Abstract

Object-oriented database system supports the powerful concepts of data abstraction and complex data modeling while deductive database system has inferential and declarative query capabilities. Previous research has demonstrated the advantages and feasibilities of the confluence of the declarative logic-based paradigm and the object-oriented paradigm in the dual-paradigm system of deductive object-oriented database systems. However, the previous research either concentrated on the homogeneous architecture of a complete logical system or the heterogeneous architecture with the schema rules carried over to the deductive database. Our research proposes the use of heterogeneous architecture that minimizes communications between an object-oriented database system and a deductive system. In this paper, we outline the architecture and the system of deductive rules. We demonstrate the efficiency in query processing by separating the schema rules of object orientation from the deductive rules. We also address the object-oriented relationships in this dual-paradigm database system.

Keywords: Deductive database, deductive object-oriented database, dual-paradigm system, heterogeneous architecture, object-oriented database, query processing

1.0 Introduction

The data and behavioral encapsulation of object-orientation have been successfully applied to programming languages and database systems. Object-orientation supports the powerful concept of data abstraction and data modeling but object-oriented database systems lack an inferential capability that is essential to support a broader range of data/knowledge-intensive applications. Object-oriented databases [4, 7] depend heavily on procedural languages that do not provide declarative query capability. This capability must be supported to make such database a viable alternative for managing large data/knowledge volume.

Deductive databases [2, 3] based on first-order logic, by contrast, do incorporate reasoning and deducing capabilities and provide a declarative, logic-based language for expressing facts, rules, and queries. Deductive databases include capabilities to define rules, which can deduce or infer additional information from facts stored in the database. However, deductive databases are not sufficiently powerful enough to model complex data structures.

Previous research has demonstrated the advantages and feasibility of the confluence of the declarative logic-based paradigm with the object-oriented paradigm to support the next generation of database applications in a dual-paradigm system of deductive object-oriented database systems [9, 12, 13]. The resulting system can directly support rules and various reasoning concepts as well as database functions. The system provides a declarative, logic-based language for facts, rules, and queries. The dual-paradigm system can greatly improve productivity and functionality because such a system is able to provide a powerful and versatile system for modeling data: the abstraction and encapsulation concepts in object-oriented databases provide a straightforward mechanism for laying out complex data
relationships and the ability to reuse code, and the inference mechanism in deductive databases provides the simplicity and efficiency of rule processing. Therefore, the combination of deductive and object-oriented technology is a promising direction to pursue in search of the next generation of databases.

In this paper, we focus on the efficient design and implementation of the deductive object-oriented database systems. Our concern in this paper is efficient query processing and optimization to get reasonable performance. We propose the use of heterogeneous architecture but unlike previous research effort [13] in heterogeneous architecture, we use object-oriented model for schema knowledge instead of schema rules. Also, previous research only addressed inheritance. Our architecture allows all data modeling concepts inherent in the object-oriented database system.

The rest of this paper is organized as follows. Section 2 provides a brief background on dual-paradigm database systems. Section 3 surveys related work in this area. Section 4 discusses our architectural approach to build an efficient deductive object-oriented database system. Section 5 describes deductive rules. Section 6 presents query processing and optimization strategies and Section 7 gives the conclusion and discusses future research. Acknowledgements and references are given in Section 8 and Section 9, respectively.

2.0 Dual-Paradigm Database Systems

The integration of programming concepts with database is one of the most significant advances in the evolution of database system. Two models of this approach have become popular: object-oriented databases and deductive databases. The goal of object-oriented databases is to apply object-oriented concepts in object-oriented programming in modeling data. It combines data abstraction and computation models of object-oriented languages with the performances and consistency features of databases. The goal of deductive databases is to integrate rules and a database of facts in logic programming to support complex reasoning.

Object-oriented databases have some features that do not exist in a conventional database (e.g., relational database). They use a set of object-oriented concepts for modeling data, such as object-identity, encapsulation, inheritance, and polymorphism. The generalization and specialization concepts are captured in the class and sub-classes relationships. These concepts are sufficiently powerful to support data modeling requirements of many types of applications.

Deductive databases also have some features that do not exist in a conventional database. A deductive database is a database in which new facts may be derived from the existing facts that are stored by using an inference engine. A deductive database comprises an extensional database (EDB) consisting of a set of facts explicitly stored in a physical database, and an intensional database (IDB) consisting of a set of deductive rules. These rules can be used to derive new facts from the facts in EDB. A salient feature of deductive databases is their ability to support a declarative, rule-based style of expression. The major advantage of this feature is that facts, deduction rules, and queries can all be written in a uniform database language typically based on first-order logic.

