EXTENDED OQL FOR OBJECT ORIENTED PARALLEL QUERY PROCESSING

S. G. Fountoukis[1]* and M. P. Bekakos[2]

[1]Dept of Informatics with Applications in Biomedicine, University of Central Greece, Lamia 35100, Hellas*
Email: sfount@phs.uoa.gr
[2]Dept. of Electrical & Computer Engineering, School of Engineering, Democritus University of Thrace, Xanthi 67100, Hellas
Email: mbekakos@ee.duth.gr

ABSTRACT

Herein, an extension to the object query language (OQL) for incorporating binary relational expressions is investigated. The extended query language is suitable for query submissions to an object oriented database, whose functionality is based upon the algebra of binary relations. Algebraic expressions, consisting of simple and multiple merged chains of binary relations, are stated in SQL syntax-based object queries, which are utilized by a multiwavefront algorithm mapped on a multi-directional multi-functional engine(M^2FE), for object oriented parallel query processing. The proposed extension also attempts to solve other object oriented database issues, such as inheritance, relationships between objects and literals, and recursive queries.

Keywords: Object oriented databases, Object query language extension, Object oriented recursive queries, Parallel query processing, Algebra of binary relations

1 INTRODUCTION

During the operation of an object oriented information system, objects are derived, which are holding references one to another. Thus, sets of related (connected through references) pairs of objects are formed. In a pair of objects, the direction of the relationship between its members is defined to be from the object that holds the reference towards the other whose reference in being held. A binary relation is considered as a set of ordered object *pairs*. These sets of ordered object pairs can constitute an object oriented database, whose object manipulation (Gyssens, 1994b; Sarathy, 1993; Van den Bussche, 2001) can be based upon the algebraic framework of the binary relations algebra (Tarski, 1941; Givant, 1994; Desharnais, 1989).

Attempts to use the algebra of binary relations for object manipulation purposes resulted in graph-oriented systems, where graphs were used for querying object databases (Gyssens, 1994a; Gyssens, 1994b; Sarathy, 1993) or structural manipulation of software system architectures (Hold, 1998; Fahmy, 2000). These graph systems obviously have serious disadvantages, because the large number of derived objects results in large scheme graphs, which are difficult to understand, handle or even to be displayed. To avoid graph complexities and disadvantages in querying object-oriented databases, an *extension* to the existing OQL (Cattell, 2000) for incorporating expressions of binary relations algebra in queries, is proposed.

Because OQL has adopted the use of the SQL form, the algebraic expressions can be stated using SQL syntax, which can also facilitate users familiarized with legacy database systems. Connecting a series of binary relations through the algebraic composition operator, simple and multiple chains are formed. The multiple chains can be merged for query optimization purposes. These chains are stated in queries using SQL syntax. A multiwavefront-like algorithm, which is mapped on a multi-directional multi-functional engine, is proposed for efficient query processing purposes. This algorithm (and engine) processes the simple and multiple merged chains of binary relations introduced in the OQL extension and stated in queries. The basic elements of the multi-functional engine architecture can be of two types of array processors: a (synchronous) square systolic device for wavefront-like computations - S^2DWC (Fountoukis, 2004) or a bidirectional parallel cubic engine - *BPCE* (Fountoukis, 2005).

[1*]*Correspondence Address: POBox 4176, Athens 10210, Hellas.*

Issues concerning object inheritance, relationships between objects and literals, and the object oriented recursive queries are attempted to be solved in the framework of the algebra of binary relations.

The paper is structured as follows. Section 2 describes the basic concepts of binary relations between objects. Section 3 provides an overview of the algebra of binary relations. The proposed object query language extension is covered in Section 4. In section 5 a short description of the multiwavefront algorithm, which processes the simple and multiple merged chains of binary relations and synthesizes multiple relations, along multiple directions is given. Section 6 concludes the whole work.

2 BINARY RELATIONS OF OBJECTS

There are three types of relationships (except *inheritance*) between classes of objects: *association*, *aggregation* and *composition*. All these relationships imply that one object is tied to another through a reference stored in an instance attribute. The direction of the relationship is considered to be from the object that holds the reference towards the other whose reference is being held and used (left part of Figure 1).

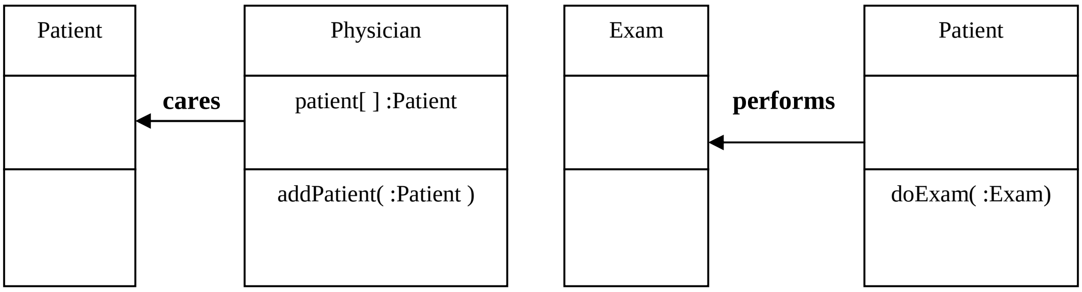


Figure 1. The directed binary relationship between two (classes of) objects at left and two (classes of) dependent objects at right.

If an object has a method that *uses* another object (e.g., through a reference as parameter), then the first object *depends* upon the second object to do something. In this case the relationship and its direction are considered to be as in the previous case (right of Figure 1).

The pairs of objects, thus formed, are entered in two column tables constituting the binary relations. Because each object has a unique identifier ("id"), which discriminates it from other objects, instead of pairs of objects, pairs of their corresponding "ids" can populate these tables (relations). The identifiers can be integers or combination of letters and integers, which are again equivalent to integers. Each binary table can be named with the name of relationship between the corresponding classes of objects (Figure 1). Object relations (tables) are illustrated in Figure 2.

cares

Phy	Pat.
ph0	pa0
ph1	pa2
ph1	pa3
ph2	pa1

performs

Pat	Exa.
pa0	ex1
pa0	ex3
pa2	ex0
pa3	ex4

Figure 2. Object relations (binary tables), which are populated by pairs of object "ids"

3 ALGEBRA OF BINARY RELATIONS

Assume that a set A of object binary relations, whose relations r, s are members, exists. Each member of the set A is also a set, having as members ordered pairs of object ids. Each ordered object pair indicates a directed relationship between its two members. The algebra of binary relations is an important model of *relation algebras* axiomatized by Tarski and his colleagues (Tarski, 1941; Givant, 1994). According to their original works, the core of the binary relations algebra consists of five operators: *union, inverse, complement, composition* and *identity*. However, in the works of (Gyssens, 1994b; Sarathy, 1993; Van den Bussche, 2001) only the first four of the previous listed operators are considered; the identity is excluded while two new operators, the left and the right tagging, are introduced. The identity and the intersection operators are expressed through these new operators (Gyssens, 1994b).

