*Appl. Math. and Comp. Sci., 1992, vol.2, No.1, 49–63*

# A FORMAL DATA MODEL AND OBJECT ALGEBRA FOR OBJECT–ORIENTED DATABASES

REDA AL–HAJJ*, M. EROL ARKUN*

In this paper we describe a formal object–oriented data model and a corresponding object–algebra. Described in the literature are two trends in query support for object–oriented databases. The first trend is to preserve existing objects, while the other trend allows for the creation of new objects. The object algebra presented in this paper is of the second type and includes operators of the relational algebra in addition to some others, but the semantics of all those operations has been specified to accomodate to the object–oriented features. An operand should have a defined set of objects and a set of message expressions. A message expression is sequence of messages. Also the result of any operation is defined to have the two sets and therefore, can be an operand. Hence the closure property is satisfied. Finally we include some of the properties of the described object algebra operators.

## 1. Introduction

In object–oriented systems (Goldberg and Robson, 1983; Stefik and Bobrow, 1986), an entity of the real world is modeled by an object that captures and encapsulates both the state and behavior of the represented entity. Due to encapsulation, objects are treated using messages to invoke corresponding methods in the behavior. An object is distinguished by a single identity that provides for independence and supports object sharing (Khoshafian and Copeland, 1986). Objects are grouped into classes according to their common state structure and behavior. Furthermore, classes are arranged in a hierarchy or lattice to overcome duplication and provide for reusability. In this sense, a class is allowed to have one or more superclasses and one or more subclasses.

It was recognized that the conventional relational model does not satisfy the requirements of data intensive applications such as CAD/CAM, OIS and AI. Hence extensions to the relational model were proposed. The introduction of set valued attributes was followed by the arbitrary nesting of tuples and sets as complex objects (Abiteboul *et. al.*, 1989). Next complex objects were combined with object identity (Abiteboul and Kanellakis, 1989; Bancilhon *et. al.*, 1987) in a step towards the combination of object–oriented features with the database technology (Atkinson *et. al.*, 1989; Kim, 1990).

* Bilkent University, Faculty of Engineering, Depart. of Comp. Engng. and Inform. Sci., Bilkent 06533, Ankara – TURKEY

A number of object-oriented data models have been proposed (Banerjee *et. al.*, 1987; Beeri, 1989; Carey and DeWitt, 1986; Deux *et. al.*, 1990; Fishman *et al.*, 1987; Maier and Stein, 1987), but still there is no agreement on a standard object-oriented data model. In addition there is no standard object-oriented query language. Generally speaking, there are two trends in providing query support for object-oriented databases. The first trend preserves existing objects (Alashqur *et. al.*, 1989; Banerjee *et. al.*, 1988; Carey *et. al.*, 1988; Straube, 1991), while the second trend adds the possibility of creating new objects (Cluet *et. al.*; Dayal, 1989; Guo *et. al.*, 1991; Kim, 1989; Osborn, 1988; Shaw and Zdonik 1989). The reason behind the first trend is that there is no need to have new relationships added to the model as all the required relationships are specified at the modeling level and hence only object retrievals need to be supported. But, it is not possible to have all the relationships predefined, or to restrict possible relationships to those predefined. Hence, operators that build new relationships should be supported.

There are some points to consider while allowing for the creation of new objects. One point is to allow the output of an operation to be an operand, i.e., to satisfy the closure property. Another point is related to the characteristics of the output of an operation and whether it is a class whose proper place in the lattice should be determined.

Concerning the closure property, we say that the set of natural numbers $N$ is closed with respect to addition and multiplication but neither subtraction nor division. That is, $\forall x, y \in N$, it is for sure that $(x+y) \in N$ and $(xy) \in N$, but it is not guaranteed that the subtraction of any two elements of $N$ to be an element of $N$, i.e., $\forall x, y \in N$, $x < y \Rightarrow (x-y) < 0$, not an element of $N$. When applying the same concept to the relational model, elements of the relational model are relations and the allowed operations are those of the relational algebra (Date, 1986). The relational model satisfies the closure property with respect to the relation algebra operations and the result of any operation is a relation. Concerning an object-oriented model the closure property requires that the result of a query operation to be used as an operand.

In this paper we describe an object algebra that handles the creation of new objects, i.e., the introduction of new relationships. Therefore, we should satisfy the closure property and place created objects in the lattice. While the closure property is treated in this paper, the placement in the lattice is left out as the subject of another paper in preparation. So in this paper we describe the object algebra and how the closure property is satisfied. But before doing that, we introduce our data model and the basic terminology used in formalizing the algebra.

In our object algebra, we support the operators of the relational algebra in addition to other operators, but with different semantics. The difference in semantics is due to the object-oriented features (Atkinson *et. al.*, 1989; Kim, 1990; Stefik and Bobrow, 1986) added to enrich the object-oriented data model. In our algebra, an operand should have a defined set of objects and a set of message expressions used in manipulating objects in the first set. A message expression is a sequence

of messages and returns some value from the receiving object. An operation in our algebra acts on its operand(s) to produce a result that has a defined set of objects and a set of message expressions and hence can be an operand. In addition to the relational operators, we introduce a nest operator that adds a required relationship to the model. Another operator, one level projection, forms new objects by evaluating some given message expressions on objects of the operand.

