Deductive Object Databases

Elisa Bertino
Dipartimento di Scienze dell'Informazione
Università di Milano (Italy)
bertino@hermes.mc.dsi.unimi.it

Giovanna Guerrini
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova (Italy)
guerrini@disi.unige.it

Danilo Montesi *
Departamento de Informatica
Universidade de Lisboa (Portugal)
danilo@di.fc.ul.pt

Abstract

This paper proposes a new approach to model deductive object databases. Each object database is described by means of a Datalog language extended with extensional updates, called U-Datalog. Each object can change its state and cooperate with other objects. We introduce an extension of U-Datalog to approach the problem of composition among object databases. It can be used for modular database design and for cooperation among databases. The resulting language has a clear semantics for the evolution of objects and for modeling the transactional behavior of the resulting database. Finally, we describe some architectural issues of the prototype which has been developed.

Keywords: Logic languages, object based paradigm, composite databases, transactional behavior.

1 Introduction

In the last few years the object-oriented paradigm has being widely used in several areas of computer science, such as programming languages, databases, software engineering, user interfaces. In particular, the notion of modularity, which is already widely accepted in the programming language field, is an important notion also in the database field. The need for considering federated databases rather than a centralized one is motivated by several issues. The role played by databases in Artificial Intelligence, when an expert system requires

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

processing large volumes of data and rules, is one of them. In the specific case of a distributed expert 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, e.g. the Computer Integrated Manufacturing applications), the database system is inherently distributed. Many applications integrating and using data and services of local database systems [13] are designed and needed. In all cases, the specification of a federated system presents strong analogies with the specification of composite (database) systems. In [22], such concerns motivate the identification 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 to Datalog considering updates and transactions. The language in which our modules are expressed is in fact Update Datalog (U-Datalog) [10], an extension of Datalog which allows the specification of updates in rule bodies. 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 set is ground and consistent (i.e. it does not contain complementary updates on the same fact).

The language we propose in this paper (called Obj-U-Datalog) is based on the notion of object. Each object is an U-Datalog database. Each object has a state (a set of facts) and a set of methods. Methods may also contain update atoms to modify the object state. In such a way we model objects with OIDs, state, behavior and state evolution. Moreover the computational model of our language is based on cooperation among objects through message passing. In a conventional deductive database, in fact, the evaluation of a query may make use of all facts and rules in the database. In an Obj-U-Datalog database, by contrast, facts and rules are grouped in objects, and the evaluation of a query is performed in a specific object. During the evaluation process an object may request the evaluation of a subquery to another object (through a message call), in such a case the computation moves to another object (we will also say that a *context switch* has happened).

Our approach to deductive object databases relates to previous work on logical object databases as follows. Some of these approaches do not consider state evolution of deductive objects [12, 19, 27, 28, 24, 2, 15, 21, 23]. Others consider state evolution [5, 16] but objects have the granularity of terms. By contrast we consider an object with larger granularity, that is a theory, i.e. a set of logical clauses. Moreover many of the considered approaches do not consider the behavioral component of objects (i.e. the methods). We think that this is an important issue because it overcomes the dichotomy between data and operations of the relational model. The approach to deductive object databases of [11] models objects with a theory granularity, classes and state evolution has no formal semantics. This is related to control constructs in rules bodies extending Datalog language with updates. The result of this paper is a clear, formal semantics 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 with a transactional behavior. A prototype of the language proposed in this paper has been implemented, via a translation from Obj-U-Datalog to U-Datalog which "flats" the structure of the database, and an interpretation of the obtained U-Datalog program with a bottom-up interpreter. In the paper we briefly describe the developed prototype. The remainder of the paper is organized as follows. Section 2 informally introduces the U-Datalog language and its non-immediate update semantics. Section 3 introduces the Obj-U-Datalog language, presenting its syntax. Section 4 defines the semantics for the language, while Section 5 describes the prototype of the language that has been implemented. Finally, Section 6 presents some conclusions and outlines future works.

2 Overview of U-Datalog

A Datalog program consists of a set of *base relations* (EDB) and a set *rules* (IDB). Many extensions to Datalog have been proposed to express updates (see [1] for a survey). In the following we summarize a new approach based on a non-immediate update semantics that we will use in order to model deductive object databases.

U-Datalog is a rule language supporting declarative specification of updates in program rules. The execution model of U-Datalog consists of two phases, the marking phase and the update phase [33]. The first phase collects the updates found during the evaluation process, without, however, executing them. During the second phase the updates are executed altogether only if they are ground and consistent. If the set is not consistent, or if it contains non-ground updates, the transaction is *aborted* and no update in the set is performed. The notion of consistency is an important one, in that it prevents a set of updates containing both an insertion and a deletion of the same fact to be executed. By contrast in DLP [25], $\mathcal{LDL}$ [30] and DL [3], updates are executed as soon as they are evaluated. This approach leads to complex semantics and to computations performed in a sequence of states instead of in a single one. In the following we recall the language and its informal behavior (see [29] for a complete description).

We consider pairwise disjoint sets of *constants* Σ, *predicate names* Π and *variables* V. Predicates have a certain arity. A *term* is either a constant or a variable. The *atoms* are expressions of the form $p(t_1, ..., t_n)$, where the $t_i, 1 \leq i \leq n$, are terms and p is a predicate name. Predicate symbols are partitioned in two disjoint sets: the extensional predicates (Π_{EDB}), and the IDB-*predicate symbols* (Π_{IDB}), occurring in the intensional database but not in the extensional one. The extensional atoms are those built starting from the extensional predicates. *Updates* are extensional atoms prefixed with + or − to denote respectively insertions or deletions. A *rule* is an expression of the form $H \leftarrow U_1, \ldots, U_i, B_{i+1}, \ldots, B_n, n \geq 0$, where the *body* $U_1, \ldots, U_i, B_{i+1}, \ldots, B_n$ is a sequence of updates and atoms and the head H is an intensional atom. If $n = 0$, a rule is also called a *fact.* in the following, $\tilde{t}$ denotes the tuple $t_1, ..., t_n$, while $\bar{A}$ denotes a (possibly empty) conjunction of atoms $A_1, \ldots, A_k$. A *substitution* is a function $\theta : V \rightarrow Term$ which associate with each variable a term in $Term$. It extends to apply to any syntactic object (e.g. term, atom, rules, etc.) in the usual way. We require rules to be *safe*. Our notion of safety extends that in [31]. A rule is safe if each variable in the head appears in a non update atom in the body.

An *extensional* database (EDB) is a set of facts. The extensional database is a time-varying component, so we may denote it with EDB_i to emphasize that we consider the extensional database at time i. An *intensional* database (IDB) is a finite set of rules. Atoms and updates which do not contain variables are called *grounds*. A *transaction* is a rule with no head.

