# Generic Methods in Deductive Object Databases

Elisa Bertino
Dipartimento di Scienze dell'Informazione
Università di Milano
Via Comelico, 39 20133 Milano, Italy
`bertino@hermes.mc.dsi.unimi.it`

Giovanna Guerrini
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova
Viale Benedetto XV, 3 16132 Genova, Italy
`guerrini@disi.unige.it`

Danilo Montesi *
Informatics Department
Rutherford Appleton Laboratory
Chilton, Didcot OX11 0QX, UK
`danilo@inf.rl.ac.uk`

**Abstract**

Deductive objects have been introduced in [3] to support declarative object specification in the database context taking advantage of the large body of results on Datalog-like language. However, the rigidity of logical languages does not reflect the flexible programming style of object-oriented systems. For instance the application of the same method to different objects. In this paper we propose an extension based on variable labels that allow to express generic methods through rules. The semantics of this approach is still based on fixpoint computation.

**Keywords:** Object-oriented paradigm, state evolution, knowledge bases.

## 1 Introduction

The object-oriented paradigm has being widely applied in several areas of computer science, such as programming languages, information systems, software engineering and user interfaces. In the specific case of a distributed information system the distribution of data/rules requires to consider the distribution of the database and thus the development of cooperative databases. Moreover, for some complex application domains (like Computer Integrated Manufacturing applications), the information system is inherently distributed. Many applications integrating and using data and services of local database systems [4] are designed and needed. In all cases, the specification of a cooperating information system presents strong analogies with the specification of composite (database) systems. Information systems can be seen as a collection of "objects" each of them incorporating some knowledge (like facts, rules, and constraints) and being a unit of design that can be composed with other objects. In this paper we consider the above problem in the context of deductive databases. The motivation for this choice is related to the formal model underlying deductive databases, which provides a formal behavior and also a computational model. Note that we will consider an extension called Obj-U-Datalog [3] considering deductive objects, that is objects expressed through a logic language and which can change the state. The relevant characteristic of U-Datalog is that updates are not executed as soon as they are evaluated, rather they are collected in a set and executed altogether at the end of the refutation process, if this process succeeds and the

*The work of D. Montesi has been partly supported by the ERCIM fellowship *Information and Knowledge Systems*.

set is ground and consistent (i.e. it does not contain complementary updates on the same fact). The feature of Obj-U-Datalog is instead to group data and rules to form deductive objects. Such objects interact through labeled atoms. The language we propose in this paper (called X-Obj-U-Datalog) is based on the notion of deductive object. Each deductive object is an U-Datalog database. Each object has a state (a set of facts) and a set of methods (rules) to manipulate the data. Methods may also contain update atoms to modify the object state. Moreover the computational model of our language is based on cooperation among objects through message passing. In Obj-U-Datalog such cooperation was fixed once for all at program development time. In this paper, instead we propose an approach to dynamic message passing where the label is a variable and can be instantiated at execution time. Our approach greatly improves the flexibility of deductive objects and is in the main stream of object calculus, according to [8]. In our approach, methods expressed through rules, cooperate with objects which are not fixed. They can change over time according to different instances for variable labels. We call them *generic methods*. Unfortunately, this dynamicity is paid with more computation. Indeed, in X-Obj-U-Datalog, the evaluation of a transaction must consider this new dynamic component which was not present in Obj-U-Datalog databases. The protptype has bee implementated translating the X-Obj-U-Datalog database into and U- Datalog database and by means of a bottom-up meta interpreter for U-Datalog [2]. The result of this paper is a rule language of cooperating objects expressed as deductive databases preserving the nice computational model of Datalog language. This allows the re-use of the already developed techniques for efficient query evaluation. Moreover, in databases an important issue is to ensure transactional behavior of a set of updates, that is all of them are executed or none of them is performed. Thus any collection of cooperating databases should ensure a transactional behavior. This is the second result of this paper, that is the semantics of cooperating databases has a transactional behavior. In the remainder of the paper we introduce the language and we show the flexibility through an example. We assume some previous knowledge of Datalog language [5].

