ICOT Technical Report: TR-519

TR-519

Towards Integration of Deductive Databases and Object-Oriented Databases: A Limited Survey

by
K. Yokota & S. Nishio (Osaka Univ.)

November, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

TOWARDS INTEGRATION OF DEDUCTIVE DATABASES AND OBJECT-ORIENTED DATABASES: A Limited Survey*

Kazumasa Yokota
Institute for New Generation Computer Technology (ICOT)
1-4-28, Mita, Minato-ku, Tokyo 108, Japan
e-mail: kyokota@icot.jp

Shojiro Nishio
Dept. of Information & Computer Sciences
Osaka University
Toyonaka, Osaka 560, Japan
e-mail: nishio@osaka-u.ac.jp

ABSTRACT

There have been many proposals for extensions of relational databases. It can be considered that two major streams of them are DDBs and OODBs. Research on fusion or integration of their advantages has been being done at the forefront of research on database and logic programming. For the target, the first international conference on deductive and object-oriented databases (DOOD89) was held in Kyoto.

In this paper, we propose a framework for the extensions along these lines, and introduce some research results such as ψ-term, O-logic, F-logic, DOT and hierarchical deductive databases, in the framework, and discuss future directions.

1 Introduction

As extensions of relational databases or new databases (data models), there have been many proposals such as deductive databases (DDBs), nested relations, complex objects, semantic data models, and object-oriented databases (OODBs). As their background, there are various reasons such as maturity of relational database technologies, enlargement of application domains, development of hardware technologies, influence of programming paradigms, development of knowledge information processing technologies, and problems of impedance mismatch. By combining these, the requirements of new databases have been increased.

It can be considered that two major streams of active research on databases are DDBs and OODBs. The advantages of DDBs are high inference capability and well formal foundations; the disadvantage is that the modeling capability is poor and there are few applications. With OODBs, as the advantages are rich modeling capability and high extensibility, so they are expected to be adaptable for many applications. As for their disadvantages, the remarkable point is that there is not necessarily a consensus on the concept of the data model, although the conventional concept itself has been changed. Also the inference capability is poor and formal foundations are not well, compared with DDBs.

In view of these mixed results, the next generation of intelligent databases must adopt the advantages of both streams, on well formal foundations. We call them *deductive and object-oriented databases* (DOODs).

We describe the research framework of DOODs as extensions of DDBs in Section 2.

There are two aspects of an object-orientation paradigm: passive and active aspects. We give some examples to explain the embedding characteristics of passive objects into DDBs in Section 3, and about some characteristics of DDBs as active objects in Section 4. Then we overview perspectives on future research directions on DOODs in Section 5.

2 Framework of Deductive and Object-Oriented Databases

In order to mitigate various disadvantages of DDBs, as described in the previous section, there have been many proposals for their extensions. We can classify them as follows:

(1) Logical Extensions
Introduction of more logical factors such as negation, disjunction, explicit or implicit existential quantifiers, and certainty factors.

(2) Encapsulary Extensions

a. Structural Extensions
Introduction of structured data such as nested relations and complex objects, that is, extensions of term representation to be appropriate for the above structured data.

b. Procedural Extensions
Encapsulation of data and procedures, that is, behaviors of the data are described in an object.

*Japanese Title: 演繹・オブジェクト指向データベースの構築に向けて (最近の研究成果から)

c. Introduction of Object Identity
Direct representation of objects by object identifiers, that is, an object identifier itself is captured as invariant through state changes.

(3) Paradigmatic Extensions

a. Constraint Logic Programming Paradigm
Query processing by resolution and constraint solvers of a constraint logic programming language based on objects.

b. Object-Orientation Paradigm
Query Processing by message passing between objects and internal processing in objects.

As for (1), there have been many works such as stratified databases, disjunctive databases, and quantitative DDBs. Also as for (2), there have been many results as explained in Section 3. However, as for (3) there have been only few works. As each of (1), (2) and (3) is an independent extension from others, they can be combined at various levels. For integration of deductive databases and object-oriented databases, we can focus mainly on the *object-oriented extensions*, that is, (2) and (3)*b*. In this paper, we consider deductive and object-oriented databases (DOODs) in such a formal framework as extensions of DDBs.

Before explaining them, we consider some characteristics of object-oriented databases (OODBs) as one of the components of DOODs. With the background mentioned in the previous section, OODBs have been affected by object-oriented programming languages such as Smalltalk-80. However, there is not necessarily a consensus on each feature, although there have been some efforts [5] to reach the common agreement. In this paper, we consider the term 'object-oriented' or 'object-orientation', as being how to reflect the object-orientation paradigm, rather than how many conditions listed as 'necessary conditions' are to be satisfied. In our viewpoints, we classify the concept of object-orientation into two aspects: *passive objects* for natural modeling of entities and *active objects* for natural modeling of computational process.