Example 2.1 *Consider* $EDB_i = \mathtt{q(b)}$ *and*

$$\begin{aligned} IDB \quad = \quad & \mathtt{p(X)} \leftarrow -\mathtt{q(X)}, \mathtt{q(X)}. \\ & \mathtt{r(X)} \leftarrow +\mathtt{t(X)}, \mathtt{p(X)}. \\ & \mathtt{s(X)} \leftarrow \mathtt{t(X)}. \end{aligned}$$

The transaction $T_1 = \mathtt{r(X)}$ *evaluated in* $EDB_i \cup IDB$ *computes the binding* $\mathtt{X} = \mathtt{b}$ *and collects the updates* $-\mathtt{q(b)}, +\mathtt{t(b)}$*. Informally the new extensional database* $EDB_{i+1} = \mathtt{t(b)}$ *is the result of the application of these updates to* EDB_i*. The transaction* $T_2 = \mathtt{s(X)}$ *evaluated in* $EDB_{i+1} \cup IDB$ *computes the binding* $\mathtt{X} = \mathtt{b}$ *and does not compute any update, thus the new extensional database is still* EDB_{i+1}*. The transaction* $T_3 = +\mathtt{q(X)}, \mathtt{s(X)}$ *evaluated in* $EDB_{i+1} \cup IDB$ *computes the binding* $X = b$ *and collect the update* $+\mathtt{q(b)}$*, thus the new extensional database is* $EDB_{i+2} = \mathtt{t(b)}, \mathtt{q(b)}$*. The transaction* $T_4 = +\mathtt{q(X)}, \mathtt{p(X)}$ *computes the binding* $\mathtt{X} = \mathtt{b}$*, and collects the updates* $+\mathtt{q(b)}, -\mathtt{q(b)}$*. They are not consistent and therefore* T_4 *aborts.* ◇

Transaction can be composed forming a sequence, $T_1; \ldots; T_k$ that behaves as a transaction itself. Note that control is allowed only at transaction level. For instance, the sequence construct ";" is not allowed in rule bodies. This is the major difference with the already existing approaches to integrate updates in declarative query languages. Moreover, due to the two phases computation there is no rollback, that is, updates undo is not needed.

3 Obj-U-Datalog

The deductive database $EDB_i \cup IDB$ of Example 2.1 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 uses rules as methods. The approach proposed by Abiteboul et al. [4] does not consider state evolution. The approach proposed by Ceri et al. [17], considers state evolution, however it does not provide a formal semantics due to active rules (e.g., production rules extended with events) used to express methods. As in th object oriented paradigm, methods ensure encapsulation and allow cooperation among objects [32].

An 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 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 3.1 *A set of methods is a set of rules of the form*

$$H \leftarrow U_1, \ldots, U_i, B_{i+1}, \ldots, B_w, obj_1 : B_{w+1}, \ldots, obj_p : B_z.$$

where H is an intensional atom, $obj_1 : B_{w+1}, \ldots, obj_p : B_z$ 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 ($B_{i+1}, \ldots, B_w, obj_1 : B_{w+1}, \ldots, obj_p : B_z$) cannot be both empty.

The intuitive meaning of a rule is: "if B_{w+1} is true in obj_1, ..., B_z is true in obj_p, $B_{i+1}, \ldots, B_w$ 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 obj_j 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 allows message passing among objects, so we refer to labeled atoms also as message atoms.

Definition 3.2 *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 3.1.*

Definition 3.3 *A transaction has the form*

$B_1, \ldots, B_w, obj_1 : B_{w+1}, \ldots, obj_p : B_z.$

where $B_1, \ldots, B_w, obj_1 : B_{w+1}, \ldots, obj_p : B_z$ *are as in Definition 3.1.*

Transactions cannot contain update atoms. However its execution may generate updates indirectly, because of the invocation of rules with update atoms in their bodies. We do not allow update atoms in transaction 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. 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 3.1 *We assume a collection of cooperating rule based databases. Each of them can change its state through updates. Consider the following cooperating databases:*

$obj_1 = \langle EDB_1, IDB_1\rangle, obj_2 = \langle EDB_2, IDB_2\rangle, obj_3 = \langle EDB_3, IDB_3\rangle$

with

$$\begin{array}{rcl} EDB_1 & = & \mathtt{p(a).} \\ & & \mathtt{q(b).} \\ IDB_1 & = & \mathtt{k(X)} \leftarrow \mathtt{p(X)}, \mathtt{obj_2} : \mathtt{k(X)}. \end{array}$$

$$
\begin{aligned}
& \mathtt{r(X)} \leftarrow +\mathtt{p(X)}, \mathtt{q(X)}. \\
& \mathtt{m(X)} \leftarrow +\mathtt{q(X)}, \mathtt{k(X)}. \\
& \mathtt{m(X)} \leftarrow \mathtt{obj_3} : \mathtt{s(X)}, \mathtt{obj_2} : \mathtt{t(X)}.
\end{aligned}
$$

$$
\begin{aligned}
EDB_2 &= \mathtt{g(a)}. \\
IDB_2 &= \mathtt{k(X)} \leftarrow \mathtt{g(X)}, \mathtt{obj_3} : \mathtt{s(X)}. \\
& \quad\; \mathtt{t(X)} \leftarrow +\mathtt{g(X)}, \mathtt{obj_1} : \mathtt{r(X)}. \\
& \quad\; \mathtt{m(X)} \leftarrow \mathtt{k(X)}, \mathtt{obj_1} : \mathtt{k(X)}.
\end{aligned}
$$

$$
\begin{aligned}
EDB_3 &= \mathtt{s(a)}. \\
& \quad\; \mathtt{s(b)}. \\
IDB_3 &= \mathtt{w(X)} \leftarrow \mathtt{obj_1} : \mathtt{m(X)}, \mathtt{obj_2} : \mathtt{m(X)}. \\
& \quad\; \mathtt{n(X)} \leftarrow +\mathtt{s(X)}.
\end{aligned}
$$

- *The transaction* `m(X)` *given to the resulting database computes* `X/b` *(in obj_1) and* `X/a` *(both in obj_2 and obj_1). Note that these answers are the result of a cooperation among the objects. As side effect the insertion of* `p(b)` *and of* `q(a)` *is performed in obj_1, while* `g(b)` *is inserted in obj_2.*
- *The transaction* `p(X)` *computes the binding* `X/a`*, as transaction* $\mathtt{obj_1} : \mathtt{p(X)}$*. By contrast, both* $\mathtt{obj_2} : \mathtt{p(X)}$ *and* $\mathtt{obj_3} : \mathtt{p(X)}$ *fail.*
- *The transaction* `t(X)` *computes* `X/b` *and changes the state of obj_1 and obj_2 adding* `p(b)` *and* `g(b)` *respectively. Note that those updates are performed in parallel. They do not form a sequence.*
- *The transaction* `k(a)` *is simply true, due to the refutation obtainable both in obj_1 and in obj_2.*
- *The transaction* $\mathtt{obj_3} : \mathtt{n(X)}$ *aborts in that it generates a non ground update* $+\mathtt{s(X)}$*.* ◇

A labeled condition ($obj_j : k(X)$) represents a channel (k) between the rule of the database where it is defined (obj_i) and the label (obj_j) of the condition. The cooperation is provided through the condition and the parameters are passed through unification [27]. From the above example we can note that there are three types of (synchronous) cooperation:

- $obj_1 \Leftrightarrow_k obj_2$. This is a two ways, one-to-one cooperation on the channel k.
- $obj_2 \Rightarrow_r obj_1$. This is a one way, one-to-one cooperation on the channel r.
- $obj_3 \Rightarrow_m obj_1, obj_2$. This is a one way, one-to-many cooperation on the channel m.

Transactions may consider the cooperating databases just as one database, that is they may span several objects, or they may be explicitly addressed to a specific object. Although, the control is introduced at transaction level, this this is not inconsistent with the local view of the world encouraged by the object-oriented paradigm because the state manipulation is allowed just inside methods expressed through rules bodies. Indeed, with U-Datalog we have moved the control from rule language to transactional language taking full advantage of this transactional language in terms of a simple and clear semantics, transactional behavior and the straightforward application of already existing evaluation techniques to our extended schema.