In the present work, the five operators, originated by Tarksi and his colleagues are considered, instead of these alternative definitions. The tagging operators, introduced in (Gyssens, 1994b), herein are defined as *direct product* functions, whose images satisfy the direct product properties, which will be proved later on. The existence of the identity is required by the definition of the direct product of the algebra. The intersection operator does not belong to the core of the algebra; however it is naturally expressed, through the union and the complement, as a derived operator.

Let $r \in A$ be a relation. The set of *atomic objects* involved in the relation r is defined as $|r| = \{u \mid \exists v : (u,v) \in r \ \vee \ (v,u) \in r\}$. The *active domain* of r consisting of all the pairs of atomic objects involved in the relation r is the Cartesian product $|r| \times |r| = |r|^2$. The active domain of a relation can be used instead of infinite relations (Gyssens, 1994b; Sarathy, 1993; Bussche, 2001), since the number of objects participating in an object oriented database is finite.

The five operators, which consist of the core of the binary relations algebra, are:
(1) Union: $r \cup s$ that is the union of two sets (relations) r, s. More explicitly: $r \cup s = \{(u,v) \mid (u,v) \in r \ \vee \ (u,v) \in s\}$.
(2) Inverse: r^{-1}, that is the set $r^{-1} = \{(u,w) \mid (w,u) \in r\}$.
(3) Finite Complementation: $\overline{r}$, that is the set $\overline{r} = |r|^2 - r$. More explicitly: $\overline{r} = \{(u,v) \in |r| \times |r| \mid (u,v) \notin r\}$.
(4) Composition: $r \circ s$, that is the set $r \circ s = \{(u,w) \mid \exists v : (u,v) \in r, (v,w) \in s\}$.
(5) Identity: $I_r = \{(u,u) \mid u \in |r|\}$.
The algebra of binary relations is a Boolean algebra, according to the relation algebras axiomatization. Therefore, from De Morgan laws (Kelley, 1975; Dugundji, 1965), the intersection can be expressed through union and complement:
(6) $r \cap s = \overline{\overline{r} \cup \overline{s}}$.
The pair (r, s) of relations is called *direct product* (Desharnais, 1989) if and only if:
(a) $r^{-1} \circ r = I$, (b) $s^{-1} \circ s = I$, (c) $r^{-1} \circ s = 1$ and (d) $r \circ r^{-1} \cap s \circ s^{-1} = I$.
Two functions $r^{\lhd}$, $r^{\rhd}$: $|r|^2 \to |r|^2 \times |r|$, named left and right tagging operators, respectively, are defined with:
(7) $r^{\lhd}(x, y) = ((x, y), x), \quad x, y \in |r|$ and
(8) $r^{\rhd}(x, y) = ((x, y), y), \quad x, y \in |r|$.
Their images $r^{\lhd} = \{((x,y),x) \mid x, y \in |r|\}$ and $r^{\rhd} = \{((x,y),y) \mid x, y \in |r|\}$ constitute the pair ($r^{\lhd}$, $r^{\rhd}$), which is a direct product, since:
(a) $(r^{\lhd})^{-1} \circ r^{\lhd} = \{(x,(x,y)) \mid x, y \in |r|\} \circ \{((x,y),x) \mid x, y \in |r|\} = \{(x,x) \mid x \in |r|\} = I_{|r|^2}$.
(b) $(r^{\rhd})^{-1} \circ r^{\rhd} = I_{|r|^2}$ (similarly).
(c) $(r^{\lhd})^{-1} \circ r^{\rhd} = \{(x,(x,y)) \mid x, y \in |r|\} \circ \{((x,y),y) \mid x, y \in |r|\} = \{(x,y) \mid x, y \in |r|\} = 1_{|r|^2}$,

due to the fact that $1_{|r|^2}$ is defined as the whole set $|r| \text{ x } |r| = |r|^2$ and $0_{|r|^2}$ as the empty set $\varnothing$.

(d) $r^{\lhd} \circ (r^{\lhd})^{-1} \cap r^{\rhd} \circ (r^{\rhd})^{-1} =$

$$\{((x,y),x) \mid x,y \in |r|\} \circ \{(x,(x,y)) \mid x,y \in |r|\} \cap \{((x,y),y) \mid x,y \in |r|\} \circ \{(y,(x,y)) \mid x,y \in |r|\} =$$
$$\{((x,y),(x,y)) \mid x,y \in |r|\} \cap \{((x,y),(x,y)) \mid x,y \in |r|\} =$$
$$\{((x,y),(x,y)) \mid x,y \in |r|\} = I_{|r|^2 \times |r|^2}.$$

(9) $r^{\pi l} = (r^{\triangleleft})^{-1} \circ r^{\triangleleft}$, *left projection operator*, which is derived by the property (a). It maps the relation r onto the set $\{(u,u) \mid \exists v : (u,v) \in r\}$.

(10) $r^{\pi r} = (r^{\triangleright})^{-1} \circ r^{\triangleright}$, *right projection operator*, which is derived by the property (b). It maps the relation r onto the set $\{(v,v) \mid \exists u : (u,v) \in r\}$.

(11) $r^{\pi} = (r^{\triangleleft})^{-1} \circ r^{\triangleleft} \cup (r^{\triangleright})^{-1} \circ r^{\triangleright}$, *combined left and right projection operator*, which is derived by the combination (union) of the left and right projection operators. It maps the relation r onto the set $\{(u,u) \mid \exists v : (u,v) \in r \ \vee \ (v,u) \in r\}$, that is the set $\{(u,u) \mid u \in |r|\}$.

(12) The operator derived by (c), preserves the relation, that is, it maps r onto itself, while the operator derived by (d), maps r onto the set $\{(t,t) \mid t = (u,v),\ (u,v) \in r\}$.

The tagging operators are very useful in object ids creation (Gyssens, 1994b; Sarathy, 1993).