A dual-paradigm database system is a system that combines the modeling power of object-orientation and the inference power of deduction. In this paper, we propose the use of an object-oriented database for the EDB and a deductive system for the IDB.

3.0 Related Work

Realizing the benefits of a dual-paradigm database system, research has been done in the past to design such a system. However, most researchers emphasized on the development of a logical language for object-oriented features [5, 6]. Few papers stress the importance of efficiency in the interaction between two existing paradigms. Compilation method was first introduced in [10] for deductive database
that interacts with relational database. Yoon and Ku [13] have proposed a system with efficient query processing and optimization by minimizing the communication traffic between the object-oriented database and the deductive database. In this paper, we try to improve the work we have done previously in [13] by totally separating the data modeling concepts from the deductive rules.

The concept of virtual classes as views has been studied extensively in the context of relational databases and explored in object-oriented databases [11]. In [13], we suggested a mechanism to define a virtual class with rules that is efficient, flexible, and manageable because it is easy to introduce a virtual class, redefine it, and delete it. These virtual class rules together with schema rules constitute all the rules in the deductive object-oriented database system. In this paper, we restrict the rules in the dual-paradigm system to be virtual class rules only. The schemas information is retained in the object-oriented model to reduce query processing time.

4.0 The Heterogeneous Architecture

There are several ways of combining a logic system and an object-oriented system. At one end of the spectrum is the homogeneous approach where a single integrated system combining a logic system with an object-oriented database. At the other end of the spectrum is the heterogeneous architecture we adopted in our research. It consists of two components: an object-oriented database to manage the facts and a logic system to perform deductive reasoning using deductive rules. Between these two ends of the spectrum, we have other architecture such as the one proposed in [13] which is a heterogeneous architecture but there are schemas rules with knowledge of the object-oriented structure stored in IDB.

The heterogeneous approach uses two developed technologies, a deductive logic system and an object-oriented database system. The advantages of this approach are obvious. We will be able to make use of existing systems with few changes without building an integrated homogeneous system from scratch. Existing logic systems and object-oriented databases can be used in implementing a deductive object-oriented database system with great savings in system development and database conversion time.

The heterogeneous architecture in [13] separates the rules and the facts. Thus the object-oriented architecture only concerns about facts. The object-oriented modeling capabilities are captured in the form of schema rules and stored in the rule-base of the deductive database. The IDB is the front-end where a declarative query will be submitted and deduced (resolved). The EDB is a backend system providing facts to satisfy the query. The heterogeneous architecture we propose here does not require schema rules. The IDB can still be the front-end but it needs to know whether a query is an IDB query or an EDB query. Therefore, a more clear-cut approach is to build an interface with knowledge about IDB predicates and EDB predicates. When a query comes into this interface, it will direct the IDB query to the deductive logic system and the EDB query to the object-oriented database system. Section 6 discusses how this works and compares the two approaches.

5.0 Rule Generation

We use the Prolog-like notation [1] for a rule in the deductive system: 
\[ a : \neg b_1, b_2, \ldots, b_m; \]
where \( a \) is an atomic formula and \( b_i \)'s are literals. All variables occurring in the rule are assumed to be universally quantified. The literal \( a \) is called the head of the rule; and \( b_i \)'s are referred to as the body of the rule.

5.1 Virtual Class Rules

In a deductive object-oriented database, we can greatly increase the flexibility and convenience of object-oriented databases by adding a capability to derive a new class, called virtual class which is basically a view definition that a class is derived from other classes.
A major incentive to exploit database systems is the ability to support sharing of data among applications. In certain cases, the system cannot easily serve each user group’s unique needs. The deductive object-oriented database provides a particular user group with the ability to define a virtual class according to its own need. A virtual class is able to provide a user group with a class hierarchy that is more appropriate to its needs than the actual hierarchy in EDB. Virtual classes provide each user group with a more organized view of the data. The mechanism for defining a new class in IDB provides a great deal of restructuring capability.

We can define a virtual class as a single class that is derived from base classes (in EDB) or other previously defined virtual classes (in IDB). Base classes are defined during initial schema definition, and their instances are explicitly stored as base objects in the object-oriented database. Virtual classes are defined during the lifetime of the database using deductive rules defined by a user group’s need. Virtual classes are not actually stored, but can be computed from the base classes by applying an inference mechanism based on the rule specification. Under this restriction, efficient storage and retrieval are still feasible. Even though we update the objects in the base class on which the virtual class is defined, we do not have to change anything in the virtual class. If we modify the objects in the base class on which the virtual class is defined, the virtual class automatically reflects these changes. A virtual class has no proper data of its own. The deductive rules of the virtual class will be resolved until base classes (and property or condition literal – see below) remain, then the population of data occurs.