How entities in the real world could be represented in the symbolic world has been considered as one of main purposes in a database area, so most OODB researchers have tended to focus on the passive aspect of objects. The characteristics of the passive aspect are as follows (although some of them are related to the active aspects): object identity, complex objects, methods and information hiding, types and classes, and hierarchical structures. However, even in each characteristic there are many disagreements, as many of them depend also on application domains. On the other hand, one of the most important characteristics of the active aspects is autonomy of each object for query processing and database management with the following properties: message passing, cooperative problem solving, heterogeneity of each object, and concurrency, recovery and persistence control by each object, which also correspond to the concept of autonomy discussed in the context of distributed or parallel database management systems. It is a major problem how to combine these two aspects in the formal framework of DDBs.

Like some OODBs in the commercial market, we can consider a database management system which manages only passive objects. However, under the term 'object-orientation', we should consider both aspects of objects. In this sense, OODBs could be discriminated from 'object bases' and 'abstract data type bases'. These aspects are not independent but mutually related in many features such as a type hierarchy, methods and message passing, interpretation of procedures, and management facilities. They correspond to encapsulary extensions and *b* of paradigmatic extensions in the above object-oriented extensions, and can be combined at various levels. Although many works have been done on each feature, it is very important to consider them uniformly as research on DOODs in such a framework as the above.

3 Embedding Object-Orientation Concepts into Deductive Databases

A basic unit of representation in deductive databases (DDBs) is a first order term (or an atomic predicate). In order to introduce various features of passive objects into DDBs, how to extend term representation is important. In this section, we describe the main problems to be considered in term representation, and introduce some research results from viewpoint of how they could be embedded.

3.1 Problems to be Considered

What we should consider here is not the capability of term representation but the efficiency. In this sense, many researchers have pointed out inefficiency of a first order term, such as fixed positions and arity of arguments. In order to improve it by introduction of an object-orientation concept, we can list the following problems in term representation:

- Tuple Representation
  To deduce restrictions of positions and arity of arguments by tuple representation in attribute-value pair notation, and make it possible to represent also infinite data structure.
- Type Hierarchy
  To introduce types (or classes) and the hierarchies in order to reduce redundancy between expressions. There

is not necessarily a consensus on a concept of types (or classes) and the kinds of hierarchies.

- Set Representation
  To make it possible to represent complex objects by introducing set constructors as well as tuple constructors. Whether sets are objects or not, especially whether the semantics is higher order or not, has been discussed.

- Object Identity
  To specify an object by an object identity, make the sharing possible, and make dependency relationships between objects clear. It is also extensions of identifiers in tuple representation. There are many variants in treating identifiers.

- Method
  To define methods to an object and the corresponding procedures. What kind of languages should be used to represent the procedures has been discussed.

- Other Extensions
  There are many other problems such as differences between types and objects, discrimination between individual and set values, semantics of inheritance and kinds of constructors.

3.2 ψ-Term — Tuple Representation with Type Hierarchy [3,4]

Data representation in tuple representation has been done as research not only in the database area, but also in knowledge representation language and feature structures in natural language processing. Aït-Kaci [3] proposed *ψ-terms*, which is considered as precise formalism and is summarized as follows:

(1) Types are represented as tuples called ψ-terms. The construction is aimed at overcoming the weak points of first order terms, already mentioned. Consider the following example (a ψ-term for person information):

$$\begin{aligned} X : per[&id \Rightarrow name[last \Rightarrow Y : str,\\ &\qquad first \Rightarrow str],\\ &addr \Rightarrow \top,\\ &father \Rightarrow per[id \Rightarrow name[last \Rightarrow Y]],\\ &spouse \Rightarrow per[spouse \Rightarrow X]] \end{aligned}$$

'[' and ']' are tuple constructors. In '[' and ']', the right hand side of $\Rightarrow$ is an attribute name and the left hand side is a type. For example, '*per*' (person), '*name*', '*str*' (string) and '$\top$' are types, and '*id*', '*last*', '*first*', '*addr*' (address), '*father*' and '*spouse*' are attribute names. 'X' and 'Y' at the left hand side of ':' are called *tags*, and represent equality constraints of types. For example, in the above example, a *per*'s *last* name should be same as her/his *father*'s, and s/he should be a *spouse* of her/his *spouse*.

Formally, ψ-term is defined as a triple (Δ, ψ, τ), where Δ is a tree domain constructed by a set of attribute names including an empty attribute ϵ, ψ is a function from Δ to a set of types, and τ is a function from Δ to a set of tags. The above example can be pictured in **Figure 1**, where tags Z_i are omitted in the above expression.

