A Deductive Method for Entity-Relationship Modeling
(Extended Abstract)

Giuseppe Di Battista and Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza"
Via Buonarroti 12, 00185 Roma, Italia

Abstract

We present a simple entity-relationship-oriented model, which essentially includes the notion of class, together with different types of relations among classes, such as is-a, part-of, and disjointness. We define the semantics of the model in terms of first order logic, and present a sound, complete, and efficient inference algorithm for such a model. We argue that our model and the associated inference capabilities provide a suitable formal basis for designing an effective environment supporting conceptual modeling.

1 Introduction

Recent works on data base modeling show a growing interest in the object-oriented paradigm (see, for instance, [Ul87]). One of the basic motivations for such an interest comes from the need of embedding several types of abstraction capabilities into the existing data models. Object-oriented data models provide many concepts which seem particularly suited for such a requirement. Perhaps, the most important one is inheritance, which represents a powerful mechanism for explicitely stating that the properties of a class propagate over other related classes.

It is interesting to note that the above objectives were already present in the research on semantic data models (see [HK87]), which has addressed many problems related to the semantic aspects of data description, with the specific goals of defining the basic modeling primitives needed in a data base formalism, and studying their characteristics. We believe that one novelty of the investigation on object-orientation is the concern on the deductive capabilities of the data model, especially those related to inheritance.

Recent works (see [AM86], [MZ86], [AS87], [Le87a], [Le87b]) have carried out an investigation on the basic modeling primitives of object-oriented data models.

Proceedings of the Fifteenth International Conference on Very Large Data Bases

Amsterdam, 1989

All of these works deal with simple representational mechanisms, which essentially include the notion of class (an abstraction for a set of objects) and various forms of containment assertions on classes, with the main goals of investigating on the inherent complexity of reasoning about these assertions, and providing algorithms for the inference problem. For example, Arisawa and Miura [AM86] consider assertions of the form: (A and B and C is-a D and E), which states that the intersection of the (set of instances of the) classes A,B and C is contained in the intersection of D and E, and propose a polynomial time algorithm for computing inferences on these assertions. Aggregation relations on classes, allowing for representing a class as a property of another class, as well as other features of object-oriented data models, related to the definition of operations on classes, are not taken into account in these works.

The goal of this paper is to go one step further in the above direction, by considering aggregation relations among classes. In particular, we present a simple object-oriented data model, called ERL, which allows one to deal with entities (simple classes) and relationships (aggregated classes), and provides several assertional mechanisms for describing how such classes relate to each other. Both positive assertions (stating that a certain relation holds among classes) and negative assertions (stating that a relation does not hold) are expressible in this language.

Four types of basic semantic relations are taken into account, namely subsetting, disjointness, typing, and mandatory participation of an entity in a given relationship. For example, we can assert in ERL that the set of instances of a given class A is included in the set of instances of another class B (subsetting), or that classes A and B cannot have common instances (disjointness). As an example of negative assertion, we can state that A and B are not disjoint, i.e. there is at least one object that is both an instance of A and an instance of B. Typing allows for asserting that objects participating in a given relationship B are instances of an entity A. Conversely, we can state that every instance of A participates in at least one instance of B (mandatory participation).

Our modeling language is formally defined in terms of first order logic. Besides providing the formal tool for expressing the semantics of the language, logic allows us to devise sound and complete inference procedures for ERL. In fact, we present a sound, complete, and polynomial time

algorithm for computing inferences on a set of ERL assertions.

It is our opinion that the resulting inference technique may constitute the formal basis for building a sound and effective environment for conceptual modeling. During the past years, many efforts have been devoted to provide the data base designer with useful methodologies and tools supporting her/his activities. Many of these works are based on the entity-relationship data model (see, for example, [Ce83], [BLN86], [TYF86]). These proposals are often based on pragmatical criteria for achieving "good" qualities of the conceptual schema, such as consistency, clarity, and minimality. In our approach, a formal definition of such concepts is provided, based on the precise definition of the semantics of the language. Moreover, the inference techniques associated with the language can be directly used to build a system which is able to automatically check the representation against several correctness and minimality criteria, and to support the designer in all the activities requiring reasoning about the description, and exploring different modeling choices.