3.1 Extensions of the binary relations algebra

Adding a *while* construct and multiple assignments, the algebra of binary relations becomes a computationally *complete* database language (Gyssens, 1994b). Multiple assignments (assignment queries) are defined as expressions that compute several binary relations and return a single relation. They are used extensively in *while* loops.

4 OBJECT QUERY LANGUAGE EXTENSION

The object-oriented system, illustrated in Figure 3, partially represents an information system designed for a microsurgery clinic. The classes of the system are related through aggregation, composition and inheritance. Aggregation and composition are special forms of association, and they can be handled uniformly as associations. Inheritance relationships are also shown in the diagram. Two tables explaining the association and inheritance relationships between the classes follow the system diagram.

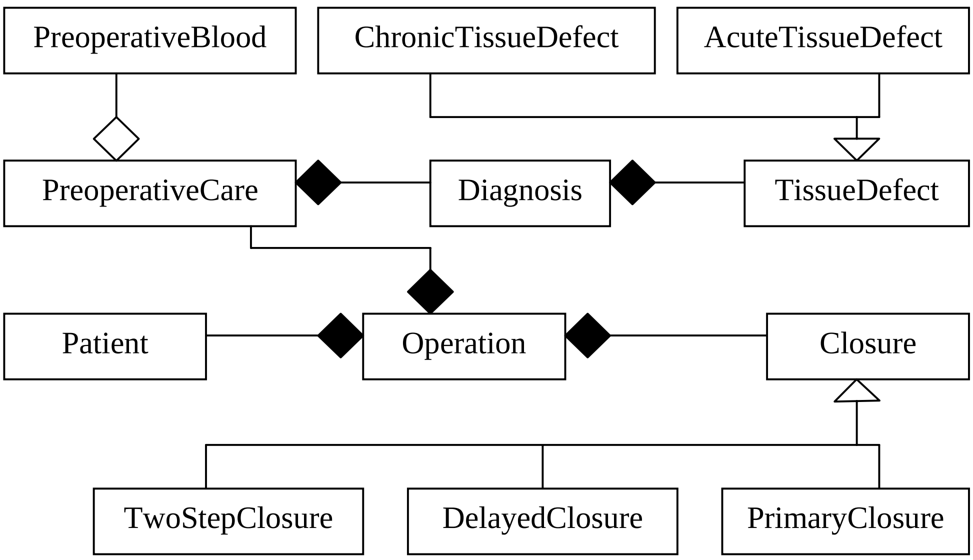


Figure 3. Draft class diagram of an object oriented system.

Table 1. Associations between classes. The order designates the direction of the relationships.

Associations			
Aggregations		**Compositions**	
PreoperativeCare	PreoperativeBlood	PreoperativeCare	Diagnosis
		Diagnosis	TissueDefect
		Operation	PreoperativeCare
		Operation	Patient
		Operation	Closure

Table 2. Inheritance relationships between classes.

Inheritance	
Parent Class	**Child Class**
TissueDefect	ChronicTissueDefect
TissueDefect	AcuteTissueDefect
Closure	TwoStepClosure
Closure	DelayedClosure
Closure	PrimaryClosure

PC	PB
pc0	pb0
pc1	pb2
pc1	pb3
pc2	pb5

PC	D
pc0	d0
pc2	d1
pc2	d3
pc3	d4

D	TD
d0	td0
d2	td1
d3	td3
d5	td5

O	P
o0	p1
o2	p1
o3	p4
o5	p5

O	PC
o1	pc0
o2	pc1
o4	pc2
o6	pc3

O	C
o0	c0
o2	c1
o3	c3
o5	c5

Figure 4. Binary relations of objects for the system of Figure 3.

For each association (associated pair of classes) of the system (Figure 3 & Table 1), an ordered binary table (relation) is created (Figure 4) containing pairs of objects (or their identifiers – "ids") derived by the two classes that are related. The following table explains the header names of the binary tables according to the relationships (associations) between classes that they represent.

Table 3. Correspondence betweeen header names of the relations (Figure 4) and associations (Table 1).

PC – PB	PreoperativeCare – PreoperativeBlood
PC – D	PreoperativeCare – Diagnosis
D – TD	Diagnosis – TissueDefect
O – P	Operation – Patient
O – PC	Operation – PreoperativeCare
O – C	Operation – Closure

In the works of Sarathy (1993) and Gyssens (1994b), where graph oriented systems are studied, binary relations are also formed for:
(a) the inheritance relationships,
(b) the relationships between objects and literals,
(c) each class separately.
In this paper, for all the above cases, a different approach is followed attempting to avoid additional complexity.

4.1 Dealing with inheritance

According to inheritance and polymorphism characteristics of the object oriented programming, an object of a subclass can be referred by a reference variable of the type of its super class. The following script of code illustrates this case:

```
Employee emp = new Director();
```

where *Director* is a subclass of the *Employee* super class. Therefore, in each column of each binary table, where super class objects are stored, its subclasses objects can also be stored and a super class reference variable can refer to any object of the column. For example, in the column TD, which represents the class *TissueDefect*, of the binary table (relation) D – TD (Figure 4), the objects of the subclasses (Figure 3) *ChronicTissueDefect* and *AcuteTissueDefect* can also be stored. Hence, the subclasses objects will be treated exactly as their super class objects. This extends the notion of the relation between the objects of two classes to the notion of the relation between the objects of two hierarchies of classes. The proposed solution is best suited for single inheritance object systems.

4.2 Representation of single objects and binary relations between objects and literals.

Let *p* represent an object of the *Patient* class containing the variables named *age*, *sex*, *address* and *name* of data types: *int* (*integer*), *String, String* and *String*, respectively. The values of type *int* are literals having no ids because they are not objects, while the values of type *String* are considered to be objects having as ids the values themselves. The binary relations between the class objects and the *values* stored in their variables can be represented writing the pairs: $(p-age)$, $(p-sex)$, $(p-address)$, $(p-name)$. If the principle of encapsulation has been applied during the design and implementation of the object oriented system, the instance variables of the objects and the values held by them might not be directly accessed. However, they can be indirectly accessed through *set* and *get* methods. In this case the pairs that represent the binary relations can be written: $(p-getAge())$, $(p-getSex())$, $(p-getAddress())$, $(p-geName())$. In this way, the applied encapsulation property does not have any negative impact in the object oriented query processing.