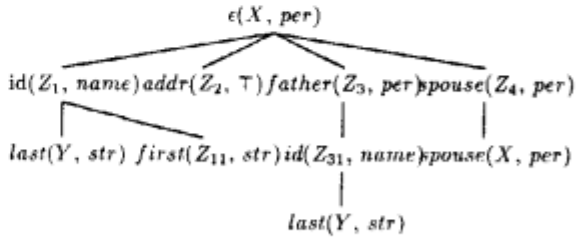

Figure 1. Structure of ψ-term

(2) The original intention of ψ-terms is to formulate inheritance. For the purpose, subsumption (partial order) relationship is extended to the relationship between ψ-terms, and results in a lattice. For example, partial order relationship between types such as $student \preceq per$, $employee \preceq per$, and $grad\_student \preceq student$, is used to inherit attributes of upper types. $\top$ in the example of (1) shows top of this lattice. The join operation in the lattice, corresponding to unification in first order terms, results in inheritance of attributes.

Furthermore, [3] continues the formulation as follows:

(3) Algorithm of join operation unification between ψ-terms is provided.

(4) ψ-terms are extended by introducing set notation called ϵ-terms, which is used only as abbreviation of a set of ψ-terms.

(5) A new programming language called KBL is defined as a term rewriting system of ψ-terms based on (2), (3) and (4).

Although ψ-terms contribute to solve only a few problems in Section 3.1 from an object-oriented database (OO-DB) point of view, and are not treated from a database point of view, the work contributes to works on object-oriented extensions of DDBs such as embedding ψ-terms into definite clauses [4, 8], application to complex objects [6], and some extended term representation described in succeeding subsections.

3.3 O-Logic [16,18]

Maier [18] made clearer the concept of object identity which had not been discussed consciously in ψ-terms, proposed *O-terms*, similar with ψ-terms, in which rules can be written, and constructed *O-logic* under the term representation. In the beginning of the paper, he declared that the objectives of O-logic are to provide formal foundations for fusion of a logic for data-intensive applications and object-oriented programming. However, the paper is confined within a proposal of O-logic and suggestion of problems such as the semantics and the treatment of quantifiers, sets and inconsistency. And O-logic itself is first completed theoretically by the reconstruction by Kifer and Lu [16]. The proposal of O-logic has much influence on the succeeding research on DOODs, and it can be said that the idea is a landmark. The following topics are based on the revisions by Kifer and Lu, which can be briefly summarized as follows:

(1) Treatment of a set of objects as an attribute value. The semantics is devised to be first-order [10].

(2) Introduction of a lattice structure on a set of objects and inheritance of attributes from the upper types when the corresponding attributes are not specified.

(3) Substitution of "inconsistency value" of the lattice for logical inconsistency of attribute values, which results in local inconsistency and no influence on inference based on other normal values.

(4) Sound and complete proof procedure based on resolution.

O-terms are recursively defined by a set of objects without internal structures, that is, atomic objects. The terms are based not only on atomic objects, but also on object variables, by which equality constraints as in ψ-terms can be represented. In O-terms, besides a set of types, a set of labels (attribute names) is introduced and each label is interpreted as a function from a set of objects to a set of attribute values or the power set. In O-logic, more complex representation can be constructed by combination of single or multiple O-terms with logical connectors such as $\vee$, $\wedge$, $\Leftarrow$ and $\neg$, and quantifiers such as $\forall$ and $\exists$.

First, as an O-term without object variables, consider the following example of an O-term representing an object 'mary' which belongs to a type 'employee':

```
employee:mary[name→str:"Mary",
              age→int:30,
              hobby→{game:chess, game:tennis},
              office→ faculty:CS2[f-name→
                                  str:"com.sci."]]
```

where the left hand side of ':' is a type, and the right hand side is an object; and the left hand side of '→' is a label corresponding to an attribute name, and the right hand side is an attribute value. The representation allows a set value such as a value corresponding to 'hobby', and it is easy to represent complex objects in O-logic.

In O-terms, object identifiers can be constructed by combination of atomic objects and object variables by object constructors (if necessary, by applying different function symbols repeatedly). Especially, such object identifiers are called *id-terms*. In order to understand how id-terms work, consider the following O-term:

$$\text{arc}{:}E[\text{start}{\rightarrow}\text{node}{:}X,\ \text{end}{\rightarrow}\text{node}{:}Y]$$

which represents a relation of directed arcs between nodes, that is, a directed graph. In the graph, a set of paths between nodes is recursively defined by the following rules with id-terms:

$$\begin{aligned}&\text{path}{:}add(E,nil)[\text{start}{\rightarrow}X,\ \text{end}{\rightarrow}Y] \Leftarrow \\ &\qquad \text{arc}{:}E[\text{start}{\rightarrow}\text{node}{:}X,\ \text{end}{\rightarrow}\text{node}{:}Y] \\ &\text{path}{:}add(E,P)[\text{start}{\rightarrow}X,\ \text{end}{\rightarrow}Y] \Leftarrow \\ &\qquad \text{arc}{:}E[\text{start}{\rightarrow}\text{node}{:}X,\ \text{end}{\rightarrow}\text{node}{:}Z], \\ &\qquad \text{path}{:}P[\text{start}{\rightarrow}\text{node}{:}Z,\ \text{end}{\rightarrow}\text{node}{:}Y]\end{aligned}$$

where $f \Leftarrow g$ means implication as "if g then f". In the example, $add(E, nil)$ and $add(E, P)$ are id-terms. The first rule means a set of paths constructed by only one arc, and the second means a set of paths constructed by an arc and the adjacent existing path. Therefore the two rules define a set of all paths. For a query that asks a set of all paths,

$$\text{path}{:}P[\text{start}{\rightarrow}\text{node}{:}X,\ \text{end}{\rightarrow}\text{node}{:}Y]?$$

a set of answers is returned in the following form:

$$\text{path}{:}add_{1,k}[\text{start}{\rightarrow}\text{node}{:}a_i,\ \text{end}{\rightarrow}\text{node}{:}b_i]$$

where $add_{1,k}$ means $add(e_1, add(e_2, \cdots, add(e_k, nil), \cdots))$, e_j is an object identifier of an arc, and a_i and b_i are identifiers of nodes.

In this way, id-terms play an effective role to enhance representative capability of O-terms.

3.4 F-Logic [15]

Kifer and Lausen [15] proposed Frame-logic (shortly *F-logic*), which is based on a concept of frames, used as one of knowledge representation languages in artificial intelligence, and is an extension to O-logic: all expressions in O-logic can be written in F-logic. Like frame-based representation, there are no differences between types and entities (objects) in *F-terms*, which are basic components of F-logic. Furthermore, as id-terms with object variables can be used

as labels (that is, labels also can be treated as objects), *methods* can be defined declaratively.

Consider the following example of method description in F-logic:

$$X[\mathrm{chi}(Y) \rightarrow \{Z\}] \Leftarrow \\ \mathrm{per}{:}Y[\mathrm{chi\_obj} \rightarrow \mathrm{chi}(Y)[\mathrm{member} \rightarrow \{\mathrm{per}{:}Z\}]], \\ \mathrm{per}{:}X[\mathrm{chi\_obj} \rightarrow \mathrm{chi}(X)[\mathrm{member} \rightarrow \{\mathrm{per}{:}Z\}]]$$

This rule defines a method 'chi' (child) as follows: for each 'per' (person) X and a substitution of a person for an object variable Y, the rule returns a set of all children common to persons X and Y. It is remarkable that a term 'chi(Y)' appears not only as method description in the location of an attribute label in the head of the rule, but also as an id-term in the location of an attribute value of a label 'chi' of Y in the body of the rule. Therefore, for example, the instantiated id-term 'chi(mary)' has two meanings: an object representing all children of mary; a function from a given person to a set of all children common to mary and the person. In this way, by changing the meaning according to the location, labels also can be treated as objects.

While the syntax of F-terms is high-order as in the above example, the semantics is kept to be first-order, just like as O-terms [10].

3.5 DOT — Extended Terms Based on Dot Notation [22]

There are various ways to specify attributes of an type, and, as the result, there could be various patterns of inheritance between types.

For example, assume an IS-A relationship

'student IS-A person',

and an attribute about affiliation of a person such as

'a person is affiliated with an organization'.

And then, if

'a student is affiliated with a university'

is written explicitly as an attribute 'affiliation' of the student, a relationship

'university IS-A organization'

can be considered to be derived by inheritance (although, strictly speaking, there might be some problems). However, if an attribute 'affiliation' of the student is not written explicitly, only a relationship such as

'a student is affiliated with an organization'

is obtained.

Therefore, in DOT [22], even if an attribute 'affiliation' of a student is not written explicitly, it is possible to introduce a *virtual object* (actually, a type or an entity) about the attribute, and derive a new IS-A relationship between objects by using the virtual one as a medium of IS-A relationships. A dot-based representation DOT was proposed [22]. In it, there are no differences between types and entities, and it is easy to write inheritance and IS-A relationships between objects by introducing new object representation including virtual ones.

In DOT, an attribute value of an attribute p of an object a is represented abstractively as $a.p$, whether it is explicitly written or not. Such representation is called a *DOT formula*. A dot notation has been used to represent an attribute value in fields such as a relational database, while in DOT, a DOT formula itself is treated as an object. In the above example, it is possible to write a virtual object such as 'student.affiliation', by which it is possible to represent a new relationship such as

'student.affiliation IS-A person.affiliation',

even if the attribute 'affiliation' of the student is not described explicitly. And, if

'a student is affiliated with a university'

would be described explicitly, the attribute value could be written by the IS-A relationship as follows:

university IS-A student.affiliation

And the value 'university' is inherited one after another by the following IS-A relationships. However, such an IS-A relationship can be written in the following one sentence:

if a IS-A b then $a.p$ IS-A $b.p$

In conventional languages, it is impossible to represent indirectly an object such as 'student.affiliation', and it is not so easy to represent such an inheritance.