The paper is organised as follows. In Section 2 we informally define the language, and briefly discuss its expressive power. In Section 3, we present the formal definition of the language. In Section 4, we describe our method for performing inference on a set of assertions expressed in the language. In Section 5, we briefly discuss the use of the inference technique in the definition of an environment supporting entity-relationship modeling. Finally, in Section 6 we present the conclusions and outline future developments of our research.

In the remainder of the paper, we assume that the reader is familiar with the basic notions of first order logic (at the level of [Me64]). In this extended abstract, the proofs of the theorems are omitted (the interested reader is referred to [DL89] for the full paper).

2 An Entity-Relationship-based modeling language

In this section we informally discuss the basic characteristics of a simple ER-based modeling language, called ERL (Entity Relationship Language), which can be used to define entities and relationships and to state several types of assertions on how they relate to each other. The formal definition of ERL appears in section 3.

In ERL, the universe of discourse is partitioned into two levels, called extensional and intensional.

In the extensional level, both individual objects, and tuples of individual objects are represented. An individual object is an atomic object, identifiable through a unique name, whereas a tuple is an aggregation of individual objects. The number of objects which are components of a tuple is called the arity of the tuple. Each component is referenced by its position in the tuple: for example, the individual objects a,b and c are said to be the 1-component, 2-component and 3-component of the tuple <a,b,c>, respectively.

The objects belonging to the intensional level represent classes of either individual objects, or tuples. Classes are distinguished into entities and relationships.

An entity class (simply entity in the following) is a class of individual objects, whereas a relationship class (relationship) is a class of tuples of the same arity, which is called the arity of the relationship. The objects belonging to a given class constitute the set of instances of that class.

Note that, for the sake of simplicity, we do not explicitely deal with attributes, which are usually considered in the ER model. However, it is easy to see that what follows applies with minor changes to a language incorporating attributes.

In order to represent meaningful properties of entities and relationships, ERL provides the modeler with several types of assertions, which are discussed in the following.

Typing: Individual objects which are components of the tuples belonging to a given relationship, can be asserted to belong to a certain entity. For example, if Tutoring is a relationship of arity 2, we can assert that for each instance t of Tutoring, the 1-component of t is an instance of the entity Person, and the 2-component of t is an instance of the entity Course. In the classical ER model, typing corresponds to the fact that relationships are defined on a fixed collection of entities. Conversely, in ERL there is no limitation on the number of typing assertions for a given relationship: for example, the i-component of the tuples belonging to the relationship R, can be asserted to be both of type A and of type B.

Subsetting: One class can be asserted to be a subset of another class: in this case, every instance of the former, is also an instance of the latter. For each assertion of this type, we require the two classes to be either both entities, or both relationships of the same arity. For example, if Person and Student are entities, Tutoring and Summer_Tutoring are relationships of arity 2, we can assert Professor is subset of Person and Summer_Tutoring is subset of Tutoring. Notice that, in the literature, the subset relationship is often referred to as is-a relationship.

Disjointness: Two types of disjointness assertions can be expressed in ERL. Assertions of the first type are used to state that the extensions of two classes are disjoint. In this case, we require the two classes to be either both entities or both relationships of the same arity. For example we can state that the two entities Graduate_Student and Undergraduate_Student are disjoint. The second type of disjointness assertion can be used to state that the set of objects that are i-components of the tuples belonging to a given relationship is disjoint from the set of instances of a certain entity. Assertions of this type are used to represent the fact that an entity cannot participate in a given relationship in a specified role. For example, we can state that the instances of Undergraduate_Student cannot participate in the relationship Tutoring as 1-components.