The binary relations between objects and literals should be treated as purely conceptual taking no care to encode them as binary tables, in contrast to the relations between objects. They can be derived from the *single* objects themselves, which in the algebra of binary relations are represented as pairs of the form: $(p - p)$. The binary representation of a single object is required because everything in the algebra of binary relations is handled uniformly. In this case, if the binary form of a single object is not written explicitly in a query, it is assumed that it is derived from *left or right projection operators* applied on other relations containing the required object as a member. These relations should be involved in any clause of the query. It is an implementation issue to realize this assumption.

4.3 Extension of the object query language

The incorporation of the following features of the extended (Section 3.1) binary relations algebra in the object query language – OQL is proposed: 1) the specification of the binary relations, 2) the original and derived algebraic operators, and 3) the extensions of the algebra (looping constructs). The proposed extension is described in the following paragraphs.

4.3.1 Input and result of queries

Let `relation` be a type of data defined by the algebraic binary relation concept.
The query:

```
select distinct (p – age)^πr
from (p – p)
where (p – name)^πr == ("Mike","Mike")
```

selects the set of ages of all persons named "Mike", returning a set of pairs having identical integer numbers as members. For example, a member of the set of pairs can be the (*35*, *35*), where 35 is a distinct age. Therefore, the result is of type: *set*<(*age*: *int*, *age*: *int*)>. Obviously the *age* and *name* are variables of the object holding *integer* and *String* values, respectively. The pair ("*Mike*", "*Mike*") is a value of the variable *name* having binary form. The exponential operator πr is the right projection operator. It holds: $(p - age)^{\pi r} == (age - age)$.
The query:

```
select distinct struct ((p – age)^πr,(p – sex)^πr)
from (p – p)
where (p – name)^πr == ("Mike","Mike")
```

performs similarly, but for each person named "*Mike*", it builds a structure containing pairs of values of the variables *age* and *sex*. Thus, the result is of type: *set*<(*struct*((*age*: *int*, *age*: *int*), (*sex*: *String*, *sex*: *String*)). For example, the structured pairs: ((*35*, *35*), ("*male*", "*male*")) can be a member of the result set.
The query:

```
select distinct struct ((o – type)^πr ,
                              select distinct (p – name)^πr
                              from (o – p)^πr )
from (o – p)^πl
```

returns a structure: *set*<*struct*((*type*: *String*, *type*: *String*), *set*<(*name*: *String*, *name*: *String*)>)>, where for every *type* of surgical operation (=o) builds the set of *names* of patients (=p) who have undergone it. The exponential operator πl is the left projection operator. It holds: $(o - p)^{\pi l} == (o - o)$. This query has a nested form, as a part of it is another query.
Similarly, the following query has also a nested form, because another query is declared in the *from* clause:
relation r;

```
select distinct struct ((p – age)^πr,(p – sex)^πr)
from (select r
      from (p – p)
      where r ⊂ (p – p) and (p – city)^πr == ("Athens","Athens"))
where (p – name)^πr ==("Mike","Mike")
```

The result of the whole query is of type: *set*<(*struc*((*age*: *int*, *age*: *int*), (*sex*: *String*, *sex*: *String*)))>. The result of the internal query is a sub relation: $r \subset (p-p)$, where the *String* value of the variable *city* contained in the object *p* of the *Patient* class is "*Athens*". In other words, it is the subset of all the patients living in the city of "*Athens*".

4.3.2 Path expressions

Consider the partial object oriented system illustrated in Figure 3 and Tables 1 and 2. Suppose that a diagnosis is required of the type (acute or chronic) of the tissue defect of a patient who has undergone a surgical operation. The diagnosis is assumed to be formed during the preoperative care phase. Because both the *ChronicTissueDefect* and *AcuteTissueDefect* are subclasses of the *TissueDefect* class, objects of either can be referred by the *tissueDefect* reference. Therefore, in the (not extended) OQL the following *path expression* gives the required type of tissue defect of a patient:

```
patient.operation.preoperativeCare.diagnosis.tissueDefect
```

where, *patient*, *operation*, *preoperativeCare*, *diagnosis*, *tissueDefect* are references of the corresponding classes: *Patient*, *Operation*, *PeoperativeCare*, *Dagnosis*, *TissueDefect*. Since the objects involved in the path hold references one of the other according to the diagram in Figure 3, the path written above expresses a complex object. The dot "." operator permits the entrance into the complex objects.

In the algebra of binary relations only relationships between objects are considered. A path expression can be written as a *series of compositions* of binary relations of objects. By applying the composition operator to them successively, a binary relation is derived, which expresses what is required. The expression:

$$(o - p)^{-1} \circ (o - pc) \circ (pc - d) \circ (d - td),$$

is a series of compositions of binary relations, where the *exponent* -1 is the reverse operator defined as $r^{-1} = \{(u,w) \mid (w,u) \in r\}$. The successive application of the composition operator on this expression produces the relation $(p-td)$. This relation expresses the patient and the diagnosed type of tissue defect.

These results introduce the use of series of compositions of relations in the queries syntax.

4.3.3 Definition of the extended composition of relations

In the algebra of binary relations, the expression $(a-b)\circ(b-c)\circ(c-d)$ implies the relation $(a-d)$, which is finally derived by successive applications of the composition operator on the relations of the given expression. When the composition concept is applied successively to a series of relations, it is by definition limited to a single relation. However, relations considered as intermediate in a successive composition process can be very useful in queries (Section 4.3.4). The *extension* of the concept of relation composition to include all the possible (initial, intermediate and final) relations derived during the successive application of the composition operator on a series of relations can be defined as follows:

I) A series of connected relations through the composition operator is defined as a *chain* of relations. The example expression $(a-b)\circ(b-c)\circ(c-d)$ is a chain of relations.

II) Given a chain of relations $r_1 \circ r_2 \circ \ldots \circ r_n$, where $n > 1$, the *extended composition* is the union of the set of relations $r_{i,j} = r_i \circ r_{i+1} \circ \ldots \circ r_j$, $\forall i, j : 1 \le i < j \le n$ and the set $\{r_i \mid i = 1, 2, \ldots, n\}$.

Applying the extended composition on the example chain, the set of the derived relations is $\{(a-c),\ (b-d),\ (a-d),\ (a-b),\ (b-c),\ (c-d)\}$. Note that the initial relations are all included in the extended composition. In case where the given chain consists of two relations, the application of the extended composition gives exactly the relation that is derived by the application of the original composition, plus the two initial relations.

The extended composition operator is *also included* in the proposed OQL extension.