As there are no differences between entities and types in DOT representation, the IS-A relationship includes not only usual IS-A relationship but also a relationship between a set and the element as the extension.

Then, consider an example of term representation. For example, 'per' (person) is described in DOT as follows:

per(par→{per},
    name→{str},
    age→{int},
    sex={make, female})

The direction of an arrow corresponds to the direction of an IS-A relationship (for example, per.par IS-A per, that is, a person's parent IS-A a person), and '=' means a bi-directional IS-A relationship. In the above example, the fact that there are only 'male' and 'female' as a kind of 'per's 'sex' corresponds to a bi-directional IS-A relationship. A person 'Mr-A' is represented as follows:

$$\begin{aligned}\text{Mr-A:}\{\text{per}\}\{&\text{name}=\{\text{paul}\},\\ &\text{age}=\{24\},\\ &\text{sex}=\{\text{male}\},\\ &\text{par}\leftarrow\{\text{john}\},\\ &\text{par}\leftarrow\{\text{mary}\}\}\end{aligned}$$

A part of the IS-A relationships is shown in **Figure 2**.

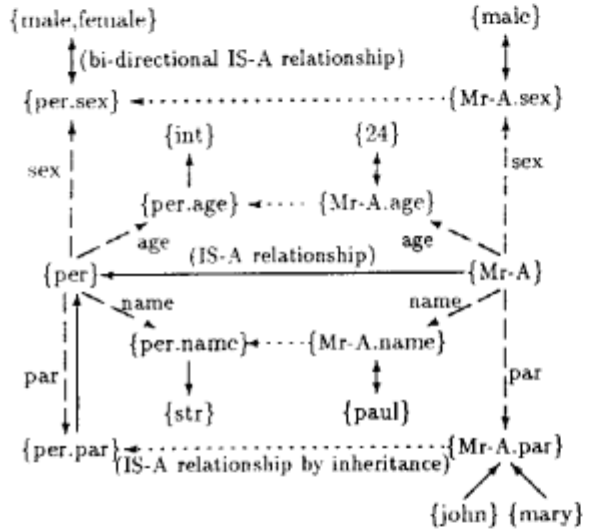

Figure 2. Example of Terms

An arc written as a straight line which means an IS-A relationship is called an *IS-A link*. An arc written as a dotted line which corresponds to an attribute is called an *l-link*. The attribute value corresponding to an attribute l of an object a is represented in a DOT formula $a.l$, and the attribute value is exactly represented by an IS-A relationship from the abstractively represented object. For example, let the attribute value of an attribute 'par' of 'Mr-A' be abstractively 'Mr-A.par', and draw an IS-A link such as 'john IS-A Mr-A.par'. Dotted arcs in the figure show such derived IS-A relationships.

As IS-A relationships are an order relationship, new IS-A relationships are derived by tracing IS-A relationships transitively. In **Figure 2**, for example, an IS-A relationship 'john IS-A per' is derived by tracing IS-A relationships: from 'john' to 'Mr-A.par', from 'Mr-A.par' to 'per.par', and from 'per.par' to 'per'.

And, as a query in the term representation, a 'par' of 'Mr-A' is obtained as a 'per' of 'Mr-A' by tracing IS-A links from 'Mr-A.par', and a 'par' of 'Mr-A' is obtained as 'john' and 'mary' by tracing backwards IS-A links into 'Mr-A.par'.

Although not explained in the above examples, it is possible to express DOT terms with object variables, and also to describe rules as in Section 4.2.

4 Deductive Databases as Active Objects

Even if deductive databases (DDBs) are written in any logic programming language, they can be considered to correspond to objects which represent (abstract) entities consisting of facts and rules. Such objects can be considered also as modules of knowledge. In this section, we consider a DDB as such an object, and query processing to a set of the objects. It is natural to introduce inheritance and overriding mechanism between such objects, considered in the context of an object-orientation paradigm. Furthermore, as each DDB has the ability to process queries, the query processing to a set of objects can be treated as cooperative problem solving between objects. In this section, first we embed a set of DDBs into a hierarchy, and secondly consider query processing mechanism by active objects. As its applications, there might be knowledge bases with modules of knowledge, CAI system with development processes of students, and hypothetical reasoning system in expert systems.

4.1 Problems to be Considered

As a DDB is not defined as an object as it is, and not given any relationship to other objects, its representation should be extended:

- Object Identity
  To introduce object identity into objects. A DDB itself or a definition of a predicate can be considered as a unit of an object.