The rest of this paper is organized as follows. A formal description of the data model is given in section 2 where we define classes, objects and methods in addition to message expressions, predicate expressions and the terminology required in formalizing the algebra. In section 3, we formally define the object algebra together with some properties that may be considered crucial in the optimization. Section 4 includes related work and some conclusions.

## 2. Formal Description of the Data Model

Suppose that the following sets are given:

- a countably infinite set of unique identifiers $U_I = \{id_1, id_2...\}$,
- a countably infinite set of classes $C = \{c_1, c_2, ...\}$,
- a countably infinite set of objects $O = \{o_1, o_2, ...\}$,
- a countably infinite set of domains $D = \{d_1, d_2, ...\}$,
- a countably infinite set of instance variables $I_V = \{iv_1, iv_2, ...\}$,
- a countably infinite set of methods $M_T = \{mt_1, mt_2, ...\}$,
- a countably infinite set of messages $M_S = \{ms_1, ms_2, ...\}$, there is a mapping from $M_T$ to $M_S$, such that for every method there exists a corresponding message.

Having these sets, next we give formal definitions for classes, objects and methods.

### 2.1. Classes

Simply speaking, a class is a set of objects with the common state structure and behaviour definitions. In this section we give the formal definition of a class, define a partial ordering among classes and define the set of domains.

**Definition 2.1.** A class $c$ is a tuple $(S, I_v, M_t, U_i)$ where,

- $S \subset C$ is the set of superclasses of class $c$,
- $I_v \subset I_V$ is the set of instance variables of class $c$,

  $\forall iv_i \in I_v$, $\exists d_i \in D$ where $D$ is the set of domains to be defined later in this section, we denote $\text{domain}(iv_i) = d_i$,

  $\forall iv_i, iv_2 \in I_v$, $iv_1 \neq iv_2$, i.e., no two instance variables of the same class can have the same name.

- $M_t \subset M_T$ is the set of methods of the class (the method definition is given in section 2.3.).
- $U_i \subset O$ is the set of objects in $c$, not common with any of its subclasses. □

Related to a class we use the following notation:

- $supers(c) = S$, we enumerate elements of $S$ as $s_1, s_2, ....$
- $I_{variables}(c) = I_v \bigcup_{i=1}^{card(S)} I_{variables}(s_i)$, where $card(S)$ is used to denote the number of elements in the set $S$.
- $methods(c) = M_t \bigcup_{i=1}^{card(S)} methods(s_i)$,
- $instances(c) = U_i$.
- $T_{instances}(c) = instances(c) \bigcup_{i=1}^{card(C1)} T_{instances}(c1_i)$, where $C1$ is the set of direct subclasses of class $c$. This is due to substitutability where an instance of a class $c$ may be used whereever an instance of a superclass of class $c$ is required.
  In the rest of this paper, $c$ will be used in any context where $T_{instances}(c)$ is expected, distinction will be indicated only in case of confusion.

After the formal definition of a class, next we define a partial ordering $(\leq_c)$ among classes.

**Definition 2.2.** Given two classes $c_1$ and $c_2$, we say that $c_1 \leq_c c_2$ iff:

- $I_{variables}(c_2) \subseteq I_{variables}(c_1)$

  That is, $\forall iv \in I_{variables}(c_2) \exists iv_1 \in I_{variables}(c_1)$ such that,

  $$name(iv) = name(iv_1) \wedge (domain(iv_1) \leq_c domain(iv) \vee domain(iv) = domain(iv_1))$$

- $methods(c_2) \subseteq methods(c_1)$

$c_1 \leq_c c_2 \Leftrightarrow c_1$ is a subclass of $c_2$ and $c_2$ is a superclass of $c_1 \Leftrightarrow c_2 \in supers(c_1)$. □

Given two classes $c_1$ and $c_2$ such that $c_1 \leq_c c_2$, then all of the following are true:

- $c_2 \in supers(c_1)$
- $instances(c_1) \subseteq T_{instances}(c_1) \subseteq T_{instances}(c_2)$
- $I_{variables}(c_2) \subseteq I_{variables}(c_1)$
- $methods(c_2) \subseteq methods(c_1)$; this property reflects a Cardelli-like semantics of subtyping (Cardelli and Wegner, (1985).

Now we define the set of domains $D$ which is dependent on defined classes.

**Definition 2.3.** The set of domains $D$ is defined to include the following:

- The conventional domains, such as the set of integers, the set of reals, the set of characters, etc.

- for any class $c, T_{instances}(c) \in D$. □

Elements of $D$ are not disjoint. For instance, given that $c_1 \leq_c c_2 \Rightarrow T_{instances}(c_1) \subseteq T_{instances}(c_2)$, and given that $(c_1 \leq_c c_2 \wedge c_1 \leq_c c_3) \Rightarrow T_{instances}(c_1) \subseteq (T_{instances}(c_2) \bigcap T_{instances}(c_3))$

Given $c \in C$ and let $iv \in I_{variables}(c)$, $value(iv) \in domain(iv) \cup 2^{domain(iv)}$ where $value(iv)$ denotes the value of $iv$ and $2^X$ denotes the power set of the set $X$, i.e., set of proper subsets of $X$. Hence, $iv$ may have either a single value or a set value. Although in this paper a single value is mostly assumed, but this does not introduce any restriction.

### 2.2. Objects

In this section we define objects which are instances in classes. An object has identity and value and hence objects are equal either by value or by identity. Equality of objects is defined next in this section.