4.3.4 Expressions of chains of relations in queries

The following query has a nested form:

$$
\begin{array}{l}
\texttt{select distinct struct } (p - td)^{\pi r} \ \texttt{(select distinct } (o - type)^{\pi r} \\
\qquad\qquad\qquad \texttt{from } (o - d) \\
\qquad\qquad\qquad \texttt{where } (d - date)^{\pi r} == (\text{“}Dec\ 3\ 2006\text{”},\ \text{“}Dec\ 3\ 2006\text{”}) \texttt{ and} \\
\qquad\qquad\qquad\qquad (o - date)^{\pi r} == \{\text{“}Dec\ 4\ 2006\text{”},\ \text{“}Dec\ 4\ 2006\text{”}\}) \\
\texttt{from } (o - p)^{-1} \circ (o - pc) \circ (pc - d) \circ (d - td) \\
\texttt{where } (p - name)^{\pi r} == (\text{"}Mike\text{"}, \text{"}Mike\text{"})
\end{array}
$$

A chain of binary relations of objects appears in the *from* clause of the outer query. It must be clarified that (*o-p*), (*o-pc*), (*pc-d*), (*d-td*) are relations of the object database (Figure 4). Clearly the (*p-td*) is the final relation derived by successively applying the composition operator on the *from* clause relations. The (*o - d*) relation, declared in the *from* clause of the inner query, is an intermediate relation derived during the process of the successive application of the composition operation on the chain of relations in the *from* clause of the outer query. Both the relations (*p-td*) and (*o - d*) can be considered as derived by the application of the extended composition operator.

The result of the above query is of the form: *set*<struct((*td*: *TssueDefect*, *td*: *TssueDefect*), set<(type: String, type: String)>)>, where the reference variable *td* can be referred to any object of the *TissueDefect* class hierarchy. The *AcuteTissueDefect* and the *ChronicTissueDefect* object, which either can be referred by the *td* reference, contain detailed information about the tissue defect of the patient. If the query was further declared in detail, all the detailed information contained in the object could be displayed.

4.3.5 Merging of common parts of chains for optimization purposes

In the query:

$$
\begin{array}{l}
\texttt{select distinct stuct } (p - name)^{\pi r}, (td - defectType)^{\pi r}, (c - closureType)^{\pi r} \\
\texttt{from } (o - p)^{-1} \circ (o - pc) \circ (pc - d) \circ (d - td), \\
\qquad (o - c)^{-1} \circ (o - pc) \circ (pc - d) \circ (d - td) \\
\texttt{where } (p - td)^{\pi r} == (c - td)^{\pi r}
\end{array}
$$

two chains of relations, having a common part, appear in the *from* clause. In this case merging the common parts of the single chains should be applied. The result is a unified chain having branches:

$$\texttt{from } [(o - p)^{-1}, (o - c)^{-1}] \circ (o - pc) \circ (pc - d) \circ (d - td)$$

which is equivalent to the following diagram of relations:

$$
\left.\begin{array}{l}
(o - p)^{-1} \\
\\
(o - c)^{-1}
\end{array}\right\rangle (o - pc)(pc - d)(d - td)
$$

where, two branches emerge from the common part of the chain. The unified chain will also be called a *merged* chain. Note that the two relations $(p - td)$, $(c - td)$ appearing in the *where* clause, do not exist among the initial relations of the merged chain. Instead, they are derived as final relations after the successive application of the composition operator upon the unified chain, starting *concurrently* from both the left branches towards the opposite right single end.

The merging of multiple chains with common parts into unified ones plays an important role in the processing of queries. The M^2FE undertakes the processing of all the single and merged chains, applying the extended composition and using parallel techniques. The design and the functionality of the parallelism of this engine are dependent upon the merged chains of relations. Therefore, the algorithm that is mapped upon the M^2FE utilizes the herein introduced merged chains to achieve efficiency.

Even the sequential processing of the chains is better to be applied upon the merged than the single chains because the path of the merged is sorter than the total sum of the paths of the singles. Thus, in both the parallel (M^2FE) and sequential processing of the chains, query optimization can be achieved applying the merging of chains.

The result of the above query is a structure: set <struct ((name: String, name: String), (defectType: String, defectType: String), (closureType: String, closureType: String))>.

4.3.6 Join

Two chains of relations (as singles or merged), appear in the *from* clause of the previous example query. These chains are not directly related; however, they can be *joined* applying the identity expression written in the *where* clause:

$$(p - td)^{\pi r} == (c - td)^{\pi r}$$

where the *td* objects of type *TissueDefect* existing in both relations must be identical. The last expression takes the final form:

$$(td - td)_{(p-td)} == (td - td)_{(c-td)}$$

where both the left and the right sides are derived by the right projections (πr) on the relations $(p-td)$, $(c-td)$, respectively.

4.3.7 Group-by, having and order-by

The last example query can be rewritten as:

$$\texttt{select distinct struct } (p - name)^{\pi r}, (td - defectType)^{\pi r}, (c - closureType)^{\pi r}$$
$$\texttt{from } [(o - p)^{-1}, (o - c)^{-1}] \circ (o - pc) \circ (pc - d) \circ (d - td)$$
$$\texttt{where } (p - td)^{\pi r} == (c - td)^{\pi r}$$
$$\texttt{group by } (c - closureType)^{\pi r}$$
$$\texttt{having } (p - age)^{\pi r} > (\text{"28"}, \text{"28"})$$
$$\texttt{order by } (p - name)^{\pi r}$$

where the operators *group by*, *having* and *order by* appear. Applying the right projection (πr), the last part of the query takes the final form:

$$\texttt{group by } (closureType - closureType)$$
$$\texttt{having } (age - age) > (\text{"28"}, \text{"28"})$$
$$\texttt{order by } (name - name)$$

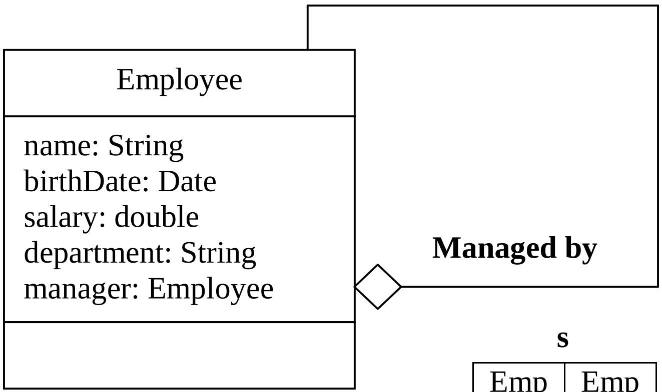
The result of the query is of type: *set* <*struct*((*name*: *String*, *name*: *String*), (*defectType*: *String*, *defectType*: *String*), (*closureType*: *String*, *closureType*: *String*))>,where all the names of the patients above the age of the 28 with the corresponding type of the (tissue) defect and the type of closure applied during the surgical operation are displayed. The results are grouped by the type of the closure, which can have the distinct object values *twoStepClosure*, *delayedClosure* and *primaryClosure* (Figure 3) and are ordered by the names of the patients.

