, Models and Algorithms, John Willey & Sons, Inc.

Geoferion, A. M., 1987, An Introduction to Structured Modeling, Management Science, Vol. 33, No. 5, May, pp. 547-588.

Liang, T. P., 1986, A Graph-based Approach to Model Management, *Proceedings of sixth International Conference on Information Systems*, San Diego, pp. 136-151.

Lloyd, A., 1986, A practical introduction to denotational semantics, Cambridge University Press.

Meyer, B., 1988, Object-oriented Software Construction, Prentice Hall, UK.

Spague, R. H. Jr., 1989, A Framework for the Development of Decision Support Systems, Decision Support Systems: Putting Theory into Practice, 2nd edition, Prentice-Hall, pp. 9-35.