- Methods
  Predicates defined in an object can be considered as methods of the object. There are some discussions such as name spaces of predicates, and multiple definitions of a predicate.

- Hierarchical Relationship
  To represent relationships between objects and inherit properties dynamically in order to reduce redundancy of knowledge between objects.

- Cooperative Problem Solving
  To execute query processing to an object cooperatively by message passing between objects.

- Others
  There could be many other extensions such as local knowledge and coexistence of multiple languages.

We assume that each DDB has a capability of concurrency, recovery and persistence controls, and we do not describe here the synchronous mechanism, although it is very important.

4.2 Deductive Databases Embedding into a Hierarchy

In order to embed a set of DDBs (or definitions of a predicate) into a hierarchy, assume a set I of object identifiers and a set R of facts and rules. A function ρ from I to R defines a set of DDBs with object identifiers. And, by introducing a partial order relationship $\preceq$ into I, $(I, \preceq, \rho)$ defines a set of DDBs embedded into a 'hierarchy'. For example, **Figure 3** shows such a set, where $K_i (1 \leq i \leq 7)$ is an object identifier and each link is a partial order relationship between DDBs.

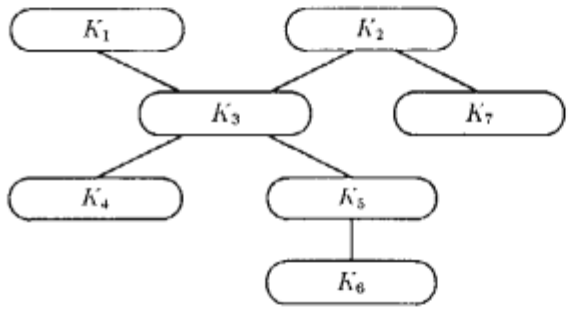

Figure 3. Hierarchy of Deductive Databases

If $S_i = \{K_j | K_i \leq K_j\}$, a DDB $\rho(K_i)$ corresponding to K_i can be considered to mean $\bigcup_{K_j \in S_i} \rho(K_j)$ from a simple inheritance point of view, although there is a problem about how to guarantee some restrictions of each DDB, such as self-containedness of the subgoals and stratification in case of a DDB with negation. Dynamic verification mechanism for such restrictions might be needed. Furthermore, if a predicate is not defined over multiple objects, an overriding mechanism can be introduced naturally as well as inheritance.

It is easy to understand that the formalism can be applied to DDBs consisting of extended terms in Section 3 with object identifiers. For example, consider an example in F-logic

$$\begin{aligned} &\mathrm{per}[\mathrm{sg}(X) \rightarrow \{X\}] \\ &\mathrm{per}[\mathrm{sg}(X) \rightarrow \{Y\}] \Leftarrow \mathrm{per}{:}X[\mathrm{par} \rightarrow \{Z\}], \\ &\qquad\qquad\qquad\quad \mathrm{per}{:}Z[\mathrm{sg}(Z) \rightarrow \{W\}], \\ &\qquad\qquad\qquad\quad \mathrm{per}{:}W[\mathrm{chi} \rightarrow \{Y\}] \end{aligned}$$

or an example in DOT

$$\begin{aligned} &X(\mathrm{sg} \leftarrow X, \\ &\quad \mathrm{sg} \leftarrow X.\mathrm{per.sg.chi}) \Leftarrow X : \mathrm{per} \end{aligned}$$

which defines a 'per' (person) object with a method 'sg', and inheritance relationship between objects (types) by the partial order relationship. There are two possibilities for introduction of the above function ρ: an original object identifier corresponds to a new object; a new object identifier defines a new object consisting of multiple original objects, where there is a problem between an original and a new order relationships. In both cases, a DDB is divided into a set of DDBs, and embedded into a hierarchy.

4.3 Query Processing Based on Message Passing

In the example of **Figure 3**, assume that each object can perform query processing of a DDB, which inherits dynamically rules from the upper objects according to the necessity. Extensional databases (EDBs) might be divided horizontally. In the case, assume that the query processing to an object is controlled at the EDB level by the object according to the necessity.

Consider a query processing to an object K_5. As K_5 inherits rules of K_1, K_2, and K_3, necessary rules for the processing should be gathered dynamically into the lowest object which defines the corresponding predicate.

Consider bottom-up evaluation such as HCT/R [19] as a strategy of recursive query processing. By propagating binding information in the goal, recursive rules are transformed to restrict a search space, and the transformed rules except initial clauses (seeds) generate new objects corresponding to components, each of which is closed within recursive relationship. Such new objects are generated with a specification relationship to the lowest object which defines the predicate, and might be reused for another query processing with the same binding pattern. These objects play a role of coordinator objects for the related objects (C_1-C_5 in **Figure 4**).