4.3.8 Recursive queries

The need for recursive query declarations in an object oriented database arises from the fact that the architecture of the object oriented system, whose objects are handled by the database, may contain cyclic forms. Cyclic architectures are often met in advanced applications of object-oriented databases; therefore a query model should support recursive queries (Bertino, 1992). Because most of them are *linear* (Bancilhon, 1986), recursive queries of linear type are investigated next.

Let *obj* be an object of the class *Object* and let *var* be a variable of type *Object* (that is able to hold references of instances of *Object* class) defined inside the class. Considering the type *Object* objects, which will be referred to by the variable *var* of *obj* or referred to by the variable *var* of another object *obj1* of type *Object*, which will be referred to by the variable *var* of *obj*, a *direct* recursion can be defined. In this case the class *Object* is mutually recursive with itself. *Employee* is such a class, illustrated below (Figure 5).

Indirect recursive relationships require at least two classes to be involved. Let *a_1, a_2... a_n* be objects of the class *Alpha* and *b_1*, *b_2,... b_n* be objects of the class *Beta*. Let also *obj_a* be a variable of type *Alpha* defined inside the class *Beta* and *obj_b* be a variable of type *Beta* defined inside the class *Alpha*. If the object *b_i* is referred to by the variable *obj_b* of the object *a_i* and the object *a_j* is referred to by the variable *obj_a* of the object *b_(j-1)*, for i = 1,2,…n and j = 2,…n, then an indirect recursion is defined (Figure 6). The two example classes *Device* and *Interface*, illustrated below (Figure 6), are recursively related. At the bottom of the left lower part of figure 6, a real life representation of the recursively related objects of these classes is shown.



Managed by r

Emp	Emp
e0	e1
e1	e3
e2	e4
e3	e6
e4	e7
e6	e8
e9	e10

s

Emp	Emp
e0	e1
e2	e4

t

Emp	Emp
e0	e1
e0	e3
e0	e6
e0	e8
e2	e4
e2	e7

Figure 5. Direct recursive relationships between objects

Within the algebraic framework of the binary relations algebra, the problem of direct recursive queries can be handled as follows:

(I) A binary table can be constructed (relation *r* in Figure 5), where the object pairs derived from the mutually recursive class (Employee in Figure 5), are stored.

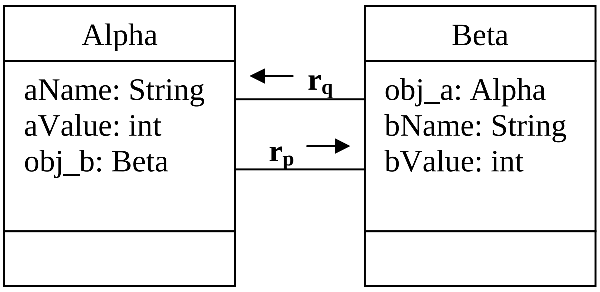
(II) Let *r* be a relation defined as above. It contains recursive relationships of objects of the class, which is mutually recursive with itself. Let *s* (Figure 5) be a sub-relation of *r* ($s \subset r$), whose pair-members contain those left single-members that the objects recursively related to them (stored as right single-members in the pairs of the relation *r*) are required by a query. That is: $\forall(x, y) \in s$, *x* is a left single-member object, *y* is directly related to *x*, and all the recursively related to *x* objects are required.

A *recursive query operator* can be used to find a relation (*t* in Figure 5) containing all the objects that are recursively related to all the first single-members of all the pairs of the relation *s*. The definition of this operator, based on the while construct, will be given further on.

Indirect recursive queries can be handled as follows:

(I) For each connection between the two interconnected classes *Alpha* and *Beta* (upper left part of Figure 6) a binary table can be constructed (upper right part of Figure 6). In the first table $r_p = (A_p\text{-}B_p)$, pairs of the form (*a_i*, *b_i*) where the object *b_i* is referred to by the variable *obj_b* of the object *a_i*, i = 1,…n, are stored, according to the definition of the direction of relationship between two objects (Section 2). Similarly, in the second table $r_q = (B_q\text{-}A_q)$, pairs of the form (*b_(j-1)*, *a_j*) where the object *a_j* is referred to by the variable *obj_a* of the object *b_(j-1)*, j = 2,…n, are also stored. Clearly $(A_p\text{-}B_p) \neq (B_q\text{-}A_q)^{-1}$. The arrow tails of the lower right part of figure 6, indicate the objects whose variables refer to the objects at the arrowheads.

(II) The query can be converted to a single recursive relation *r* problem, and the recursive query operator can be used.

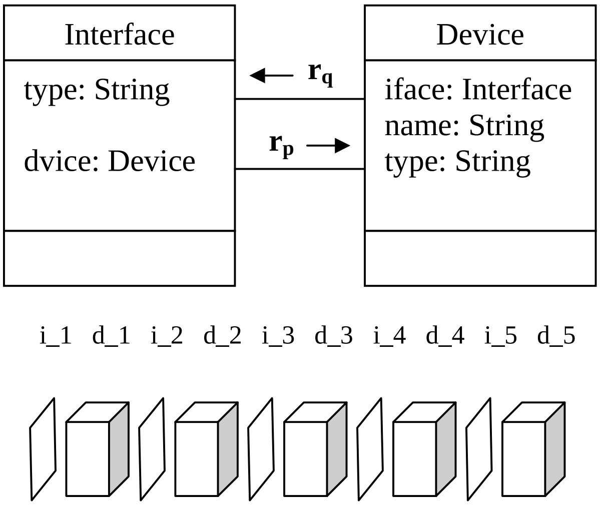


r_p

A_p	B_p
a_1	b_1
a_2	b_2
a_3	b_3
a_4	b_4
a_5	b_5

r_q

B_q	A_q
b_1	a_2
b_2	a_3
b_3	a_4
b_4	a_5



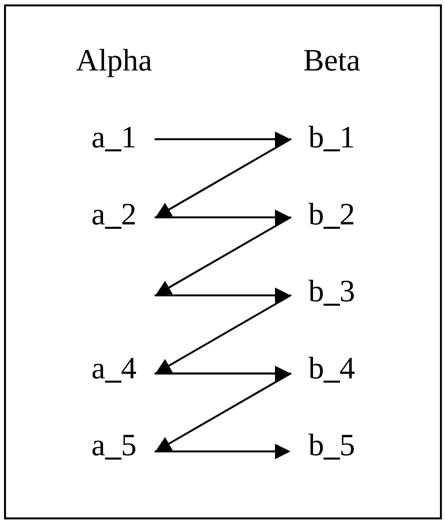


Figure 6. Indirect recursive relationships between objects.