**Definition 2.4.** An object $o$ is a triplet $(id, c, v)$ where,

- $id \in U_I$ is the identity of the object,
- $c \in C$ is the class of the object,
- $v$ is the value of the object

  $$v \in X_{i=1}^{card(I_{variables}(c))} domain(I_{variables}(c)_i),$$

  where $X$ denotes the cross–product and $I_{variables}(c)_i$ is the $i$–th element in $I_{variables}(c)$. □

Concerning an object, we use the following notation:

$$identity(o) = id, \quad identity^{-1}(id) = o, \quad class(o) = c, \quad value(o) = v.$$

In the rest of this paper, $o$ will be used in any context where $identity(o)$ is expected, i.e., the $identity()$ function will be dropped in case of no confusion.

Let $p$ be a predicate expression, we use $p(o)$ to denote the application of the predicate expression $p$ to object $o$. Predicate expressions are defined in section 2.3.1.

Related to objects, the following holds by definition:

- $\forall o \in O, \quad o \in instances(class(o)) \Rightarrow o \in T_{instances}(class(o))$
- $\forall o_1, o_2 \in O, \quad identity(o_1) \neq identity(o_2) \Leftrightarrow (o_1$ and $o_2$ are not identical), i.e., $identity()$ is an injective function.

**Definition 2.5. Equality of objects.** Two objects $o_1$ and $o_2$ are:

- identical $(o_1 = o_2)$ iff $identity(o_1) = identity(o_2)$
- shallow–equal $(o_1 \doteq o_2)$ iff $value(o_1) = value(o_2)$
- deep–equal $(o_1 \cong o_2)$ iff by recursively replacing every $id \in U_I$ in $value(o_1)$ and $value(o_2)$ by $value(identity^{-1}(id))$ equal values are obtained. □

For instance, given objects $o_1$ and $o_2$ with $value(o_1) = (x, id_1, y)$, $value(o_2) = (x, id_2, y)$ and $value(identity^{-1}(id_1)) = value(identity^{-1}(id_2)) = (a, b)$; objects $o_1$ and $o_2$ are neither identical nor shallow–equal, but deep–equal because of the equality of their values after replacing $id_1$ in $o_1$ and $id_2$ in $o_2$ by the value $(a, b)$. Concerning $id_1$ and $id_2$, they are not identical but shallow–equal and deep–equal. In general,

$$(o_1 = o_2) \Rightarrow (o_1 \doteq o_2) \Rightarrow (o_1 \cong o_2)$$
$$\text{identical} \Rightarrow \text{shallow} - \text{equal} \Rightarrow \text{deep} - \text{equal}$$

and these correspond to 0–equal, 1–equal and $w$–equal of $O_2$ (Cluet *et. al.*) and to identity, shallow–equality and deep–equality of *smalltalk*–80 (Goldberg and Robson, 1983).

### 2.3. Methods

In this section, we define methods and based on methods we define predicate expressions.

**Definition 2.6.** A method $mt$ is defined as a pair $(ms, f)$, where

- $ms \in M_s$ is the message used to invoke the method
- $f : c_1 \times c_2 \times ... \times c_n \rightarrow c_r$, is the function of the method, where $c_1$ is the domain of the receiver with $c_2, ..., c_n$ being the domains for arguments of $f$ and $c_r$ is the domain of the result. That is, $\forall o_i \in c_1 \exists o_2 \in c_2 \exists o_3 \in c_3 ... \exists o_n \in c_n \exists o_r \in c_r$ such that

  $$(o_1\ ms) = f(o_1, o_2, ..., o_n) = o_r$$

  we use the notion $arguments(f)$ to denote $c_1, c_2, ..., c_n, c_r$ □

Consider the following illustrating examples. Let $c_3$ and $c_4$ be two classes. $(manager,\ manager,\ f : c_4 \rightarrow c_3)$ is a method defined for class $c_4$ such that:

$$\forall o \in T_{instances}(c_4),\quad f(o) \in T_{instances}(c_3)$$

$$o_4\ manager() = f(o_4) = o_3$$

$(inc\_sal,\ increase\_salary,\ f : c_3 \times integer \rightarrow integer)$

if $salary(o_3) = 13K$ then $o_3\ increase\_salary(2K) = f(o_3, 2K) = 15K$

$(greater\_than,\ >,\ f : integer \times integer \rightarrow boolean)$

$$(3 > 4) = f(3, 4) = F$$

Given that class $c \in arguments(f), \forall c_i$ such that $c_i \leq_c c, arguments(f)$ and $arguments(f) - \{c\} + \{c_i\}$ are equivalent, i.e., $c_i$ substitutes $c$ in $arguments(f)$, this is because

$$T_{instances}(c_i) \subseteq T_{instances}(c)$$

### 2.3.1. Message Expressions

Given a class $c$, we use $messages(c)$ to denote the set of messages corresponding to $methods(c)$. To define a message expression we have

- $\forall iv \in I_{variables}(c)$, let $domain(iv) = c_j$, $\forall o \in T_{instances}(c) \exists m \in messages(c) | (o\ m) = value(iv) \wedge (o\ m) \in T_{instances}(c_j) \Rightarrow$ $(T_{instances}(c)m) \subseteq T_{instances}(c_j)$, ($c_j$ is the domain of the result of the function that corresponds to $m$),
- and $\forall m_i \in messages(c_j)$, $(o\ m)m_i$ is defined,

we call $m\ m_i$ a *message expression.*

**Definition 2.7.** A *message expression* is a sequence of messages $m_1 m_2 ... m_n$ with $n \geq 1$, such that given an object $o_0$,