## 2 X-Obj-U-Datalog

A deductive database $EDB_i \cup IDB$ expresses an object where the EDB part is the *object state* and the IDB part expresses *the methods* to manipulate the state. The deductive capability comes from the logical nature of the Datalog language. Rules are used to express simple methods which can query and/or update the object. Other languages use rules as methods. The approach proposed by Abiteboul et al. [1] does not consider state evolution. The approach proposed in [6] considers state evolution and has a formal semantics, but uses active rules (e.g., production rules extended with events) to express methods. An X-Obj-U-Datalog program consists of a set of object databases, each object in the program consists of the object state and the methods, that is $obj_j = \langle EDB_j, IDB_j \rangle$. $obj_j$ is the object identifier which is defined over a fixed domain of constant object names $OID$. The *object state* is a set of facts, that is a set of ground atoms. The object state is a time-varying component, so in the following we may denote with $EDB_j^i$ the possible states of object $obj_j$, i.e. $EDB_j^i$ denotes the $i$-th state of object $obj_j$.

**Definition 2.1** *A set of methods is a set of rules of the form*

$$H \leftarrow U_1, \ldots, U_i, B_{i+1}, \ldots, B_w, X_1 : B_{w+1}, \ldots, X_p : B_z.$$

*where $H$ is an intensional atom, $X_1 : B_{w+1}, \ldots, X_p : B_z$ are labeled conditions, that is they refer to specific objects. $B_{i+1}, \ldots, B_w$ (as in Datalog) are unlabeled conditions, that is they refer to the object itself where the rule is defined. $U_1, \ldots, U_i$ is the update part. To ensure encapsulation the updates refer to the object itself. The updates $(U_1, \ldots, U_i)$ and conditions $(B_{i+1}, \ldots, B_w, X_1 : B_{w+1}, \ldots, X_p : B_z)$ cannot be both empty. The variables $X_1, \ldots, X_p$ ranges over $OID$ and must appear as arguments of an extensional predicate.*

The intuitive meaning of a rule is: "if $B_{w+1}$ is true in the object to which $X_1$ is instantiated , ..., $B_z$ is true in the object to which $X_p$ is instantiated, $B_{i+1}, \ldots, B_w$ are true in the object where the rule is defined and the updates $U_1, \ldots, U_i$ are consistent, then $H$ is true". The notion of consistency is given informally. Intuitively, the updates $+p(X), -p(X)$, i.e. complementary updates, are not consistent. The updates $+p(Y), -p(X)$ could be consistent if the bindings for the variables were for example $X = tom, Y = bob$. By contrast with the bindings $X = tom, Y = tom$, they are not consistent. Cooperation among objects is supported using labeled atoms in rule bodies. If the object $obj_i$ has a rule containing the labeled atom $obj_j : B_s$, this means that object $obj_i$ cooperates with another object (the one to which variable $X_J$ is instantiated), calling the method $B_s$. Note that a method call can involve updates only as side effect. This ensures the encapsulation. Thus a method call is a channel, where we have synchronous communication and parameter passing through unification. The use of labeled atoms in rules supports message passing among objects, thus we refer to labeled atoms also as message atoms.

**Definition 2.2** *An Obj-U-Datalog program consists of a fixed set of cooperating objects*

$$O - DB = \{obj_1, obj_2, \ldots, obj_s\}$$

*where each* $obj_j$, $1 \leq j \leq s$, *consists of an extensional component* $EDB_j$, *which is a set of ground facts, called object state, and an intensional component* $IDB_j$, *which is a set of methods, as in Definition 2.1.*

A transaction has the form $B_1, \ldots, B_w, obj_1 : B_{w+1}, \ldots, obj_p : B_z$. and cannot contain update atoms. However its execution may indirectly generate updates, because of the invocation of rules with update atoms in their bodies. We do not allow update atoms in transactions to provide encapsulation, i.e., an object state can only be modified through its methods. Note that a transaction may contain two different kinds of atoms: labeled ones and unlabeled ones. Unlabeled atoms stand for the request for a refutation of the atom in any object constituting the database, while labeled atoms are directed to a specific object. The values to these objects can be given at transaction time. Note that the language does not support a strict encapsulation, in that it allows to directly access the attribute values (through queries on extensional predicates). We only disallow the modification of object attributes from outside the object. A *complex transaction* $T$ is a sequence of transactions $T_1; \ldots; T_k$. It should be clear that a transaction provides different roles: the role of a query, in that it returns a set of bindings, an update role (even if indirectly, as seen) with a transactional behavior (all the updates are executed or, in case of inconsistencies, none of them is performed).