4.3.9 Definition of a recursive query operator

A recursive query operator:

```
recursiveQuery(r : relation, s : relation) : relation
```

is defined as a function of the form:

```
relation recursiveQuery(relation r, relation s) {
            relation t, z;
            t = s;
            z = s;
               while (z ≠ ∅) {
                        z = z ∘ r;
                        t = t ∪ z;
                       }
                return t ;
          }
```

The relation *t*, *z* initially contain only the pairs of the sub-relation *s*. The relation *z* is used to control the loop. The composition $z \circ r$ should signal the end of the procedure producing the empty relation. For each (*x*, *y*) member of the sub-relation *s*, all the pairs having form (*x, v*), where *v* are objects recursively related to *x*, are stored within the relation *t*, which is returned by the recursive operator *recursiveQuery*(*r, s*). In other words, the union $t \cup z$ accumulates all the pairs of the form (*x, v*) within the relation *t*. Note that $s \subset t$.

4.3.10 Use of the operator for direct and indirect recursive queries

Let *r* be the binary relation named *managed by* containing the recursive relationships of the objects of the class *Employee* (Figure 5). Suppose that the set of the names of all the supervisors of all levels of the organizational structure, who are aged over 34, of a specific employee, represented by the object *e0*, is required.

The query can be handled using the operator (function) defined above (Section 4.3.9). For the detection of all the pairs of the form $(e0, ey) \in r$, which contain as left single-members the object *e*0, directly related with any objects *e*y, the following sub query can be used:

```
relation s;
select s
from   r
where  s ⊂ r  and  s^πl == (e0, e0)
```

which returns the subset $s = \{(e0, e1)\}$.

Given *r*, the whole query is:

```
relation s, t;
select (e − name)^πr
from   r,(select s
          from   r
          where  s ⊂ r  and  s^πl == (e0, e0)),
do as t {
         t = recursiveQuery(r, s);
         return t;
         }
where  t^πr == (e − age)^πl  and  (e − age)^πr  > ("34","34")
```

All the pairs of the form (*e*0*, e*z), where *e*z are objects either directly or recursively related to *e*0, are stored in the relation *t*, returned by the recursive procedure. From these pairs, the right single-members, holding values greater than 34, in their variables named *age* are selected and the values held in their *name* variables are projected as pairs. Note that the relation *r* in the from clause, participates in the subquery that returns the relation *s*, and both the relations *r* and *s* participate as arguments in the recursive procedure. It holds $s \subset r$ and $s \subset t$.

The result of the query is of type: *set*<(*name*:*String*, *name*:*String*)> containing the required names of supervisors.

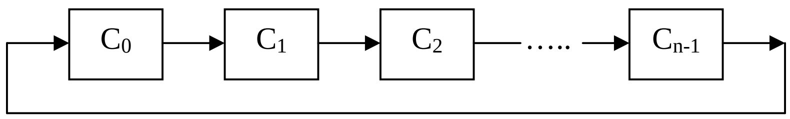
Finally, let r_p *and* r_q be the two binary relations of the indirect recursive relationships between objects illustrated in figure 6. Suppose that the set of all the pairs of the form (*a_1, a_z*), where *a_z* are objects of the class *Alpha* recursively related to *a_1*, are required by a query. Defining $r = r_p \circ r_q$ the query is converted to a single recursive relation *r* problem and the same procedure as above can be followed.

r

A_p	A_q
a_1	a_2
a_2	a_3
a_3	a_4
a_4	a_5

4.3.11 Generalization of recursive relationships

Let the classes C_0, C_1,…C_{n-1}, n>1 participate in an indirect recursive relationship (upper part of Figure 7). For each connection between two successive classes a binary relation (table) is constructed. Let $r_i = (C_i - C_{i+1})$, i = 0, 1,…n-2, and $r_{n-1} = (C_{n-1} - C_0)$ be the constructed relations (lower part of Figure 7). The recursive relationship between the objects of the classes thus formed is called recursive relationship of *order n*. A recursive relationship of order 1 is a direct recursion (Figure 5). For an object c of a class C_i, i = 0,1,…n-1, the recursively related objects at k ($\leq n$ and > 0) *distance* are within the set of the objects of the class $C_{(i+k)\mathrm{mod}(n)}$. These objects are a subset of the right single-members of the pairs of the relation $r = r_i \circ .... \circ r_{(i+k-1)\mathrm{mod}\,(n)}$. Applying the recursive operator to r, the required objects can be found. Therefore, every recursive query of order n > 1 can be converted to a direct recursion (of order 1) problem with a single recursive relation r.



$\mathbf{r}_0$

C_0	C_1
c_0_1	c_1_1
c_0_2	c_1_2
c_0_3	c_1_3
c_0_4	c_1_4

$\mathbf{r}_1$

C_1	C_2
c_1_1	c_2_1
c_1_2	c_2_2
c_1_3	c_2_3
c_1_4	c_2_4

....

$\mathbf{r}_{n-1}$

C_{n-1}	C_0
c_{n-1}_1	c_0_2
c_{n-1}_2	c_0_3
c_{n-1}_3	c_0_4
c_{n-1}_4	c_0_5

Figure 7. Recursive relationship of order *n*

5 PARALLEL MULTIDIRECTIONAL MULTIFUNCTIONAL ENGINE-M²FE

Figure 8 illustrates the binary relations based design of the object oriented database, which supports the workings of the object-oriented system illustrated in figure 3. For every association between two classes in figure 3, a processing node is created, which stores the corresponding binary relation from the figure 4. The multiwavefront algorithm utilizes this design to process the simple and multiple merged chains of relations stated in queries. Thus, the design corresponds to the algorithm architecture, which is finally mapped on a multidirectional multi-functional engine - *M^2FE,* which processes the simple and multiple merged chains of binary relations and synthesizes multiple relations, along multiple directions. Herein, only a short description of the algorithm (and engine) is given.