In the following we provide another example to show the effectiveness of our approach to model real application domains. We do not provide the complete Obj-U-Datalog code for the example due to space limitations.

Example 3.2 *Consider an University office. Suppose that this office is organized in four separated units:*

1. *a didactic division*
2. *a teaching division*
3. *an administrative division*
4. *a plans of studies and theses division.*

In Obj-U-Datalog we model each division with a module, i.e. an object. The first division handles data about students, i.e. year attended and exams passed. Suppose that these data are stored respectively in two extensional predicates:

- $year(Stud, Year)$ *which relates the student with student code Stud to the Year he attends*
- $exam(Stud, Course, Date, Mark)$ *which records the fact that the student with student code Stud has passed the exams for Course in Date with Mark.*

Suppose that the following operations must be provided

- *modifications of the extensional data*
- *computation of the average mark of a student*
- *computation of the exams to be done for a student.*

These operations are expressed as methods through rules for the intensional predicates mod_year, ins_ex, average, miss_ex. Note that to determine the exams to be done for a student the object needs to cooperate with the division 4 (plans of studies) to get the exams in the student plan of studies.

The second division handles data about teachers, i.e. their role and the taught courses. Suppose that those data are stored respectively in two extensional predicates:

- $role(Teach, Role)$ *which relates the teacher with code Teach to its Role, i.e whether he/she is a researcher, an assistant professor or a professor.*
- $course(Teach, Course, Role)$ *which records the fact that the teacher with code Teah teaches Course with a given Role (i.e. as a professor or as a lecturer).*

For this division only the modification operations are needed. These operations are expressed as methods by rules for the IDB predicates mod_role, mod_course.

The third division handles administrative data. For each person of interest for the University it stores name, address, date of birth, and so on. Suppose that these data are stored in an extensional predicate:

- $adm\_data(Code, Name, Address, Date)$ *which relates the person (either a student or a teacher) with code Code to its Name, Address and Date of birth.*

Suppose that the following operations must be provided

- *modifications of the extensional data*
- *computation of the fee to be paid by each student*
- *computation of the salary for each teacher.*

These operations are expressed as methods through rules for the intensional predicates mod_adm, fee, salary. Suppose that the fee to be paid depends on the year of course, the average and the number of missing exams of the student. Therefore we need a cooperation with the first division (by accessing the attribute year and invoking the methods average and miss_ex). Suppose moreover that the salary of a teacher depends on the role of the teacher and on the number of courses he/she teaches. Therefore we need a cooperation with the second division (by accessing the attributes role and course).

The fourth division handles data about plans of studies and theses. Suppose that these data are stored respectively in two extensional predicates:

- $plan(Stud, Course, Year)$ *which stores that the student with student code Stud has planned to attend Course in Year*
- $thesis(Stud, Course, Advisor)$ *which stores that the student with student code Stud is doing a thesis on topics related to Course with a given Advisor.*

Suppose that each thesis has a co-advisor, which is determined by the topics of the thesis and the advisor, and that an operation is needed to determine the co-advisor. This operation is expressed as a method (intensional predicate $co-adv$*). We suppose that to implement this method a cooperation with the division which stores information about courses must be performed, by accessing the attribute course.*

The cooperations among objects in the resulting database are expressed by the following relations

$$div_1 \Rightarrow_{plan} div_4$$
$$div_3 \Rightarrow_{year,average,miss\_ex} div_1$$
$$div_3 \Rightarrow_{role,course} div_2$$
$$div_4 \Rightarrow_{course} div_2 \qquad \diamond$$

4 Semantics of Obj-U-Datalog

The semantics of an Obj-U-Datalog program is given in three steps. Before introducing this semantics we remark that, in our approach, due to the query-update feature of the language, we are interested in modeling as observable property of a transaction the following information: the set of answers, the database state, and the result of the transaction itself (i.e. commit or abort). To model the transactional behavior we consider a two step computation which mimics the marking and update phases. The marking phase of a Obj-U-Datalog program has to model the answers to a given transaction. The answers to the marking phase are bindings and (hypothetical) updates.

In the marking phase we consider the semantics of the database as a collection of independent objects. Each object interacts with other objects only through explicit *context switches* i.e. requests for the evaluation of a subquery sent to another object. Note that each object consists of a time-varying component (its state) and a time-invaryng one (its methods). Therefore, the semantics of an object should be given in a compositional way (as done in [29] for U-Datalog). We do not consider this level of composition here, because each object regards the others as single units, being not interested in the subdivision between extensional and intensional components. In this phase we regard an Obj-U-Datalog program as a tuple of logic databases. The computation is performed in an object until an explicit message call moves it to another object. We define a bottom up semantics for the marking phase, based on a "parallel immediate consequence operator". This semantics is based on a composite structure for interpretation in which all the objects in the database are interpreted simultaneously. The bindings found during the entire computation are independent from the object in which they are found (i.e. logical variables are instantiated with respect to the entire database, and the bound ones are kept bound when the computation moves to another object). By contrast the gathered updates are kept related to the object that has generated them (on which they have to be performed), so we keep a tuple of sets of updates to be performed.

The first semantic step is related to Obj-U-Datalog as a declarative specification of queries and updates. The updates are not executed, neither the transactional behavior of a query is considered. The second semantic step, the update phase, performs the update and provides a transactional behavior to a query. The update phase receives as input the hypothetical updates computed by the marking phase and executes them. In addition the result of the transactional behavior, that is, commit or abort, is provided, altogether with a set of bindings to model the answers to a query. The third semantic step considers complex transactions, i.e. sequences of simple transactions. Complex transactions are in fact defined by means of elementary transactions such as update queries and the sequence constructor. Therefore the semantics of complex transactions must be defined in terms of the semantics of elementary ones.

In the following subsections we define the three step semantics. Each step uses the semantics defined in the previous step as input. The interpretation given to updates is the same as in U-Datalog [29], and for brevity reason we do not deal with it.

4.1 Marking Phase Semantics

In this first semantic step we consider the cooperation among objects, regarding an Obj-U-Datalog database as a collection of objects. Each object sees other objects as a single unit, without distinction between intensional and extensional components.

A well accepted notion to model the cooperation among modular logic programs (and therefore, possibly, among modular deductive databases) is the notion of open programs. Open programs were introduced in [12] to model the composition of logic programs with respect to a set of predicates denoted with Ω. An open program is a program whose information about the predicates in the set Ω are regarded as incomplete. Predicates in Ω can be extended with definitions provided from any other program. This notion is a very relevant one in modular logic programming. However, the cooperation among objects in an object database cannot be easily modeled with the compositional semantics of open programs. This is due to the fact that cooperation among objects is achieved through explicit *messages* (i.e. labeled atoms) to enforce the evaluation of the atom in a specified object. This is the concept of message passing in the object oriented paradigm, but it is quite different from the notion of composition for open logic programs. This difference is illustrated by the following example.