**Example 2.1** *We assume a collection of cooperating rule based databases. Each of them can change its state through updates. Consider a cooperating databases containing four objects* $obj_1, obj_2, obj_3$ *and* $obj_4$, *where*

$$obj_1 = \quad \texttt{name(computer\_science), n\_exams(18), good\_score(S)} \leftarrow \texttt{leq(S, 25)}$$

$$obj_2 = \quad \texttt{name(mathematics), n\_exams(15), good\_score(S)} \leftarrow \texttt{leq(S, 27)}$$

$$\begin{aligned} obj_3 = \quad & \texttt{dept(obj}_1\texttt{), year(3),} \\ & \texttt{good(NEx, Avg)} \leftarrow \texttt{dept(X), year(Year), X : n\_exams(NE),} \\ & \qquad\qquad \texttt{X : good\_score(Avg), obj}_4 \texttt{ : ad(NEx, Year, NE),} \\ & \texttt{chDept(NDept)} \leftarrow -\texttt{dept(Dept)}, +\texttt{dept(NDept), dept(Dept).} \end{aligned}$$

$obj_4 =$ `contains a table of facts for the predicate` *ad*.
`This object does not have methods.`

*Since* $\mathtt{dept(obj_1)}$ *is contained in* $obj_3$*, the rule for the good predicate contains two message calls to object* $obj_1$*. Suppose now to execute the transaction* $\mathtt{obj_3 : chDept(obj_2)}$*, then the rule for the good predicate contains now two message calls to object* $obj_2$*. In such a way, if the value for the attribute department of object* $obj_3$ *is changed, the good predicate refers automatically to the data of the new department.* $\diamond$

From the above example we can note that there are different kinds of (synchronous) cooperation:

- $obj_3 \Rightarrow_{ad} obj_4$. This is a one way, one-to-one fixed cooperation on the channel $ad$.
- $obj_3 \Rightarrow_{good\_score, n\_exams} obj_1$. This is a one way, one-to-one dynamic cooperation on the channels $good\_score, n\_exams$. Indeed, after the execution of the transaction $obj_3 : chDept(obj_2)$ this cooperation is substituted by $obj_3 \Rightarrow_{good\_score, n\_exams} obj_2$.

A labeled condition $(X_j : k(x))$ represents a channel $(k)$ between the rule of the database where it is defined $(obj_i)$ and the label $(X_j)$ of the condition. The cooperation is provided through the condition and the parameters are transmitted through unification [7].

The semantics of an X-Obj-U-Datalog program is an extension of that introduced in [3]. Unfortunately, the dymanicity induced by variable labels do not allow to apply straightforward efficient query evaluation methods. However, the approach to transform an X-Obj-U-Datalog database into a U-Datalog one allow use to transform variable labels into special varible and the to apply efficient query evaluation strategies to the transformed database. On open interesting point is the application of extended efficient query evaluation strategies directly to X-Obj-U-Datalog.

## References


[1] S. Abiteboul, G. Lausen, H. Uphoff, and E. Waller. Methods and rules. In P. Buneman and S. Jajodia, editors, *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 32–41, 1993.

[2] E. Bertino, B. Catania, G. Guerrini, M. Martelli and D. Montesi. A Bottom-up Interpreter for database languages with Updates and Transactions. To appear *Proc. Joint Conference on Declarative Programming Gulp-Prode*, Peniscola, 1994.

[3] E. Bertino, G. Guerrini, and D. Montesi. Deductive Object Databases. *Proc. Eighth European Conference on Objects-Oriented Programming*, Bologna, pages 213–235, Springer-Verlag, Bologna, 1994.

[4] M. L. Brodie. The Promise of Distributed Computing and the Challenges of Legacy Systems. In P. M. Gray and R. J. Lucas, editors, *Proc. BNCOD 10*, Lecture Notes in Computer Science, vol. 618, pages 1–28. Springer-Verlag, Berlin, 1992.

[5] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag, Berlin, 1990.

[6] D. Montesi and R. Torlone. A Rewriting technique for implementing Active Object Systems. To appear *Proc. International Symposium on Object-Oriented Methodologies and Systems (ISOOMS)*, Palermo, 1994.

[7] F.G. McCabe. *Logic and Objects.* PhD thesis, University of London, November 1988.

[8] O. Nierstrasz. Towards an object calculus. In *ECOOP '91 workshop on object-based concurrent computing*, Lecture Notes in Computer Science, vol. 612, pages 1–20. Springer- Verlag, Berlin, 1991.