$o_0 m_1 m_2 ... m_n$ is defined iff:

$$\begin{array}{lll} m_1 \in messages(class(o_0)) & \wedge & (o_0\ m_1) = o_1 \wedge \\ m_2 \in messages(class(o_1)) & \wedge & (o_1\ m_2) = o_2 \wedge \\ & \vdots & \\ m_n \in messages(class(o_{n-1})) & \wedge & (o_{n-1}\ m_n) \text{ is defined} \end{array}$$

$\Rightarrow \forall o_i \in T_{instances}(class(o_j))$ with $(0 \leq j \leq n-1)$,
$(o_i m_{j+1} m_{j+2} ... m_{j+k}) \in T_{instances}(class(o_{j+k}))$ with $(j + k \leq n)$.
The set $M_E$ is used to denote the universe of message expressions. □

Given a class $c$, we use $M_e(c)$ to denote message expressions of class $c$, and

$$M_e(c) = \{x | \forall o \in T_{instances}(c), (o\ x) \text{is defined}\}$$

Concerning message expressions we have the followings:

- $\forall x \in M_e(c) \exists m \in messages(c) | x = (m\ x_j) \wedge \forall o \in T_{instances}(c)$, $(o\ m) \in T_{instances}(c_j) \wedge x_j \in M_e(c_j)$
- $len(x)$ denotes the number of messages in the message expression $x$.
- Let $X = \{x_1, x_2, ..., x_n\}$ be a set of message expressions and $x_j$ be a message expression, then $(x_j X) = \{(x_j x_1), (x_j x_2), ..., (x_j x_n)\} = \{x_{11}, x_{12}, ..., x_{1n}\}$.
- Let $c \in C \wedge X \subseteq M_e(c) | \exists c_1, X = (x_i M_e(c_1))$ where $x_i$ is a message expression, then $((M_e(c) - X) \cup \{x_i\})$ is an optimized form of $M_e(c) \Rightarrow messages(c)$ is an optimized form of $M_e(c)$.
- $\forall o \in T_{instances}(c)$, due to encapsulation, the only values that can be returned from $o$ are given by $(oM_e(c))$ if $M_e(c)$ is empty then no values can be returned, although $value(o)$ is not empty.

At this point, it is required to define a predicate expression before going into the definition of the object algebra in the next section. But before doing that, we give the following concepts to be used in the definition of a predicate expression:

- $\forall x \in M_E$ such that given $o$ with $(o\ x)$ being defined and returns either a single value or a set *value* then $\exists d \in D$ such that $(o\ x) \subset d$,
- $(o_1\ x_1) = (o_2\ x_2) \Leftrightarrow (\forall o_i \in (o_1\ x_1)\ \exists o_j \in (o_2\ x_2)$ such that $o_i = o_j) \wedge (\forall o_j \in (o_2\ x_2)\ \exists o_i \in (o_1\ x_1)$ such that $o_i = o_j)$,
- $(o_1\ x_1) \doteq (o_2\ x_2) \Leftrightarrow (\forall o_i \in (o_1\ x_1)\ \exists o_j \in (o_2\ x_2)$ such that $o_i \doteq o_j) \wedge (\forall o_j \in (o_2\ x_2)\ \exists o_i \in (o_1\ x_1)$ such that $o_i \doteq o_j)$,
- $(o_1\ x_1) \cong (o_2\ x_2) \Leftrightarrow (\forall o_i \in (o_1\ x_1)\ \exists o_j \in (o_2\ x_2)$ such that $o_i \cong o_j) \wedge (\forall o_j \in (o_2\ x_2)\ \exists o_i \in (o_1\ x_1)$ such that $o_i \cong o_j)$,
- let $d \in D$ be a conventional domain, $\forall v \in d$ we have $value(v) = v$ and $identity(v) = v$.

**Definition 2.8. Predicate Expression.** Any of the following, and nothing else, is a predicate expression:

- $\boldsymbol{T}$ and $\boldsymbol{F}$ are predicate expressions representing true and false,
- $\forall o \in O\ \forall d \in D$, we have $o \in d$ and $o \notin d$ as predicate expressions,
- $\forall o \in O\ \forall x \in M_E | (o\ x)$ is defined, $\forall d \in D\ \forall d_i \subseteq d$, we have $(o\ x) \subseteq d_i$, $(o\ x) = d_i$, $(o\ x) \doteq d_i$ and $o\ x) \cong d_i$ as predicate expressions,
- $\forall o_1, o_2 \in O$, we have $o_1 = o_2$, $o_1 \doteq o_2$ and $o_1 \cong o_2$ as predicate expressions,
- $\forall o \in O\ \forall x \in M_E | (o\ x) \subset d$, where $d \in D$ is a conventional domain, we have $(\forall v \in (o\ x), v\ op\ y)$ and $(\exists v \in (o\ x), v\ op\ y)$ as predicate expressions, where $op \in \{=, >, \geq, <, \leq\}$ and $y \in d$,
- if $p_1$ and $p_2$ are predicate expressions, then $(p_1), -p_1, p_1 \vee p_2$ and $p_1 \wedge p_2$ are all predicate expressions. □

## 3. Formal Definition of Object Algebra

Based on the previously introduced terminology, in this section we formally define object algebra expressions. When speaking about $len(x)$ in any of the constraints given next in this section, we'll consider only message expressions $x$ such that $x$ returns a stored value from a conventional domain. This follows from the fact that if $len(x) = 1$ and $x$ returns a stored value from a basic domain; $len(x_1) > 1$.