Example 4.1 *Consider the following object database:*

$$\begin{array}{rcl} obj_1 & = & \mathtt{k(a).} \\ & & \mathtt{q(X)} \leftarrow \mathtt{obj_2 : k(X).} \\ obj_2 & = & \mathtt{k(b).} \\ & & \mathtt{p(X)} \leftarrow \mathtt{k(X).} \end{array}$$

In this database predicate k is a "channel" of the communication between obj_1 and obj_2. This channel is inherently one-way, i.e. obj_1 uses the clauses for predicate k defined in obj_2, but not vice versa. With a compositional semantics, if we consider obj_1 and obj_2 as open programs -with $\Omega = \{k\}$-, both objects see the clauses for predicate k in the other object. Therefore in this latter case p(a) *is true in obj_2, while in our intended semantics it is not.*

Note however that the inadequacy of the compositional semantics to model the cooperation among object based on message passing is not only due to the asymmetricity of cooperation. Consider for instance the case of the following *database:*

$$\begin{array}{rcl} obj_1 & = & \texttt{k(a).} \\ & & \texttt{q(X)} \leftarrow \texttt{obj}_2 : \texttt{k(X).} \\ obj_2 & = & \texttt{k(b).} \\ & & \texttt{p(X)} \leftarrow \texttt{obj}_1 : \texttt{k(X).} \end{array}$$

In this case the cooperation is symmetric (both obj_1 and obj_2 may use the clauses for predicate k in the other object). However, also in this case the compositional semantics does not model the intended meaning of the database. In this case, in fact, if we consider obj_1 and obj_2 as open programs -with $\Omega = \{k\}$- we have that both q(a) *and* q(b) *are true in obj_1 and that both* p(a) *and* p(b) *are true in obj_2. But this is not the intended meaning of the above program. In fact an atom $obj_2 : k(X)$ requests the computation to move in object obj_2, seeing no longer clauses for k in the current object. Therefore the intended meaning for the above database is that* q(b) *(but not* q(a)*) is true in obj_1 and that* p(a) *(but not* p(b)*) is true in obj_2.* $\diamond$

As shown by the above example, the notion of composition among programs is inherently different from the notion of cooperation through message passing among objects. Therefore, in order to model the cooperation among objects, we consider an approach similar to the one introduced for object based logic programming in [14], for the ObjectLog language. This approach is based on the use of a composite structure for interpretations in which all the objects are interpreted simultaneously. In this approach an interpretation for a database $O{-}DB = obj_1, \ldots, obj_s$ is defined as the tuple of sets $\langle I(obj_1), \ldots, I(obj_s)\rangle$ where each $I(obj_i)$ is a subset of the Herbrand Base $\mathcal{B}$ that interprets the associate object obj_i. The Herbrand Base we consider is constituted by atoms of the form $H \leftarrow \bar{U}$, with H atom (either intensional or extensional) and $\bar{U}$ updates. The presence of the atom $H \leftarrow \bar{U}$ in the interpretation means that H is true and that its evaluation causes the execution of $\bar{U}$. The notion of Herbrand Base for U-Datalog has been introduced in [29]. We extend here this approach to labeled atoms. In the following we will refer to tuples of interpretations as T-interpretation. Let $\mathcal{I}$ be the class of T-interpretations.

Now, given an Obj-U-Datalog database $O - DB$ we define a transformation T_{O-DB} whose fixpoint is the semantics of $O-DB$. T_{O-DB} is defined in terms of the immediate consequence transformation of each of the objects in $O{-}DB$. The basic idea is that of computing the consequences in parallel on T-interpretations.

Definition 4.1 *The operator $T_{O-DB} : \mathcal{I} \rightarrow \mathcal{I}$ is defined in as follows:*
$T_{O-DB}(I) = \langle T_{obj_1}(I), \ldots, T_{obj_s}(I)\rangle$
where $I \in \mathcal{I}$ and for each i, $1 \leq i \leq s$, we have

$T_{obj_i}(I) = \{A \leftarrow \bar{U} \mid A \leftarrow \bar{U} \in \mathcal{B}$, $\exists$ *clause in* obj_i

$H \leftarrow U_1, \ldots, U_m, B_1, \ldots, B_n, obj_{k_1} : B_{n+1}, \ldots, obj_{k_w} : B_{n+w}$
$\exists\ \theta$ *substitution, such that* $A = H\theta$ *and*
$\forall r = 1 \ldots n \quad B_r\theta \leftarrow \bar{U}_r \in I(obj_i)$,
$\forall q = 1 \ldots w \quad B_{n+q}\theta \leftarrow \bar{U}_{n+q} \in I(obj_{k_q})$, *and*
$\bar{U} = obj_i : U_1\theta, \ldots, obj_i : U_m\theta, \bar{U}_1, \ldots, \bar{U}_n, \bar{U}_{n+1}, \ldots, \bar{U}_{n+w}$
is consistent $\}$

where a set of updates $obj_1 : u_{1_1}, \ldots, obj_1 : u_{1_{n_1}}, \ldots, obj_s : u_{s_1}, \ldots, obj_s : u_{s_{n_s}}$ *is consistent if it does not contain complementary updates (i.e.* $+p(\tilde{X})$ *and* $-p(\tilde{X})$*) labeled by the same object identifier, i.e. if for all* i, $1 \leq i \leq s$, *in* $u_{i_1}, \ldots, u_{i_{n_i}}$ *there are no complementary updates.*

The operator defined above is continuous and monotonic on the lattice $(\mathcal{I}, \subseteq)$. This allows us to define the fixpoint semantics for Obj-U-Datalog programs.

Definition 4.2 *Let* $O-DB$ *be an Obj-U-Datalog program. The fixpoint semantics* $\mathcal{F}(O-DB)$ *of* $O-DB$ *is defined as* $\mathcal{F}(O-DB) = T_{O-DB} \uparrow n$.

Note that the above fixpoint is reached in a finite number of steps (n) due to the finiteness of the domain [20].

Example 4.2 *Consider the database* $O-DB = \{obj_1, obj_2, obj_3\}$ *of Example 3.1. In the following we show the computation of* $\mathcal{F}(O-DB)$.

$$\begin{aligned}
T^0_{O-DB} = \langle\ & \{\ p(a),\quad q(b)\ \},\\
& \{\ g(a)\ \},\\
& \{\ s(a),\quad s(b),\quad n(X) \leftarrow obj_3 : +s(X)\ \}\ \rangle
\end{aligned}$$

$$\begin{aligned}
T^1_{O-DB} = \langle\ & \{\ r(b) \leftarrow obj_1 : +p(b)\ \} \cup T^0_{O-DB}(obj_1),\\
& \{\ k(a)\ \} \cup T^0_{O-DB}(obj_2),\\
& T^0_{O-DB}(obj_3)\ \rangle
\end{aligned}$$

$$\begin{aligned}
T^2_{O-DB} = \langle\ & \{\ k(a)\ \} \cup T^1_{O-DB}(obj_1),\\
& \{\ t(b) \leftarrow obj_1 : +p(b), obj_2 : +g(b)\ \} \cup T^1_{O-DB}(obj_2),\\
& T^1_{O-DB}(obj_3)\ \rangle
\end{aligned}$$

$$\begin{aligned}
T^3_{O-DB} = \langle\ & \{\ m(b) \leftarrow obj_1 : +p(b), obj_2 : +g(b)\\
& \quad m(a) \leftarrow obj_1 : +q(a)\ \} \cup T^2_{O-DB}(obj_1),\\
& \{\ m(a)\ \} \cup T^2_{O-DB}(obj_2),\\
& T^2_{O-DB}(obj_3)\ \rangle
\end{aligned}$$

$$
\begin{aligned}
T^4_{O-DB} = \langle\ & T^3_{O-DB}(obj_1), \\
& T^3_{O-DB}(obj_2), \\
& \{ w(a) \leftarrow obj_1 : +q(a)\ \} \cup T^3_{O-DB}(obj_3)\ \rangle
\end{aligned}
$$