Bottom-up evaluation is triggered by giving a (sub)goal and initial clauses to coordinator objects, and query processing of EDBs is divided into local processing in each object and coordinator processing. The final answer results in the lowest object (C_1 in this case).

While such query processing is executed by message passing between objects, other processing such as concurrency control and memory management in traditional database management systems could be also done autonomously by each object.

Besides such bottom-up evaluation, [12] proposed an al-

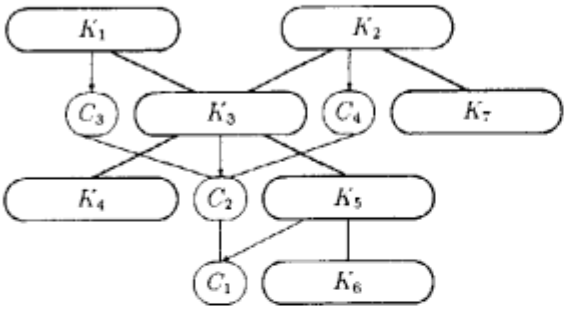

Figure 4. Coordinatee Object for Query Processing
($\rightarrow$ is a specification relation)

gorithm in top-down manner by generating dynamically processes corresponding to definitions of predicates in a parallel processing environment.

5 Perspectives

Besides deductive and object-oriented databases (DOODs), there are many approaches for extensions of deductive databases (DDBs), already mentioned in Section 2. In this section, we introduce such approaches and discuss research perspectives on new databases.

5.1 Other Extensions

Research on extensions of DDBs has been spreading from various points of view. We describe some approaches.

- Other Logic Programming Languages Based on Extended Terms
  As other extensions of DDBs, there are LDL [25], which introduces set grouping into logic programming, COL [1], which manipulates complex objects, and CRL [23], which works on nested relations. Furthermore, along with O-logic and F-logic, HiLog [10] is proposed. The syntax of the language is higher-order, while the semantics is first-order. Although the language is based on predicate notation, it offers more efficient representation than LDL, COL and F-logic from another point of views, and transformation from LDL, COL, and F-logic to HiLog is shown in [10].

- Database Programming Languages
  For the purpose to integrate database languages and programming languages, there are many works on database programming languages (DBPLs) [13]. This approach includes not only logic programming approaches but also functional and object-oriented programming ones. The purpose of DBPLs is deduction of impedance mismatch, which is also one of the purposes of object-oriented databases (OODBs). Therefore the languages mentioned above are also considered as kinds of DBPLs.

- Constraint Logic Programming Paradigm
  As an extension of logic programming languages, there are constraint logic programming languages [14], which intend to represent complex problems as constraints and solve them by intrinsic constraint solvers. Under the paradigm, there are some works: catching ψ-term as one of constraints [9] and approaches for application to database area [11, 24].

Although we focus on object-oriented extensions of DDBs from a DOOD point of view in this paper, we should discuss next generation databases from a wider point of view, such as the above approaches and the framework mentioned in Section 2.

5.2 Research Directions

It can be said that research on DDBs started at *The International Workshop on Logic and Databases* in Toulouse, France, 1977, and reached a big turning point at *Workshop on Foundations of Deductive Databases and Logic Programming* in Washington DC, 1986.

In the same year, the first meetings were held for new database management systems: for OODBs, *The International Workshop on Object-Oriented Database Systems* and *The Symposium on Object-Oriented Programming Systems, Languages and Applications* (OOPSLA), and for integration with expert systems, *The First International Conference on Expert Database Systems*. The next year, the first *Workshop on Database Programming Languages* was held. We could say that these streams show the opening of multiple paradigms for new database management systems at the end of 1980's.

With such a background, *the First International Conference on Deductive and Object-Oriented Databases* (DOOD-89) [17] was held in December, 1989 in Kyoto, focusing on integration of DDBs and OODBs, that is, DOODs, as one of the powerful candidates of new generation databases and moving towards construction of the framework of next generation databases by active discussion with both researchers on DDBs and OODBs [2, 5, 7, 26].

We confirm that the framework of DOODs will play an more important role in research and development in the database area.

6 Concluding Remarks

Research on deductive and object-oriented databases (DOODs) has been very active in extensions of deductive databases, although it is only very recently that it has been

focused on as one of the new generation databases. As DOODs can be considered as a research framework for a new generation database system, as mentioned in Section 2, and do not constrain the concept uniquely, various kinds of DOODs depending on problem and application domains could coexist.

Acknowledgments

The authors would like to thank members of the deductive and object-oriented databases working group of ICOT for stimulative and valuable discussions.

References

[1] S. Abiteboul and S. Grumbach, "COL: A Logic-Based Language for Complex Objects", *INRIA Technical Report*, 714, 1987.