**Definition 3.1. Object Algebra Expressions.** Let $E$ be the set of object algebra expressions. $\forall e \in E, M_e(e)$ is defined with $card(M_e(e)) \geq 1$ and $T_{instances}(e)$ is defined.

Given $e_1 \in E$ and $e_2 \in E$ with $M_e(e_1) = X_1$, $M_e(e_2) = X_2$ and $T_{instances}(e_1) = T_1$, $T_{instances}(e_2) = T_2$. Elements of $E$ are enumerated as follows:

- $\forall c_i$, by definition $M_e(c_i)$ and $T_{instances}(c_i)$ both exist, $\Rightarrow c_i \in E$.
- Projection: This operation hides a part of the objects in the operand.

  Given $X \subseteq X_1$, $e_1[X] \in E$ with $M_e(e_1[X]) = X$ $T_{instances}(e_1[X] = T_{instances}(e_1)$.

- Cross–Product: This operation forms new objects out of objects in the operands. It is defined in a way to satisfy associativity.

  $(e_1 \times e_2) \in E$ with,

$$M_e(e_1 \times e_2) = \begin{cases} (m_1X_1) \cup (m_2X_2) & \text{if } \exists x_i \in X_1,\ len(x_i) = 1 \wedge \exists x_j \in X_2,\ len(x_j) = 1 \\ X_1 \cup (m_2X_2) & \text{if } \forall x_i \in X_1,\ len(x_i) > 1 \wedge \exists x_j \in X_2,\ len(x_j) = 1 \\ (m_1X_1) \cup X_2 & \text{if } \exists x_i \in X_1,\ len(x_i) = 1 \wedge \forall x_j \in X_2,\ len(x_j) > 1 \\ X_1 \cup X_2 & \text{if } \forall x_i \in X_1,\ len(x_i) > 1 \wedge \forall x_j \in X_2,\ len(x_j) > 1 \end{cases}$$

  where $m_1$ and $m_2$ are two messages with $e_1$ and $e_2$ being the domains of the results of $m_1$ and $m_2$, respectively.

$$T_{instances}(e_1 \times e_2) = \begin{cases} \{o \mid \exists o_1 \in T_1\ \exists o)2 \in T_2 \wedge value(o) = identity(o_1) \\ \quad .identity(o_2)\} \text{ if } \exists x_i \in X_1,\ len(x_i) = 1 \wedge \\ \quad \exists x_j \in X_2,\ len(x_j) = 1 \\ \{o \mid \exists o_1 \in T_1\ \exists o)2 \in T_2 \wedge value(o) = value(o_1) \\ \quad .identity(o_2)\} \text{ if } \forall x_i \in X_1,\ len(x_i) > 1 \wedge \\ \quad \exists x_j \in X_2,\ len(x_j) = 1 \\ \{o \mid \exists o_1 \in T_1\ \exists o)2 \in T_2 \wedge value(o) = identity(o_1) \\ \quad .value(o_2)\} \text{ if } \exists x_i \in X_1,\ len(x_i) = 1 \wedge \\ \quad \forall x_j \in X_2,\ len(x_j) > 1 \\ \{o \mid \exists o_1 \in T_1\ \exists o)2 \in T_2 \wedge value(o) = value(o_1) \\ \quad .value(o_2)\} \text{ if } \forall x_i \in X_1,\ len(x_i) > 1 \wedge \\ \quad \forall x_j \in X_2,\ len(x_j) > 1 \end{cases}$$

  where . is being used to indicate a concatenation of the two arguments.

- Selection: Returns from the operand those objects satisfying a given predicate.

  Given a predicate expression $p, e_1[p] \in E$ with $M_e(e_1[p]) = M_e(e_1) = X_1$ $T_{instances}(e_1[p]) = \{o \mid o \in T_1 \wedge (o)\}$.

- Union: Returns all the objects in the operands.

  $(e_1 \cup e_2) \in E$ with $M_e(e_1 \cup e_2) = X_1 \cap X_2$ and $T_{instances}(e_1 \cup e_2) = T_1 \cup T_2$.

- Difference: This operation is handled in one of two ways depending on the relationship between the message expressions of the operands. If the message expressions of the first operand is subset from the of message expressions of the second operand, the difference operation returns objects from the first operand which are not in the second operand. Otherwise, it is handled as a projection of objects in the first operand on values that have no corresponding message expression in the second operand

$(e_1 - e_2) \in E$ with

$$M_e(e_1 - e_2) = \begin{cases} X_1 & \text{if } X_1 \subseteq X_2 \\ (X_1 - X_2) & \text{otherwise} \end{cases}$$

$$T_{instances}(e_1 - e_2) = \begin{cases} T_1 - T_2 & \text{if } X_1 \subseteq X_2 \\ T_1 & \text{otherwise} \end{cases}$$

- Nest: It is not possible to have all the desired relationships predefined at the modeling level; this operation introduces such relationships

  $(e_1 >> e_2) \in E$ with $M_e(e_1 >> e_2) = X_1 \cup (m\ X_2)$, where $e_2$ is the domain of the result of $m$. $T_{instances}(e_1 >> e_2) = \{o | \exists o_1 \in T_1 \wedge value(o) = value(o_1).v$, where $v = (o\ m) \wedge v \in T_2\}$.