Since no new facts can be derived, $T^4_{O-DB} = \mathcal{F}(O-DB)$*.*
The fixpoint semantics of the database is therefore

$$
\begin{aligned}
\mathcal{F}(O-DB) = \langle\ & \{\ p(a), \quad q(b), \quad k(a), \\
& \quad r(b) \leftarrow obj_1 : +p(b) \\
& \quad m(b) \leftarrow obj_1 : +p(b), obj_2 : +g(b) \\
& \quad m(a) \leftarrow obj_1 : +q(a)\ \}, \\
& \{\ g(a), \quad k(a), \quad m(a), \\
& \quad t(b) \leftarrow obj_1 : +p(b), obj_2 : +g(b)\ \}, \\
& \{\ s(a), \quad s(b), \\
& \quad n(X) \leftarrow obj_3 : +s(X) \\
& \quad w(a) \leftarrow obj_1 : +q(a)\ \}\ \rangle
\end{aligned}
$$

$\diamond$

Now we define the semantics of a query T with respect to an Obj-U-Datalog database O-DB. First we note that database systems use a default set-oriented semantics, that is, the query-answering process computes a set of answers [18]. We denote with $Set(T, O-DB)$ the set of pairs (bindings and updates) computed as answers to the transaction T. We first define the notion for atomic transaction, then we extend it to conjunction of atomic transactions (i.e. simple transactions).

Let T be an atom of the form $p(\tilde{t})$ with $\tilde{t}$ tuple of terms. Then

$Set(T, O-DB) = \{\ \langle b, \hat{u} \rangle \mid$
$\exists$ an atom $A \leftarrow \bar{U} \in \mathcal{F}(O-DB)$
$p(\tilde{t})\theta = A$, θ substitution,
b is the set of bindings correspondent to θ,
$\hat{u}$ is the tuple of update sets obtained from $\bar{U}$ grouping
the updates on the object to which they are related $\}$

where $\hat{u} = \langle u_1, \ldots, u_s \rangle$ is obtained from $\bar{U}$ in the following way: for each i, $1 \leq i \leq s$, $u_i = \{ua \mid obj_i : ua \text{ appears in } \bar{U}\}$.

Let T be a labeled atom of the form $obj_i : p(\tilde{t})$ with $\tilde{t}$ tuple of terms. Then

$Set(T, O-DB) = \{\ \langle b, \hat{u} \rangle \mid$
$\exists$ an atom $A \leftarrow \bar{U} \in \mathcal{F}(O-DB)(obj_i)$

(i.e. in the interpretation component related to obj_i)
$p(\tilde{t})\theta = A$, θ substitution, and
b is the set of bindings correspondent to θ,
$\hat{u}$ is the tuple of update sets obtained from $\bar{U}$ grouping
the updates on the object to which they are related }

Let T be a conjunction of atoms A^1, A^2 then

$$
\begin{aligned}
Set(T, O-DB) = \{\langle b, \hat{u}\rangle \mid \; & \\
& \exists \langle b^1, \hat{u}^1\rangle \in Set(A^1, O-DB), \\
& \exists \langle b^1, \hat{u}^2\rangle \in Set(A^2, O-DB), \\
& b = b^1 \cup b^2 \text{ is consistent and} \\
& \text{for each } i,\ 1 \leq i \leq s,\ u_i = u_i^1 \cup u_i^2 \text{ is consistent}\}.
\end{aligned}
$$

A set of bindings is consistent if the associated equation system is solvable, i.e. if the set associates at most a constant to each variable.

Example 4.3 *Consider the Obj-U-Datalog database $ODB = \langle obj_1, obj_2, obj_3\rangle$ of Example 3.1, whose semantics has been computed in Example 4.2.*

- *Consider the transaction* $\mathtt{m(X)}$.

$$
\begin{aligned}
Set(\mathtt{m(X)}, O-DB) \quad = \{ \quad & \langle\ \{\mathtt{X} = \mathtt{a}\}, \langle \emptyset, \emptyset, \emptyset\rangle\ \rangle, \\
& \langle\ \{\mathtt{X} = \mathtt{a}\}, \langle \{+\mathtt{q(a)}\}, \emptyset, \emptyset\rangle\ \rangle, \\
& \langle\ \{\mathtt{X} = \mathtt{b}\}, \langle \{+\mathtt{p(b)}\}, \{+\mathtt{g(b)}\}, \emptyset\rangle\ \rangle\ \}
\end{aligned}
$$

  Note that the first solution has been found in obj_2 (that is it belongs to $\mathcal{F}(O-DB)(obj_2)$), while the last two are found in obj_1.

- *Consider the transaction* $\mathtt{obj_2 : m(X)}$.
  $Set(\mathtt{obj_2 : m(X)}, O-DB) = \{\langle\ \{\mathtt{X} = \mathtt{a}\}, \langle \emptyset, \emptyset, \emptyset\rangle\ \rangle\ \}$

- *Consider the transaction* $\mathtt{obj_3 : m(X)}$.
  $Set(\mathtt{obj_3 : m(X)}, O-DB) = \emptyset$

- *Consider the transaction* $\mathtt{r(X), w(Y)}$.
  $Set(\mathtt{r(X), w(Y)}, O-DB) = \{\langle\ \{\mathtt{X} = \mathtt{b}, \mathtt{Y} = \mathtt{a}\}, \langle \{+\mathtt{p(b)}, +\mathtt{q(a)}\}, \emptyset, \emptyset\rangle\ \rangle\ \}$

- *Consider the transaction* $\mathtt{obj_3 : n(X)}$.
  $Set(\mathtt{obj_3 : n(X)}, O-DB) = \{\langle\ \emptyset, \langle \emptyset, \emptyset, \{+\mathtt{s(X)}\}\rangle\ \rangle\ \}$ $\diamond$

4.2 Update Phase Semantics

The semantics of the marking phase does not include the execution of the collected updates neither consider the transactional behavior. We now define a function which takes a set of ground updates, the current extensional components of the objects constituting the database and returns the new extensional components.

Definition 4.3 *Let $EDB_1^{i_1}, \ldots, EDB_s^{i_s}$ be the current extensional components of the objects constituting the database and $u_1, \ldots, u_s$ is a s-uple of consistent sets of ground updates. Then the new databases $EDB_1^{i_1+1}, \ldots, EDB_s^{i_s+1}$ are computed by means of the function $\Delta : \mathcal{EC} \times \mathcal{U} \rightarrow \mathcal{EC}$ as follows:*

$$\Delta(\langle EDB_1^{i_1}, \ldots, EDB_s^{i_s}\rangle, \langle u_1, \ldots, u_s\rangle) = \langle EDB_1^{i_1+1}, \ldots, EDB_s^{i_s+1}\rangle$$

where each $EDB_j^{i_j+1}$, with $j = 1 \ldots s$, is computed from $EDB_j^{i_j}$ and u_j as

$$(EDB_j^{i_j} \setminus \{p(\tilde{t}) \mid -p(\tilde{t}) \in u_j\}) \cup \{p(\tilde{t}') \mid +p(\tilde{t}') \in u_j\}$$

where $\mathcal{EC}$ denotes all the possible s-uples of extensional components (i.e. of sets of facts) and $\mathcal{U}$ denotes all the possible s-uples of updates sets.

Before introducing the update phase semantics we briefly recall that we are interested in modeling as observable property of a transaction the *set of answers*, the *object states* and *the result of the transaction* itself. It is called $Oss = \langle Ans, State, Res\rangle$ where Ans is the set of answers, $State$ is an s-uple constituted by the extensional components of objects in the database and Res is the transactional result, that is, either Commit or Abort. The set of possible observables Oss is OSS. In the following we define the semantics of a transaction T with respect to an object database $O - DB$ as a function from observables to observables.

