Extensions to the Object-Oriented Paradigm

Stanley Y. W. Su

Database Systems Research & Development Center
University of Florida

1. What does the Object-oriented Paradigm Offer?

The object-oriented paradigm, which was made popular by the programming language Smalltalk, provides many important concepts and features for the development of application systems and database management systems (DBMSs). They include: object identity, abstract data type, information hiding, user-defined operations, polymorphism, inheritance, reusable codes, object communication by message passing, and uniform representation of data and metadata. Among them, two features, namely abstract data typing and user-defined operations, have changed the traditional view of the relationship between applications and data in database management. Other concepts and features are used to facilitate this change.

In the traditional data management systems (DBMSs and file management systems), there is a clear separation between application systems and data. A database is a collection of recorded facts used by many application systems developed to support the operation of an enterprise. Control logics and decision rules necessary to carry out the operation are coded in application programs. Data are shared by the application systems but not the control logics and decision rules, which are embedded in the application programs. Consequently, the semantics represented by these logics and rules are often replicated in various application systems. The current commercially available object-oriented DBMSs such as Gemstone and Vbase allow new data types to be defined in terms of system-defined and/or user-defined data types. Furthermore, meaningful operations that can be performed on the objects of these new data types can be declared, and the programs that implement these operations (i.e., methods) can be reused by various applications. In essence, the control logics and decision rules that implement these operations are stored as a part of the database for use by the applications. This simplifies the application development to a great extent since some of the control logics and decision rules are encapsulated in the operations associated with object types. In addition to the static data associated with objects, some behavioral properties of these objects have been captured by the system-defined and user-defined operations. This is certainly a step toward a more intelligent and powerful database management system.

2. Some Extensions

It is this author's belief that the present O-O DBMSs have not gone far enough. Two important aspects of semantics are not captured in the Smalltalk-type O-O paradigm and the DBMSs that have been developed on the basis of this paradigm. The first aspect is the specification of knowledge rules which model constraints, expert knowledge, deductive rules and trigger conditions that are applicable to objects. The second aspect is the specification of various types of associations that an object type can have with other object types. Both aspects need to be incorporated into the O-O paradigm of the future DBMSs or knowledge base management systems (KBMSs) to give the needed expressive power for modeling complex objects found in more advanced application areas such as CAD/CAM, CIM, CASE, and office automation.

Also, the O-O paradigm for future DBMSs or KBMSs should extend the concept of a class from that of "object type" to "object class". In other words, the Smalltalk class definition specifies the properties of instances which can be instantiated for an object type. Once objects are instantiated, the relationship among objects of the same class cannot be easily identified. The concept of an object class containing a set of instances, which is important for finding all instances of a given class, is not supported by the typing system of the existing O-O paradigm. The concept of object class is an extension of the data typing concept. It serves two purposes: type declaration and class membership which are important for data/knowledge base applications.

An extended O-O paradigm should have a semantically rich typing system. An object class specification should include the following information:

```
CLASS Name (Parameters)
Association Declaration
Method Declaration
Rule Declaration
END_CLASS
```

The class header gives the name of the class and parameters for a generic class declaration. The method declaration is the same as the conventional O-O systems. We elaborate on the rule and association declarations below:

0730-3157/89/0000/0197$01.00 © 1989 IEEE

Rule Declaration and Rule Inheritance

The rule declaration specifies the various types of constraints and deductive rules that can be applied to the objects and the triggering conditions under which these constraints and rules should be verified and applied. A simple class specification is given below for illustration purposes.

```
CLASS X
  ASSOCIATION_SECTION
    AGGREGATION OF
      AT1: INTEGER;
      AT2: INTEGER;
      AT3: INTEGER;
  END_ASSOCIATION;
  RULE_SECTION
    RULE R1
      TRIGGER_CONDITION (AFTER UPDATE (AT1,
                                 AT2,AT3));
      AT1 = AT2 + AT3;
      CORRECTIVE_ACTION (UPDATE (AT2));
    RULE R2
      TRIGGER_CONDITION (CREATE (OBJECT OF X));
      IFSUM(AT1)>10000THEN(UPDATE(Y.ATTR),
        MESSAGE('TOTAL AT1 IS OUT OF BOUND'));
  END_RULE_SECTION;
END_CLASS;
```

The object class X has three attributes (or instance variables) AT1, AT2 and AT3. Attributes of a class is specified here as one type of association, namely Aggregation. R1 is the name of the first rule. Its trigger condition specifies that after updating any of the three attributes, the time-invariant or static constraint AT1=AT2+AT3 must be satisfied. In the case of a violation, the corrective action that the system should take is to automatically update AT2 to satisfy the constraint. Constraints or deductive rules triggered under one or several predicates expressed as a Boolean expression can be performed before, after or in parallel with the trigger condition(s). The corrective action can be optionally specified by the user or can be a default system action. The second rule R2 illustrates that (1) user-defined operations (e.g., SUM) can be used in constraints and rules, (2) multiple actions can be taken if the predicate of the left-hand side is true, and (3) the action taken may involve an operation on another object class (Y) which may trigger other operations and/or rules.

In addition to predicates that make reference to attributes and values, the constraint or rule specification in a class declaration may involve an association pattern. For example,

```
RULE 3
IF X * Y ! Z  THEN EXECUTE R2;
```

states that if any object of type X is associated with any object of type Y and the Y object in the associated pair is not connected to any object of type Z, then rule R2 should be evaluated. This example shows that complex object association patterns (patterns involving the association operator *, non-association operator !, as well as AND and OR branches connecting object types) can be used to specify the condition of a constraint or a deductive rule. A pattern specification can also occur on the right side of a rule. In that case, a sub-knowledge base, which contains those objects of the involved classes that satisfy the specified pattern, can be generated by the rule. This pattern specification capability can greatly enhance the expressive power of a knowledge specification language of an extended O-O paradigm.