[2] S. Abiteboul, "Towards Deductive Object-Oriented Databases", *Proc. DOOD89*, Kyoto, Dec., 1989.

[3] H. Aït-Kaci, "An Algebraic Semantics Approach to the Effective Resolution of Type Equations", *Theor. Comput. Sci*, vol.45, pp.293-351, 1986.

[4] H. Aït-Kaci and R. Nasr, "LOGIN: A Logic Programming Language with Built-in Inheritance", *J. Logic Prog.*, vol.3, pp.185-215, 1986.

[5] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik, "The Object-Oriented Database System Manifesto", *Proc. DOOD89*, Kyoto, Dec., 1989.

[6] F. Bancilhon and S. Khoshahian, "A Calculus for Complex Objects", *Proc. ACM PODS*, pp.53-59, Cambridge, 1986

[7] C. Beeri, "Formal Models for Object-Oriented Databases", *Proc. DOOD89*, Kyoto, Dec., 1989.

[8] C. Beeri, R. Nasr and S. Tsur, "Embedding ψ-term in a Horn-clause Logic", *Proc. the Third International Conference on Data and Knowledge Bases*, pp.347-359, Jersalem, June, 1988

[9] H. Beringer and F. Porcher, "A Relevant Schema for Prolog Extensions: CLP (Conceptual Theory)", *Proc. ICLP*, pp.131-148, Lisbon, June, 1989.

[10] W. Chen, M. Kifer and D.S. Warren, "HiLog as a Platform for Database Language", *Proc. the Second International Workshop on Database Programming Language*, pp.121-135, Gleneden Beach, Oregon, June, 1989.

[11] H. Gallaire, "Multiple Reasoning Style in Logic Programming", *Proc. FGCS'88*, pp.1089-1099, Tokyo, Nov. 28-Dec.2, 1988.

[12] G. Hulin, "Parallel Processing of Recursive Queries in Distributed Architectures", *Proc. VLDB*, pp.87-96, Amsterdam, Aug., 1989.

[13] R. Hull, R. Morrison and D. Stemple (ed.), *Proc. of Second International Workshop on Database Programming Language*, Gleneden Beach, Oregon, June, 1989.

[14] J. Jaffer, J.-L. Lassez et al, "Constraint Logic Programming", *IEEE SLP*, San Francisco, Aug.31-Sep.4, 1987.

[15] M. Kifer and G. Lausen, "F-Logic — A Higher Order Language for Reasoning about Objects, Inheritance, and Schema", *Proc. ACM SIGMOD*, pp.134-146, Portland, June, 1989.

[16] M. Kifer and J. Wu, "A Logic for Object-Oriented Logic Programming (Maier's O-Logic: Revisited)", *Proc. ACM PODS*, pp.379-393, Philadelphia, Mar., 1989.

[17] W. Kim, J.-M. Nicolas and S. Nishio (ed.), "Deductive and Object-Oriented Databases", *to be published*, North-Holland, 1990.

[18] D. Maier, "A Logic for Objects", *Proc. the Workshop on Foundations of Deductive Databases and Logic Programming*, pp.6-26, Washington DC, 1986.

[19] N. Miyazaki, K. Yokota, H. Haniuda and H. Itoh, "Horn Clause Transformation by Restrictor in Deductive Databases", *J. Inf. Process.*, vol.12, no.4, 1989.

[20] S. Nishio and Y. Kusumi, "Evaluation Methods for Recursive Queries in Deductive Databases", *Jouhou-Shori*, vol.29, no.3, pp.240-255, 1988 (in Japanese).

[21] C. Takahashi, K. Yokota, M, Yokotsuka, and T. Kajiyama, "A Prototype System of Deductive Databases Bases on Extended Terms and Hierarchical Structure", *Proc. Annual Meeting of IPSJ*, 2C-7, Fukuoka, Oct., 1989 (in Japanese).

[22] M. Tsukamoto, S. Nishio and T. Hasegawa, "DOT — Term Representation for Logical Databases with Object-Oriented Concepts", *Proc. Advanced Database Symposium*, Kyoto, Dec., 1989 (in Japanese).

[23] K. Yokota, "Deductive Approach for Nested Relations", in *Programming of Future Generation Computer II*, eds. bu K. Fuchi and L. Kott, pp.461-481, Elsevier Science Pub., 1988.

[24] K. Yokota, "Formalization of Objects and its Scheme", *Proc. SIGDE of IEICE*, DE88-25, Kyoto, Nov., 1988 (in Japanese).

[25] C. Zaniolo, "Design and Implementation of a Logic Based Language for Data Intensive Applications", *Proc. LP*, pp. 1666-1687, Seattle, 1988.

[26] C. Zaniolo, "Object Identity and Inheritance in Deductive Databases: An Evolutional Approach", *Proc. DOOD89*, Kyoto, Dec., 1989.