Definition 4.4 *Let $O - DB^i$[1] be an Obj-U-Datalog database, with EDB^i the tuple of current object states and $O{-}IDB$ the tuple of method sets of objects. The semantics of a transaction is denoted by the function $\mathcal{S}_{O-IDB}(T) : \mathcal{EC} \rightarrow OSS$.*

$$\mathcal{S}_{O-IDB}(T)(EDB^i) = \begin{cases} Oss^{i+1} & \text{if } OK \\ \\ \langle \emptyset, EDB^i, Abort\rangle & otherwise \end{cases}$$

where $Oss^{i+1} = \langle\{b_j \mid \langle b_j, \hat{u}_j\rangle \in Set(T, O{-}DB_i)\}, EDB^{i+1}, Commit\rangle$, EDB^{i+1} is computed by means of $\Delta(EDB^i, \bar{u})$. The condition OK expresses the fact that all the components of the tuple of sets $\bar{u} = \bigcup_j \hat{u}_j b_j$ are consistent, that is, there are no complementary ground updates on the same object. $\hat{u}_j b_j$ denotes the ground updates obtained by substituting the variables in $\hat{u}_j$ with the ground terms associated with the variables in b_j.

[1] Here we denote with $O - DB^i$ the Obj-U-Datalog database to emphasize that we consider object states EDB^i at time i.

Note that, according to the above definition, in Obj-U-Datalog the abort of a transaction may be caused by two different situations. The first (which we may call *abort for ungroundness*) is related to a transaction that generates a non-ground set of updates. In this case we are not able to decide what updates to execute, and therefore we abort the transaction. The second situation (which we may call *abort for inconsistency*) is related to a transaction that generates an update set with complementary updates on the same atom in the same object (both the insertion and the deletion of the atom). In this case the resulting object state would depend on the execution order of updates, so we disallow this situation by aborting the transaction.

Example 4.4 *Consider the Obj-U-Datalog database of Example 3.1, whose semantics has been computed in Example 4.2. We consider as starting observable Oss^i the one composed by an empty set of bindings, the tuple of the current object states EDB^i and Commit. Recall that*

$EDB^i = \langle\ \{\ \mathtt{p(a)},\ \ \mathtt{q(b)}\ \},\ \{\ \mathtt{g(b)}\ \},\ \{\ \mathtt{s(a)},\ \ \mathtt{s(b)}\ \}\ \rangle.$

Consider the transaction $\mathtt{m(X)}$ *issued against* $O-DB^i$. *Recall from Example 4.3 that*

$$\begin{aligned} Set(\mathtt{m(X)}, O-DB) \quad = \{ \quad & \langle\ \{\mathtt{X}=\mathtt{a}\}, \langle\emptyset,\emptyset,\emptyset\rangle\ \rangle, \\ & \langle\ \{\mathtt{X}=\mathtt{a}\}, \langle\{+\mathtt{q(a)}\},\emptyset,\emptyset\rangle\ \rangle, \\ & \langle\ \{\mathtt{X}=\mathtt{b}\}, \langle\{+\mathtt{p(b)}\},\{+\mathtt{g(b)}\},\emptyset\rangle\ \rangle\ \} \end{aligned}$$

Therefore, $Oss^{i+1}.1 = \{\ \{\mathtt{X}=\mathtt{a}\},\ \ \{\mathtt{X}=\mathtt{b}\}\ \}$[2]. *The gathered updates are*

$\bar{u} = \langle\{+\mathtt{p(b)}, +\mathtt{q(a)}\}, \{+\mathtt{g(b)}\}, \emptyset\rangle.$

Each set in the triple $\bar{u}$ is ground and consistent, so $Oss^{i+1}.3 = Commit$. The resulting object states EDB^{i+1}, i.e. $Oss^{i+1}.2 = \Delta(EDB^i, \bar{u})$, are

$EDB^{i+1} = \langle\ \{\ \mathtt{p(a)},\ \ \mathtt{p(b)},\ \ \mathtt{q(a)},\ \ \mathtt{q(b)}\ \},\ \{\ \mathtt{g(a)},\ \ \mathtt{g(b)}\ \},\ \{\ \mathtt{s(a)},\ \ \mathtt{s(b)}\ \}\ \rangle.$

The obtained observable is therefore
$Oss^{i+1} = \langle\{\ \{\mathtt{X}=\mathtt{a}\},\ \ \{\mathtt{X}=\mathtt{b}\}\ \}, EDB^{i+1}, Commit\rangle$

Now consider the transaction $\mathtt{obj_3} : \mathtt{n(X)}$ *issued against* $O-DB^i$. *As seen in Example 4.3*

$Set(\mathtt{obj_3} : \mathtt{n(X)}, O-DB) = \{\langle\ \emptyset, \langle\emptyset,\emptyset,\{+\mathtt{s(X)}\}\rangle\ \rangle\ \}.$

In this case we have

$\bar{u} = \langle\emptyset,\emptyset,\{+\mathtt{s(X)}\}\rangle$

therefore the condition OK is not verified and the transaction aborts (abort for ungroundness). The resulting observable is $\langle\emptyset, EDB^i, Abort\rangle$. $\diamond$

[2] $Oss^i.n$ denotes the n-th component of the tuple Oss^i.

4.3 Complex Transactions Semantics

Complex transaction are not modeled by the semantics introduced so far. Note that, according to the following definition, the abort of a simple transaction in a sequence results in the abort of the entire sequence.

Definition 4.5 *Let $O - DB^i$ be an Obj-U-Datalog database, with EDB^i the tuple of current object states and $O - IDB$ the tuple of method sets of objects. If T_1 and T_2 are transactions then $T_1;T_2$ is a transaction. The semantics of $T_1;T_2$ is denoted by the function $\mathcal{S}'_{O-IDB}(T_1;T_2) : OSS \to OSS$. Let Oss^i be such that $Oss^i.2 = EDB^i$.*

$$\mathcal{S}'_{O-IDB}(T_1;T_2)(Oss^i) = \begin{cases} Oss^{i+2} & \text{if } OK \\ \\ \langle \emptyset, Oss^i.2, Abort \rangle & otherwise \end{cases}$$

where $Oss^{i+2} = \mathcal{S}_{O-IDB}(T_2)(Oss^{i+1})$. $Oss^{i+1} = \mathcal{S}_{O-IDB}(T_1)(Oss^i)$ represents the observable of the database after the transaction T_1 and OK expresses the condition that $\mathcal{S}_{O-IDB}(T_2)(Oss^{i+1}).3 = Commit$ and $\mathcal{S}_{O-IDB}(T_1)(Oss^i).3 = Commit$.

Note that the above semantics does not model the answers to the first transaction, i.e. T_1. We choose this approach to avoid keeping the histories of transactions.