- One level projection: The previously introduced projection does not evaluate any message expression. On the other hand, the one level projection operation evaluates the provided message expressions and form objects out of the obtained values.

  Given $X \subseteq X_1$, $e_1![X] \in E$ with
  $M_e(e_1![X]) = \{x | \exists x_1 \in X \wedge x_1 = x_2 x \wedge\ len(x_1) = len(x_2) + 1\}$
  $T_{instances}(e_1![X]) = \{o | \exists o_1 \in T_1 \wedge value(o) = (o_1\ X)\}$.

- Unnest: defined in terms of projection as,
  $(e_1 << e_2) = e_1[X_1 - X | X = (m\ X_2) \wedge\ \forall o \in T_1,\ (o\ m) \in T_2]$.

- Intersection: defined in terms of difference as,
  $(e_1 \cap e_2) = e_1 - (e_1 - e_2)$.

- Aggregation: $e_1 < X, f, X_i > \in E$ with $M_e(e_1 < X, f, X_i >) = (m\ M_e(e_1)) \cup \{m_1\}$, where $e_1$ is the domain of the result of $m$, and the domain of the result of $f$ is the domain of the result of $m_1$. $T_{instances}(e_1 < X, f, X_i > ) = \{o | (o\ m) \subseteq T_1 \wedge (o\ m_1) = f(\{(o_1\ X_i) | o_1 \in T_1 \wedge\ \forall o_2 \in (o\ m), (o_2\ X) = (o_1\ X)\})\}$ .
  The aggregation function is applied on $e_1$ by evaluating the function $f$ on the result of the message expression $X_i$ for all objects that return the same values for elements of the set of message expressions $X$. □

Next are some equivalent object algebra expressions. We believe that equivalence of object algebra expressions will be useful in optimization, although not covered in this paper.

### 3.1. Equivalence of Object Algebra Expressions

Let $e_1$, $e_2$ and $e_3$ be object algebra expressions, such that

$$M_e(e_1) = X_1,\quad M_e(e_2) = X_2 \text{ and } M_e(e_3) = X_3.$$

- Given two predicate expressions $p_1$ and $p_2$,

  . $e_1[p_1][p_2] = e_1[p_1 \wedge p_2] = e_1[p_2 \wedge p_1] = e_1[p_2][p_1]$.

- Given $X \subseteq X_1$ and a predicate expression $p_1$,
  . $e_1[p_1][X] = e_1[X][p_1]$ iff $\forall x \in p_1 \Rightarrow x \in X$.
- $e_1 \times e_2 = e_2 \times e_1$
- $e_1 \times (e_2 \times e_3) = (e_1 \times e_2) \times e_3$
- $e_1 \times e_2 = \begin{cases} e_1 >> e_2 & \text{iff } \forall x_1 \in X_1,\ len(x_1) > 1 \wedge \exists x_2 \in X_2,\ len(x_2) = 1 \\ e_2 >> e_1 & \text{iff } \exists x_1 \in X_1,\ len(x_1) = 1 \wedge \forall x_2 \in X_2,\ len(x_2) > 1 \end{cases}$
- $e_1![X] = e_1[X]$ iff $\forall x \in X,\ len(x) = 1$ regardless of the domain underlying the value returned by $x$,
- $e_1 \bigcup e_2 = e_2 \bigcup e_1$
- $e_1 \bigcup (e_2 \bigcup e_3) = (e_2 \bigcup e_1) \bigcup e_3$
- Given $X_4 \subseteq X_1$ and $X_5 \subseteq X_1$,
  . $e_1[X_4][X_5] = e_1[X_5]$ iff $X_5 \subseteq X_4$

The proof of the given equivalences will be left out as an exercise to the reader; all follow from definition 3.1. As an example, next we sketch the steps that lead to the proof of the associativity of cross-product. We define equality of object algebra expressions and give a lemma on the length of message expressions in $(e_1 \times e_2)$.

**Definition 3.2. Equality of Object Algebra Expressions.** Let $e_1$ and $e_2$ be two elements of $E$. $e_1$ and $e_2$ are equal iff:

- $M_e(e_1) = M_e(e_2)$
- $\forall o_1 \in e_1\ \exists o_2 \in e_2 | (o_1) \cong (o_2)$, i.e., $o_1$ and $o_2$ are deep-equal,
- $\forall o_2 \in e_2\ \exists o_1 \in e_1 | (o_1) \cong (o_2)$, i.e., $o_1$ and $o_2$ are deep-equal.