Knowledge rules can be used to specify non-deterministic and random conditions found in many real-world applications. For example, in the rule shown below,

```
RULE R4
IF X THEN (Y OR Z);
```

Either Y or Z can be true if X is true. Here, the truth value of Y or Z can be determined by a user or an external condition. Also, a random function can be introduced in rules to allow actions to be taken at random as illustrated by the following example:

```
RULE R5
IF X THEN Z= RANDOM(Y);
IF  Z <  0.5 THEN ACTIVATE action_1;
IF  Z >= 0.5 THEN ACTIVATE action_2;
```

The use of non-determinism and randomfunction in rules has been introduced in the IC* model developed by Bell Communication Research [CAM88]. This should be incorporated in a knowledge rule specification language.

In the proposed typing system, constraints and rules associated with superclasses can be inherited by subclasses just like the inheritance of attributes and methods. This allows the behavioral properties of an object class defined by constraints and rules to be adopted by any new subclass without re-specification. The problem of multiple inheritance may occur when a subclass has multiple superclasses and the inherited semantic properties (attributes, associations, methods and knowledge rules) from the superclasses have a conflict. A variety of conflict resolution approaches can be taken. One approach is to select a property from a number of alternatives based on a rule. Thus, the integration of knowledge rules in object class specifications makes this approach of handling the multiple inheritance problem a possibility. Another significant advantage of the integration is that constraints and rules are naturally distributed among object classes based on their semantic properties. When the objects of these classes are accessed, constraints and rules that are relevant to these objects are readily available. This is different from the interface or bridging approach used in many expert database systems in which a database management system is loosely coupled with an inference engine. The different knowledge representations in the two components and the frequent data exchange between them during an inferencing process make this type of systems rather inefficient.

Association Types

The association declaration in an object class specifies the various kinds of associations that an object class has with other object classes. The instances of this object class have the same associations with the instances of the other object classes. Many association types have been introduced in the existing semantic data models. For example, the Aggregation and Generalization of Smith and Smith, the Relationship of Chen, and the Interaction, Crossproduct and Composition of our own *OSAM* model are some examples.* These association types provide the user with high-level modeling constructs to express different semantic relationships among object classes and their instances. For example, the following declaration specifies that class X is described by an aggregation of two attributes AT1 and AT2. The type of AT2 is DATE which is defined elsewhere. X is a superclass of class Y (the generalization association). The interaction association models the fact that an object of class X represents an interaction (or relationships) between an object of class Z and an object of class W. The cardinality constraint of this interaction is many-to-many, meaning that one Z object can pair with many W objects and one W object can pair with many Z objects. The instances of X are represented by pairs of Z and W objects and the attributes AT1 and AT2 describe their interactions rather than describing the individual participating objects. Class Q is a subclass of class W.

```
CLASS X
   ASSOCIATION_SECTION
    AGGREGATION OF  AT1:REAL;
                    AT2:DATE;
    GENERALIZATION OF Y
    INTERACTION OF (Z,W) (many-to-many)
   END_ASSOCIATION;
END_ClASS;
CLASS Y
   ASSOCIATION_SECTION
    AGGREGATION OF  AT3:INTEGER;
   END_ASSOCIATION;
END_CLASS;
CLASS Z
   ASSOCIATION_SECTION
    AGGREGATION OF AT4:INTEGER;
                   AT5:SET OF INTEGER;
   END_ASSOCIATION;
END_CLASS;
CLASS W
   ASSOCIATION_SECTION
    AGGREGATION OF  AT6: LIST OF REAL;
    GENERALIZATION OF Q
   END_ASSOCIATION;
END_CLASS;
CLASS Q
   ASSOCIATION_SECTION
    AGGREGATION OF AT7: INTEGER;
   END_ASSOCIATION;
 END_CLASS;
```

An O-O system should also allow user-defined association types to fit different application's modeling needs. It should be noted that the semantics of the association types shown above and other user-defined association types can be defined in terms of knowledge rules governing the retrieval, update, insertion and deletion operations performed on the object classes having the association types. Since they are semantic properties that are commonly recognized and used in database modeling, it is convenient to distinguish them structurally by association declarations rather than by knowledge rules. In an underlying computation model for implementing a high level semantic model, knowledge rules that capture these association types can be defined in separate object classes and can be inherited by the object classes that enter into the corresponding associations with other object classes.

In summary, I would like to suggest that future research should be toward O-O knowledge base management rather than O-O database management. An O-O KBMS should allow "knowledge" of the real-world that is useful to an enterprise to be explicitly defined in terms of: (1) structural properties, i.e., system-predefined and user-defined associations with other object classes, (2) system-predefined or user-defined operations and their implementations (the methods), and (3) knowledge rules. Thus, data (in the traditional sense of the term) as well as operations and knowledge rules can all be stored in a knowledge base and be shared by different applications. Therefore, what is shared is no longer just data but knowledge. This will allow any application system that uses a KBMS to behave more intelligently. The semantics (i.e., control logics and decision rules) that are captured in application programs can be migrated to the knowledge base if their use by other applications is needed. The traditional separation of data from applications is a concept of the past.