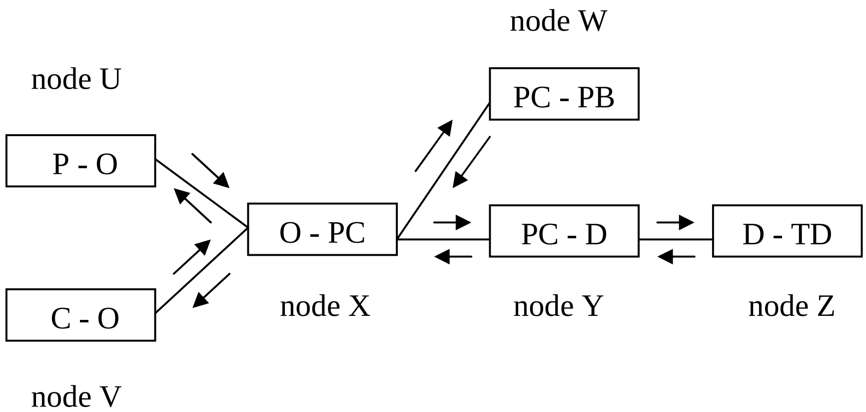


Figure 8. Architecture of the multiwavefront algorithm and engine

The architecture in Figure 8 also represents a multiple merged chain of relations which corresponds to the entire system of Figure 3. Let us assume that the entire system participates in a query. Because all the possible (initial, intermediate and final) relations, derived during the successive application of the composition operator on a chain of relations, are required for query processing purposes (Section 4.3.4), the algorithm *applies* the extended composition operator introduced in the OQL extension, to derive all these relations.

Two types of processors can be used by the processing nodes: a (synchronous) square systolic device for wavefront-like computations - S^2DWC (Fountoukis, 2004) or a bidirectional parallel cubic engine - *BPCE* (Fountoukis, 2005). Both these processors synthesize a relation from two existing ones applying the composition operator (Section 3) defined by the algebra of binary relations. All the relations start moving concurrently from all the nodes (U, V, X, W, Y, Z) towards all possible directions indicated by the arrows of Figure 8. Passing through the nodes, compositions are taking place and new relations are derived, which are also starting to move following the directions of the moving relations that participated in their compositions. Finally all the relations derived by the application of the extended composition are produced and are distributed and stored amongst the processing nodes.

6 CONCLUSIONS

The herein investigated extension to OQL for incorporating algebraic binary relational expressions can be exploited for query declarations in an object oriented database, whose functionality is based upon the algebra of binary relations. In this extension, chains of relations can be stated in SQL syntax based queries. If the chains contain common parts, they can be merged for optimization purposes, and all the relations derived during the successive application of the composition operator on them are considered to be participating in the queries. Therefore, the extended OQL introduces the concept of the extended relation composition. A multiwavefront algorithm, mapped on a Multidirectional Multi-Functional Engine - M^2FE, utilizes the herein introduced concept of extended relation composition and can process the merged chains of relations. The engine produces all the possible relations from the simple or merged chains of relations stated in the queries. The produced relations are distributed amongst the nodes of the processing system; from there they can be retrieved for further processing purposes. In the framework of the algebra of binary relations, the problem of cyclic forms of queries (recursion), which is common in architectures of advanced object oriented software systems, has also been investigated. Further research includes the implementation of the multiwavefront algorithm and the corresponding engine.

7 REFERENCES

Bancilhon F., and Ramakrishnan R. (1986) An Amateur's Introduction to Recursive Query Processing Strategies. *Proc. ACM SIGMOD International Conference on Management of Data.* Washington DC, USA.

Bertino E., Negri M., Pelagatti G., & Sbattela L. (1992) Object Oriented Query Languages: The Notion and the Issues. *IEEE Transactions on Knowledge and Data Engineering. Vol. 4, No. 3*.

Cattell, R.G.G., et al.. (2000) *The Object Data Standard ODMG 3.0.* Morgan Kaufmann.

Desharnais, J. (1989) *Abstract Relational Semantics*, PhD thesis, School of Computer Science, McGill University, Montreal.

Dugundji, J. (1965) *Topology*, Englewood Cliffs, NJ: Prentice-Hall.

Fahmy, H. & Hold, R.C. (2000) Software Architecture Transformations. *Proc. IEEE International Conference on Software Maintenance, Oct 11 - 14*, San Jose, CA, USA.

Fountoukis, S.G. & Bekakos, M.P. (2004) Binary Relational Processing on Wavefront Array Processors. *Proc. The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications. June 21-24,* Las Vegas, Nevada, USA.

Fountoukis S.G. & Bekakos M.P. (2005) Binary Relational Processing on a Bidirectional Parallel Cubic Engine. *International Journal of Neural, Parallel and Scientific Computations, Vol. 13, No 3-4, pp. 437 – 454.*

Givant S.R. (1994) The Structure of Relation Algebras Generated by Relativizations. *Contemporary Mathematics.* American Mathematical Society: Rhode Island, USA.

Gyssens, M., Paredaens, J., Van den Bussche, J., & Van Gucht, D., (1994a) A Graph Oriented Object Database Model. *IEEE Transactions on Knowledge and Data Engineering, 6.4, pp. 572-586.*

Gyssens, M., Saxton, L., & Van Gucht, D. (1994b) Tagging as an Alternative to Object Creation. In Freytag, J., Majer, D., & Vossen, G. (eds.) *Query Processing for Advanced Database Systems.* Morgan Kaufmann.

Hold, R.C. (1998) Structural Manipulation of Software Architecture Using Tarski Relational Algebra. *5th WCRE, Oct 12-14,* Honolulu, Hawai, USA.

Kelley, J. L. (1975) *General Topology*, New York: Springer-Verlag.

Sarathy, V., Saxton, L., & Van Gucht, D. (1993) Algebraic Foundation and Optimization for Object Based Query Languages. *Proc. Ninth International Conference of Data Engineering.* IEEE Computer Society Press: Vienna, Austria.

Tarski., A. (1941) On the Calculus of Relations. *Journal of Symbolic Logic. 6, pp. 73-89.*

Van den Bussche J. (2001) Applications of Alfred Tarski's Ideas in Database Theory, *Journal of Lecture Notes in Computer Science, 2142, pp. 20-37.*