□

**Lemma 3.1.** *Let $e_1$ and $e_2$ be elements of $E$.*

$$\forall x \in M_e(e_1 \times e_2),\quad len(x) > 1$$

*Proof:* Let $X_1$ and $X_2$ be the message expression of $e_1$ and $e_2$, respectively. Depending on the definition of $(e_1 \times e_2)$, there are four cases to consider:

i) $\exists x_1 \in X_1,\ len(x_1) = 1 \wedge \exists x_2 \in X_2,\ len(x_2) = 1$
$M_e(e_1 \times e_2) = \{x | x \in ((m_1\ X_1) \cup (m_2\ X_2))\}$ *(by definition)*
$\Rightarrow \forall x \in M_e(e_1 \times e_2)\ \exists x_1 \in X_1,\ len(x) = len(x_1) + 1 \vee \exists x_2 \in X_2,$
$len(x) = len(x_2) + 1 \Rightarrow len(x) > 1$

ii) $\forall x_1 \in X_1,\ len(x_1) > 1 \wedge \exists x_2 \in X_2,\ len(x_2) = 1$
$M_e(e_1 \times e_2) = \{x | x \in (X_1 \cup (m_2\ X_2))\}$ *(by definition)*
$\Rightarrow \forall x \in M_e(e_1 \times e_2)\ \exists x_1 \in X_1,\ len(x) = len(x_1) \vee \exists x_2 \in X_2,$
$len(x) = len(x_2) + 1 \Rightarrow len(x) > 1$

iii) $\exists x_1 \in X_1,\ len(x_1) = 1 \wedge \forall x_2 \in X_2,\ len(x_2) > 1$

$M_e(e_1 \times e_2) = \{x | x \in ((m_1\ X_1) \cup X_2)\}$ *(by definition)*

$\Rightarrow \forall x \in M_e(e_1 \times e_2)\ \exists x_1 \in X_1,\ len(x) = len(x_1) + 1 \vee \exists x_2 \in X_2,$

$len(x) = len(x_2) + 1 \Rightarrow len(x) > 1$

iv) $\forall x_1 \in X_1,\ len(x_1) > 1 \wedge \forall x_2 \in X_2,\ len(x_2) > 1$

$M_e(e_1 \times e_2) = \{x | x \in (X_1 \cup X_2)\}$ *(by definition)*

$\Rightarrow \forall x \in M_e(e_1 \times e_2)\ \exists x_1 \in X_1,\ len(x) = len(x_1) \vee \exists x_2 \in X_2,$

$len(x) = len(x_2) \Rightarrow len(x) > 1$

Hence, $len(x) > 1$. □

**Theorem 3.1.** it The Cross–product operation is associative.

That is, $\forall e_1, e_2, e_3 \in E$, we have:

$$(e_1 \times e_2) \times e_3 = e_1 \times (e_2 \times e_3)$$

*Proof:* Follows from definition 3.2 and lemma 3.1. □

## 4. Related Work and Conclusions

A number of proposals on query languages for object–oriented databases are reported in the literature (Abiteboul and Kanellakis, 1989; Alashqur *et. al.*, 1989; Albano *et. al.*, 1985; Bancilhon *et. al.*, 1987; Banerjee *et. al.*, 1988; Beech, 1988; Carey *et. al.*, 1988; Cluet *et. al.*; Dayal, 1989; Fishman *et. al.*, 1987; Guo *et. al.*, 1991; Kim *et. al.*, 1989; Kim, 1989; Maier and Stein 1987; Osborn, 1988; Rowe and Stonebraker, 1987; Shaw and Zdonik, 1989; Straube, 1991; Zaniolo, 1983). These languages were developed based on different paradigms. The query language of (Dayal, 1989) is based on the functional paradigm, while the query language of (Banerjee *et. al.*, 1988; Kim *et. al.*, 1989) is based on the message–passing paradigm. Other languages are based on extensions to the relational paradigm (Beech, 1988): such as extensions of QUEL (Carey *et. al.*, 1988; Rowe and Stonebraker, 1987) and extensions of SQL (Cluet *et. al.*). The query language of IRIS (Fishman *et al.*, 1987) is based on both the functional and the relational paradigms, where functions are used in an object–oriented SQL, OSQL, constructs.

A major drawback of languages such as those described in (Banerjee *et. al.*, 1988; Kim *et. al.*, 1989; Maier and Stein, 1987; Straube, 1991) is that they do not satisfy the closure property. Although operands in such languages have object–oriented properties, whereas the output is a relation which does not have the same structural and behavioral properties as the original objects. Consequently, the result of a query cannot be further processed by the same set of language operators.

The query languages of (Bancilhon *et. al.*, 1987; Carey *et. al.*, 1988; Rowe and Stonebraker, 1987) use nested relations as their logical view of object–oriented databases. Although operators in these languages operate on nested relations and

produce nested relations, we argue that nested relations do not form a proper logical representation of object associations. In order to use nested relations to represent object a large amount of data has to be replicated in the representation.

Some of the models described in the literature introduce non–object–oriented constructs in supporting the closure property. For instance in $O_2$ (Cluet *et al.*) the value concept was introduced while in (Shaw and Zdonik, 1989) the output of a query is of the Tuple type which is essentially the nested relational representation, since it allows the nesting of tuples. Regarding our work, the closure property is satisfied by the algebra described in this paper as the operand(s) and output of any operator are defined to be compatible by having a set of objects and a set of message expressions. Out of the scope of this paper, we have proved that the output of any of the described operators ends up to be a class. Also we have derived the relationship between such a class and classes existing in the lattice. All of these are being done to allow the output of a query to persist in the lattice without violating any of the object–oriented features. By doing this we'll have schema changes supported by the object algebra.