Example 4.5 *Consider the database of Example 3.1, whose semantics has been computed in Example 4.2.*

- *Consider the transaction* $\mathtt{m(X);w(X)}$. *¿From Example 4.4 we have*
  $\mathcal{S}_{O-IDB}(\mathtt{m(X)})(EDB^i) = \langle \{ \ \{\mathtt{X = a}\}, \ \{\mathtt{X = b}\} \ \}, EDB^{i+1}, Commit \rangle$
  with
  $EDB^{i+1} = \langle \ \{ \ \mathtt{p(a)}, \ \mathtt{p(b)}, \ \mathtt{q(a)}, \ \mathtt{q(b)}\}, \ \{ \ \mathtt{g(a)}, \ \mathtt{g(b)}\}, \ \{ \ \mathtt{s(a)}, \ \mathtt{s(b)}\} \ \rangle$
  ¿From the semantics of $O - DB^{i+1}$, we have
  $$\begin{aligned} Set(\mathtt{w(X)}, O - DB^{i+1}) \ = \{ \ & \langle \ \{\mathtt{X = a}\}, \langle \{+\mathtt{p(a)}, +\mathtt{q(a)}\}, \{+\mathtt{g(a)}\}, \emptyset \rangle \ \rangle, \\ & \langle \ \{\mathtt{X = b}\}, \langle \{+\mathtt{p(b)}, +\mathtt{q(b)}\}, \{+\mathtt{g(b)}\}, \emptyset \rangle \ \rangle \ \}. \end{aligned}$$
  Therefore, $Oss^{i+2}.1 = \{ \ \{\mathtt{X = a}\}, \ \{\mathtt{X = b}\} \ \}$. The gathered updates are
  $$\bar{u} = \langle \{+\mathtt{p(a)}, +\mathtt{p(b)}, +\mathtt{q(a)}, +\mathtt{q(b)}\}, \{+\mathtt{g(a)}, +\mathtt{g(b)}\}, \emptyset \rangle.$$
  Each set in the triple $\bar{u}$ is ground and consistent, so $Oss^{i+2}.3 = Commit$. The resulting object states EDB^{i+2}, i.e. $Oss^{i+2}.2$, coincide with the previous ones EDB^{i+1}, because all the atoms in $\bar{u}$ to be inserted are already present in EDB^{i+1} and $\bar{u}$ contains no deletions. The obtained observable Oss^{i+2} is therefore
  $\mathcal{S}'_{O-IDB}(\mathtt{m(X);w(X)})(Oss^i) = \langle \{ \ \{\mathtt{X = a}\}, \ \{\mathtt{X = b}\} \ \}, EDB^{i+1}, Commit \rangle$

- *Consider now the sequence $m(X), n(X), w(X)$. As seen in Example 4.4*

  $\mathcal{S}_{O-IDB}(\mathtt{n(X)})(EDB^i) = \langle \emptyset, EDB^i, Abort \rangle$

  so, due to the abort of $\mathtt{n(X)}$*, the considered transaction also aborts. The resulting observable is* $\langle \emptyset, EDB^i, Abort \rangle$*. Note therefore that the resulting states are the ones before the transaction starts.* $\diamond$

5 Architecture of the Prototype

A prototype of the system has been implemented at the University of Genova, using KBMS1 [26], a knowledge base management system developed in HP laboratories at Bristol. The language of KBMS1, kbProlog, is an extension of Prolog with modularization facilities, declarative update operations and persistence support.

The implementation of the language has been realized in two steps:

- development of a translator from Obj-U-Datalog to U-Datalog
- development of a bottom up interpreter for U-Datalog [8].

The bottom-up interpreter for U-Datalog handles updates with a non-immediate semantics and provides the transactional behavior. The choice of implementing Obj-U-Datalog via a translation in U-Datalog is due to the fact that the definition and implementation of Obj-U-Datalog is part of a project which aims at developing an enhanced database language, equipped with an efficient implementation. We are currently developing several optimization techniques for U-Datalog [9] that will lead to an optimized U-Datalog interpreter and therefore to an optimized Obj-U-Datalog interpreter.

An alternative implementation might realize a "direct" interpreter for Obj-U-Datalog, adapting one of the several evaluation techniques developed for deductive databases [6] to object deductive databases (so taking into account both message passing and object state evolution). This is a possible issue for future investigation.

Our prototype is based on the following steps: *i)* an Obj-U-Datalog program OP is translated into a U-Datalog program UP; *ii)* each query on such a program OQ is first of all translated in a U-Datalog query UQ, and then executed against the program UP using the U-Datalog interpreter.

The translation from Obj-U-Datalog to U-Datalog is simple. For each object $obj_i \in O - DB$, for each predicate p of arity n defined in obj_i we have a corresponding predicate p of arity $n+1$ defined in U-Datalog DB. The argument added to each predicate refers to the object in which the predicate is defined.

The extensional component of an object obj_i, i.e. EDB_i, is translated as follows. For each fact in EDB_i, $p(\tilde{a})$, with $\tilde{a}$ tuple of constants, we have a fact $p(obj_i, \tilde{a})$ in DB. The extensional database of the U-Datalog program consists of the union of the translation of the extensional components of each object.

The intensional rules are translated as follows. Consider the rule, defined in object obj_i,

$p(\tilde{X}) \leftarrow B_1(\tilde{Y}_1), \ldots, B_k(\tilde{Y}_k), obj_1 : B_{k+1}(\tilde{Y}_{k+1}), \ldots, obj_n : B_{k+n}(\tilde{Y}_{k+n})$.
This rule is translated in the following U-Datalog rule:

$$\begin{aligned} p(obj_i, \tilde{X}) \leftarrow\ & B_1(obj_i, \tilde{Y}_1), \ldots, B_k(obj_i, \tilde{Y}_k), B_{k+1}(obj_1, \tilde{Y}_{k+1}), \ldots, \\ & B_{k+n}(obj_n, \tilde{Y}_{k+n}). \end{aligned}$$

The intensional database of the U-Datalog program consists of the union of the translations of all the rules of the intensional component of each object.

Transactions are translated in a very simple way. A transaction is translated in the conjunction of the translation of (eventually labeled) atoms that constitute it. A labeled atom $obj_i : p(\tilde{X})$ is translated in a U-Datalog atom $p(obj_i, \tilde{X})$. An unlabeled atom $p(\tilde{X})$ in a transaction, that -as we have seen- is interpreted as a transaction directed to the whole database, is translated in $p(O, \tilde{X})$, with O new variable. Note that in this way we obtain in the solution not only the instances of $p(\tilde{X})$ satisfied by the database, but also the object in which such instances were found.

Example 5.1 *In this example we show the translation in U-Datalog of the Obj-U-Datalog database of Example 3.1. The database resulting from the translation is the following.*

$$\begin{array}{rcll} EDB & = & \mathtt{p(obj_1, a).} & \mathtt{q(obj_1, b).} \\ & & \mathtt{g(obj_2, a).} & \\ & & \mathtt{s(obj_3, a).} & \mathtt{s(obj_3, b).} \\ & & & \\ IDB & = & \mathtt{k(obj_1, X) \leftarrow p(obj_1, X), k(obj_2, X).} & \\ & & \mathtt{r(obj_1, X) \leftarrow +p(obj_1, X), q(obj_1, X).} & \\ & & \mathtt{m(obj_1, X) \leftarrow +q(obj_1, X), k(obj_1, X).} & \\ & & \mathtt{m(obj_1, X) \leftarrow s(obj_3, X), t(obj_2, X).} & \\ & & \mathtt{k(obj_2, X) \leftarrow g(obj_2, X), s(obj_3, X).} & \\ & & \mathtt{t(obj_2, X) \leftarrow +g(obj_2, X), r(obj_1, X).} & \\ & & \mathtt{m(obj_2, X) \leftarrow k(obj_2, X), k(obj_1, X).} & \\ & & \mathtt{w(obj_3, X) \leftarrow m(obj_1, X), m(obj_2, X).} & \\ & & \mathtt{n(obj_3, X) \leftarrow +s(obj_3, X).} & \end{array}$$

The transaction $\mathtt{m(X)}$ *is translated into the U-Datalog transaction* $\mathtt{m(O, X)}$*, while the transaction* $\mathtt{obj_1 : m(X)}$ *is translated into* $\mathtt{m(obj_1, X)}$*.* ◇

Finally, note that if we had not supported general transactions of the form $p(\tilde{X})$, the translation of a program would have been even simpler, in that it suffices to rename each predicate p defined in obj_i with p^i -without modifying its arity- and take as translated program the union of the objects.