There are three basic forms to construct a virtual class rule. One involves the logical AND, one involves the logical OR, and one involves the logical NOT.

(1) The first form of virtual class can be represented as follows:

\[ V := C_1, C_2, ..., C_n, P_1, P_2, ..., P_m, E_1, E_2, ..., E_k \]

where \( V \) is a virtual class; each \( C_h, 1 \leq h \leq n \), is the name of a previously defined virtual class, or a base class; each \( P_i, 0 \leq i \leq m \), is a property literal; and each \( E_j, 0 \leq j \leq k \), is a condition literal to find a set of objects. The format of a property literal is \( P(x, y) \) where \( P \) is a predicate name which represents a characteristic of an object, \( x \) represents an object that belongs to a class, and \( y \) represents the characteristic of the object \( x \). Notice that both \( P_i \) and \( E_j \) can be zero so this virtual class rule can be in the following form:

\[ V := C_1, C_2, ..., C_n \]

**Example 1:** The virtual class **Senior** can be defined from the base class **Person** with the characteristic that a person whose age is greater than 65 is a senior.

**Senior(x)** := **Person(x)**, Age(x, y), GT(y, 65)

In this example, **Age(x, y)** is a property literal to show that **Person x** is **y** years old. **GT** is obviously a condition literal.

**Example 2:** The following two rules define the virtual class **No-Income-Tax**, consisting of all adults who are students and subject to special income tax laws.

**Adult(x)** := **Person(x)**, Age(x, y), GT(y, 21), LE(y, 65)

**No-Income-Tax(x)** := **Student(x)**, **Adult(x)**

The second rule consists of base class **Student** and virtual class **Adult** without property or condition literal.

(2) The second form of virtual class can be represented as follows:

\[ V := C_1 \lor C_2 \lor ... \lor C_n \]

The virtual class is defined to have a set of object instances composed of the members of either or all of the classes involved in the definition of the virtual class.

**Example 3:** The following rule defines the virtual class **Water-Vehicle** from the two base classes, **Ship** and **Boat**.

**Water-Vehicle(x)** := **Ship(x)** \lor **Boat(x)**

The purpose of this rule is not for capturing the existing relationship in the object oriented database. The rule generated new fact (**Water-Vehicle**) from existing facts (**Ship** and **Boat**) in EDB.

(3) The third form of virtual class is defined to have a set of object instances that are members of the first group of classes but not
the second group of classes. The virtual
class defined in this way can be represented
as follows.
\[ V := C_1, C_2, \ldots, C_m, \neg C_{m+1}, \neg C_{m+2}, \ldots, \neg C_n \]
\( C_1, C_2, \ldots, C_m \) are the first group of classes
and \( \neg C_{m+1}, \neg C_{m+2}, \ldots, \neg C_n \) are the second
group of classes.

**Example 4:** The following rule defines the
virtual class *Orphan*, consisting of persons without parents.

\[ \text{Orphan}(x) := \text{Person}(x), \neg \text{Parents}(y, x) \]
x is an object from the *Person* class but x is
not an instance of the *Parents* class.

The virtual class rules constitute the
deductive database of the deductive object-
oriented database. They are the IDB of the
system.

### 5.2 Rule Transformation

The deductive object-oriented database
system stores rules in the logic system and facts
in the object-oriented database system. Such a
separation of rules and facts into disjoint sets is
always possible [8]. Two more transformations
of rules are applicable to the deductive rules. The first is the “unique intensional literals”
which means that a literal should either be
extensionally or intensionally defined but not
both. One can always get rid of this equality of
names by renaming the extensional literal \( P^* \)
and introducing the rule \( P := P^* \). Unique literals
are needed to get a unique resolution tree when
we process a query. In our approach, we need to
distinguish EDB and IDB literals and EDB and
IDB queries in query processing.

The second transformation is the “extended
term-restricted rules” which has the following
two characteristics:
(1) All the variables in the head of the rule also
appear in the body.
(2) The rule does not have any constant in the
head.

Converting our rules to extended term-restricted
rules ensures that all the information in the head
of a rule also appears in the body, which
prevents us from losing this information while
doing resolution, because the subsequent
resolvent will also contain it.