Finally, it is important to indicate that although the algebra described in (Straube, 1991) is formal, but that algebra is restricted to object retrievals where the introduction of new relationships and the closure property are left out. In this sense our algebra is more general by considering both the closure property and the introduction of new relationships, and hence the creation of new objects.

## References

**Abiteboul S., Fischer P.C. and Scheck H.J.** (Eds) (1989): *Nested Relations and Complex Objects in Databases.* Lecture Notes in Computer Science.– Heidelberg: Springer, v.361.

**Abiteboul S. and Kanellakis P.C.** (1989): *Object identity as a query language primitive.*– Proc. ACM SIGMOD, pp.159–173.

**Alashqur A., Su S. and Lam H.** (1989): *OQL: A query language for manipulating object–oriented databases.*– Proc. 15th Int. Conf. on Very Large Databases, pp.433–442.

**Albano A., Cardelli L. and Orsini R.** (1985): *Gelileo: A strongly–typed interactive conceptual language.*– ACM Trans. on Database Systems, v.10, No.2, pp.230–260.

**Atkinson M.** *et. al.* (1989): *The object–oriented database system manifesto.*– Proc. Inter. Conf. on Deductive Object–Oriented Databases.– Kyoto, Japan.

**Bancilhon F.** *et. al.* (1987): *FAD: A powerful and simple database language.*– Proc. VLDB'87, pp.97–105.

**Banerjee J.** *et. al.* (1987): *Data model issues for object–oriented applications.*– ACM Trans. on Office Information Systems, v.5, No.1, pp.3–26.

**Banerjee J., Kim W. and Kim K.C.** (1988): *Queries in object–oriented databases.*– Proc. Fourth Int. Conf. on Data Engineering, Los Angeles, CA.

**Beech D.** (1988): *A Foundation for Evolution from Relational to Object Databases.*– EDBT'88, Springer LNCS 303.

**Beeri C.** (1989): *Formal models for object-oriented databases.*– 1st Int. Conf. on Deductive and Object-Oriented Databases.– Kyoto: North-Holland, pp.370–395.

**Cardelli L. and Wegner P.** (1985): *On Understanding Types.– Data Abstraction and Polymorphism.*– ACM Computing Surveys, v.17, No.4, pp.471–522.

**Carey M.J. and DeWitt D.J.** (1986): *The architecture of the EXODUS extensile DBMS.*– Proc. IEEE Int. Workshop on Object-Oriented Database Systems, Pacific Grove, pp.52–65.

**Carey M.J., DeWitt D.J. and Vandenberg S.L.** (1988): *A data model and a query language for EXODUS.*– Proc. of ACM SIGMOD Conf. on Management of Data, pp.413–423.

**Cluet S.** *et. al.*: *Reloop, an algebra based query language for an object-oriented database system.*– Proc. First Int. Conf. on Object-Oriented and Deductive Databases.

**Date C.J.** (1986): *An Introduction to Database Systems* .– New York: Addison-Wesley, (Fourth Edition, v.1 and v.2).

**Dayal U.** (1989): *Queries and views in an object-oriented data model.*– 2nd Int. Workshop on Database Programming Languages, pp.80–102.

**Deux O.** *et. al.* (1990): *The Story of O2.*– IEEE Trans. on Knowledge and Data Engineering, v.2, No.1, pp.91–108.

**Fishman D.H.** *et. al.* (1987): *IRIS: An object-oriented database management system.*– ACM Trans. on Office Information Systems, v.5, No.1, pp.48–69.

**Goldberg A. and Robson D.** (1983): *Smalltalk-80: The Language and Its Implementation.*– New York: Addison Wesley.

**Guo M.** *et. al.* (1991): *An associative algebra for processing object-oriented databases.*– IEEE Data Engng. Conf.

**Khoshafian S.N. and Copeland G.P.** (1986): *Object identity.*– Proc. Int. Conf. on Object-Oriented Programming Systems, Languages and Applications.

**Kim K.C.** *et. al.* (1989): *Cyclic query processing in object-oriented databases.*– IEEE Data Engng. Conf.

**Kim W.** (1989): *A model of queries for object-oriented databases.*– Proc. 15th Int. Conf. on Very Large Databases, pp.423–432.

**Kim W.** (1990): *Object-oriented databases: Definition and research directions.*– IEEE Trans. on Knowledge and Data Engineering, v.2, No.3, pp.327–341.

**Maier D. and Stein J.** (1987): *Development and Implementation of an Object-Oriented DBMS.*– Research Directions in Object-Oriented Programming, Shriver B. and P. Wegner (Eds).

**Osborn S.L.** (1988): *Identity equality and query optimization.*– 2nd Int. Workshop on Object-Oriented Database Systems, pp.346–351.

**Rowe L.A. and Stonebraker M.R.** (1987): *The postgres data model.*– Proc. VLDB'87, pp.83–96.

**Shaw G. and Zdonik S.** (1989): *An Object–Oriented Query Algebra.*– Quart. Bull. of the IEEE TC on Data Engineering, v.12, No.3, pp.29–36.

**Stefik M. and Bobrow D.G.** (1986): *Object–oriented programming: Themes and variations.*– AI Magazine, pp.40–62.

**Straube D.D.** (1991): *Queries and Query Processing in Object–Oriented Database Systems.*– Ph.D. Thesis, Depart. of Computing Sci., University of Alberta, Spring.

**Zaniolo C.** (1983): *The database language GEM,* .– SIGMOD'83, pp.207–218.