6 Conclusions and Future Work

In this paper we have proposed a deductive object based language for databases. This language may be used for specifying modular cooperating databases in a deductive framework. This language models communication through message passing among objects and object state evolution with a transactional behavior. We propose a direct semantics for the language which captures both the computation based on message passing and the transactional behavior. Future works include the extension of the language in order to provide dynamic channels, the notion of class, inheritance[3] and integrity constraints. Moreover, we want to investigate transaction optimization. ¿From the implementation viewpoint we aim to adapt classical Datalog evaluation techniques to Obj-U-Datalog to implement a more efficient Obj-U-Datalog interpreter.

References

[1] S. Abiteboul. Updates, a New Frontier. In M. Gyssens, J.Paredaens, and D. Van Gucht, editors, *Proc. Second Int'l Conf. on Database Theory*, volume 326 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, Berlin, 1988.

[2] S. Abiteboul and P. Kanellakis. Object Identity as a Query Language Primitive. In *Proc. Int'l ACM Conf. on Management of Data*, pages 159–173, 1989.

[3] S. Abiteboul and V. Vianu. Procedural and Declarative Database Update Languages. In *Proc. of the ACM Symposium on Principles of Database Systems*, pages 240–251. ACM, New York, USA, 1988.

[4] S. Abiteboul et al. Methods and Rules. In P. Buneman and S. Jajodia, editors, *Proc. Int'l ACM Conf. on Management of Data*, pages 32–41, 1993.

[5] J. M. Andreoli and R. Pareschi. LO and behold! Concurrent Structured Processes. In N. Meyrowitz, editor, *Proc. Int'l Conf. on Object-Oriented Programming: Systems, Languages, and Applications*, pages 1–13, 1990.

[6] F. Bancilhon and R. Ramakrishnan. Performance Evaluation of Data Intensive Logic Programs. In J. Minker, editor, *Foundation of Deductive Databases and Logic Programming*, pages 439–519. Morgan-Kaufmann, 1987.

[3] In [7] we have proposed a logical object oriented language for databases, with the notions of classes ad inheritance, but the semantics for the language has been defined indirectly, based on the semantics of U-Datalog.

[7] E. Bertino, B. Catania, G. Guerrini. An Overview of $\mathcal{LOL}$: A Deductive Language for Object Bases. Invited paper in A. Makinouchi, editor, *Proc. International Symposium on Next Generation Database Systems*, pages 69-76, 1993.

[8] E. Bertino, B. Catania, G. Guerrini, and D. Montesi. A Bottom-Up Interpreter for a Database Language with Updates and Transactions. Submitted for pubblication, 1993.

[9] E. Bertino, B. Catania, G. Guerrini, and D. Montesi. Transaction Optimization in Rule Databases. In J. Widom and S. Chakravarthy, editors, *Proc. Research Issues in Data Engineering - Active Database Systems Workshop*, 1994.

[10] E. Bertino, M. Martelli, D. Montesi. Modeling Database Updates with Constraint Logic Programming. In *Proc. 4th Int'l Workshop on Foundations of Models and Languages for Data and Objects: Modeling Database Dynamics*, 1992.

[11] E. Bertino and D. Montesi. Towards a Logical-Object Oriented Programming Language for Databases. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Proc. Third Int'l Conf. on Extending Database Technology*, pages 168–183, 1992.

[12] A. Bossi, M. Gabbrielli, G. Levi, and M. C. Meo. Contributions to the Semantics of Open Logic Programs. In *Proc. Int'l Conf. on Fifth Generation Computer Systems*, pages 570–580. Institute for New Generation Computer Technology, 1992.

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

[14] M. Bugliesi. A Declarative View of Inheritance in Logic Programming. In K.R. Apt, editor, *Proc. Joint Int'l Conf. and Symp. on Logic Programming*, pages 113–127, 1992.

[15] M. Bugliesi, P. Mello, and E. Lamma. Modularity in Logic Programming. Technical Report P/4/242, CNR, 1993.

[16] F. Cacace, S. Ceri, S. Crespi-Reghizzi, L. Tanca, and R. Zicari. The Logres project: Integrating Object-Oriented Data Modelling with a Rule-Based Programming Paradigm. Technical Report TR 89-039, Politecnico di Milano, 1989.

[17] S. Ceri, P. Fraternali, D. Montesi, and S. Paraboschi. Active Rule Management in Chimera. Technical Report, Unpublished, 1993.

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

[19] B. Freitag. A Deductive Database Language Supporting Modules. In *Proc. Second Int'l Computer Science Conference*, 1992.

[20] M. Gabbrielli, R. Giacobazzi, and D. Montesi. Modular Logic Programs on Finite Domain. *GULP Conference on Logic Programming*, 663–678, 1993.

[21] S. Greco, N. Leone, and P. Rullo. COMPLEX: An Object-Oriented Logic Programming System. *IEEE Transactions on Knowledge and Data Eng.*, 4(4):344–359, August 1990.

[22] R. Jungclaus, G. Saake, and C. Sernadas. Using Active Objects for Query Processing. In *Proc. Object-Oriented Database: Analysis, Design and Construction, 4th IFIP Working Conference DS-4*, 1990.

[23] Y. Lou and Z. M. Ozsoyoglu. LLO: An Object-Oriented Deductive Language with Methods and Methods Inheritance. In *Proc. Int'l ACM Conf. on Management of Data*, pages 198–207, 1991.

[24] P. Mancarella and D. Pedreschi. An Algebra of Logic Programs. In R.A. Kowalski and K.A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 1006–1023. The MIT Press, Cambridge, Mass., 1988.

[25] S. Manchanda and D. S. Warren. A Logic-based Language for Database Updates. In J. Minker, editor, *Foundation of Deductive Databases and Logic Programming*, pages 363–394. Morgan-Kaufmann, 1987.

[26] J. Mantley, A. Cox, K. Harrison, M. Syrett, and D. Wells. *KBMS1 A User Manual.* Information System Centre Hewlett-Packard Laboratories, 1990.

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

[28] L. Monteiro and A. Porto. Contextual Logic Programming. In G. Levi and M. Martelli, editors, *Proc. Sixth Int'l Conf. on Logic Programming*, pages 284–302. The MIT Press, Cambridge, Mass., 1989.

[29] D. Montesi. *A Model for Updates and Transactions in Deductive Databases.* PhD thesis, Dipartimento di Informatica, Università di Pisa, March 1993.

[30] S. Naqvi and S. Tsur. *A Logic Language for Data and Knowledge Bases.* Computer Science Press, 1989.

[31] J. D. Ullman. *Database and Knowledge-Base Systems.* Computer Science Press, 1989.

[32] P. Wegner. Dimensions of Object-Based Language Design. *Proc. Int'l Conf. on Object-Oriented Programming: Systems, Languages, and Applications*, pages 181–192, 1987.

[33] M. Zloof. Query-by-example: a Data Base Language. *IBM Systems Journal*, 16(4):324–343, 1977.