### 6.0 Query Processing and
Optimization

One of the major reasons to have
heterogeneous architecture is to have database-
independent query simplification and query
optimization. Our strategy is to minimize the
number of times the system has to cross the
boundaries, because crossing the boundaries of
two systems is expensive especially over long
distance of networking or slow connectivity.
Queries for the deductive object-oriented
database system are of two types:

(1) EBD query – Only base classes appear in
this type of queries. That is, queries refer
only base classes in EDB whose instances
are answer objects to the queries. As
mentioned in Section 4, these queries can be
evaluated directly by the object-oriented
database system.

(2) IDB query – Only virtual classes or both
virtual classes and base classes are appeared
in this type of queries. In this case,
resolution is performed between a virtual
class in a query and the rules defining the
virtual class until the resolvent consists of
base classes only (together with property
and/or condition literal). Query evaluation
starts with consideration of a query as a goal
to be achieved. Unifying the query with the
rules in IDB successively transforms the
goal into subgoals until each can be
evaluated against the EDB.

Let us consider the following example and
compare the query evaluation process with the
previous research results in [13]:

**Example 5:** Suppose we have the following
two rules in IDB as specified in Example 2:

**Rule 1:** \( \text{Adult}(x) := \text{Person}(x), \text{Age}(x, y), \text{GT}(y, 21), \text{LE}(y, 65) \)

**Rule 2:** \( \text{No-Income-Tax}(x) := \text{Student}(x), \text{Adult}(x) \)

In [13], schema knowledge is stored in IDB.
Suppose the class *Person* is the superclass of
*Student*. Rule 0 is the schema rule of class
hierarchy.

**Rule 0:** \( \text{Person}(x) := \text{Student}(x) \)
A query to “find all people who do not pay any income tax” can be represented as:

**Query:** :- No-Income-Tax(x)

The resolution sequence is shown below:

**R1:** :- Student(x), Adult(x) [from Query and Rule 2]

**R2:** :- Student(x), Person(x), Age(x, y), GT(y, 21), LE(y, 65) [from R1 and Rule 1]

**R3:** :- Student(x), Student(x), Age(x, y), GT(y, 21), LE(y, 65) [from R2 and Rule 0]

After eliminating the duplicated predicate Student(x) in R3, we get:

**R4:** :- Student(x), Age(x, y), GT(y, 21), LE(y, 65)

The answer set includes all the students whose age is greater than 21 and less than or equal to 65.

Our approach does not require schema rules. **Person(x)** and **Student(x)** are remained in EDB since the inheritance relationship has already been captured by the **Person** and **Student** classes in EDB. With our approach, only two resolution steps are needed:

**R1’:** :- Student(x), Adult(x) [from Query and Rule 2]

**R2’:** :- Student(x), Person(x), Age(x, y), GT(y, 21), LE(y, 65) [from R1’ and Rule 1]

For rules with negation in the body or rules with OR (\(\lor\)) in the body, algebraic operations of difference and union can be used respectively in EDB as specified in [13]. Our approach expedites the inference process in IDB.

Obviously, less resolution steps are needed if schema rules are not in IDB. Instead of keeping all the object-oriented relationships (e.g., inheritance, aggregation, composition) in the resolution process, we let EDB handle all the queries regarding such relationships since these relationships are already in the object-oriented model. All we need is to have the user interface to recognize such EDB queries and map them to the object-oriented model/language. In doing so, all the object-oriented relationships are addressed that the previous research [13] did not and there is no need to construct rules for these relationships.

### 7.0 Conclusion and Future Research

In the previous research of heterogeneous architecture of deductive object-oriented architecture, the main purpose of the object-oriented database is to provide data (facts) as a back-end component for the deductive front-end component. In this paper, we offer an alternative implementation which makes use of the object-oriented features of the object-oriented database. We do not need schema information to be transferred to the deductive system. This results in a simpler deductive structure without the need to construct complicated rules for all the object-oriented relationships. The only requirement is the capability to distinguish EDB queries and IDB queries.

Previous research and this current proposal of alternative implementation specify the interaction between IDB and EDB at the logical level. Actual integration of object-oriented database and deductive system is needed to study the ramification of system implementation at the physical level.

Future research also includes the processing of EDB queries and resolvent sent from the IDB inference process in the object-oriented system. A mapping between logic constructs and object-oriented query language is needed to generate answers from the object-oriented database.

### 8.0 Acknowledgements

This research was supported in part by a grant from the Center for Research, College of Science and Health, and in part by the ART (Assigned Released Time for Research) program, Office of the Provost, William Paterson University of New Jersey. This research was also supported in part by the Provost Grant, Widener University.

### 9.0 References