Existence: The instances of a given entity can be

asserted to mandatorily participate in a certain relationship for a specified role. In other words, every instance of the entity must be the i-component of at least one instance of the relationship. For example, we can assert that the instances of the entity Teacher mandatorily participate in the relationship Course.

Negation: All the above assertions can appear in a negative form, in order to represent that a certain property does not hold. For example, we can assert that entity A is not related to B by means of the subset relationship, i.e. that there is at least one object which is included in the extension of A, but not in the extension of B. Another example of negative assertion is the one stating that A and B are not disjoint, i.e. there always exists one object that is in the extension of both classes. Notice that negative assertions are not usually considered in semantic data models.

In Section 3, we provide the formal definition of ERL syntax and semantics, whereas in Section 4 we address the problem of computing inference on ERL assertions. Such a problem consists of checking whether a set Σ of ERL assertions implies a single assertion σ, i.e. whether the fact that all the assertions in Σ hold, implies that σ holds as well. In Section 4, a polynomial time algorithm for this problem is presented.

Assertions of the type considered in ERL, have been taken into account in several recent works both on data bases, and knowledge representation. We already mentioned in Section 1 a number of works dealing with the problem of performing inference on containment assertions on classes. As we said before, such works do not consider aggregation relationships among classes.

The interaction of containment assertions with aggregation relationships has been considered in the context of the relational data base theory. We are referring, in particular, to those papers investigating the so-called inclusion dependencies. An inclusion dependency is a statement of the form: $R(A_1, ...,A_m) \supseteq S(B_1,...,B_m)$, where R and S are two relation schemes, and $A_1, ...,A_m,B_1,...,B_m$ are attributes. Such a dependency holds in a relational database D, if each of the tuples in the projection of s over $B_1, ...,B_m$, is also in the projection of r over $A_1,...,A_m$, where r and s are the relations in D corresponding to the relation schemes R and S, respectively.

In [CFP84], it is shown that the inference problem for inclusion dependencies is P-space complete in the general case. However, the problem can be solved in linear time if the dependencies are unary (m=1). This fact is exploited in [CaVi83], where the inference problem is studied for a language including a restricted form of inclusion and exclusion dependencies.

Notice that ERL do not limit its expressive power to unary dependencies. Indeed, it allows both the subset, and the disjointness relationship to be established both between entities, and between relationships. Moreover, ERL allows typing, existence, and negative assertions, to be expressed without any limitation. Nevertheless, the results reported in this paper show that the inference problem for ERL is tractable.

Inclusion dependencies have also been studied in their interaction with functional dependencies. Chandra and Vardi [CV85] have shown that the inference problem for a language including both dependencies, is undecidable; Kannellakis, Cosmadakis, and Vardi [KCV83], have shown that a polynomial time solution exists, if inclusion dependencies are restricted to be unary. Note that no construct is currently provided by ERL for expressing functional dependencies.

3 Logical formulation

In this section, we give a formal account of ERL in terms of first order logic. We rely on a first order language that includes constants, variable and predicate symbols. Constants and constant tuples correspond to individual and aggregated objects, respectively; 1-place and n-places (n>1) predicate symbols correspond to entities and relationships, respectively. Finally, the assertions discussed in Section 2 are expressed by means of sentences in the first order language.

Before delving into details, some remarks on the notation are in order. We shall use several metasymbols (possibly with subscripts), whose meaning is as follows: greek upper case letters for predicate symbols; x, y, z for variable symbols, and **x**, **y**, **z** for variable tuples; c, d for constant symbols, and **c**,**d** for constant tuples; w for constant or variable symbols, and **w** for tuples whose each component is either a constant or a variable; w^i for the i-th component of the tuple **w**. Finally, we write $\exists x$ instead of $\exists x_1,...,\exists x_n$ and $\forall x$ instead of $\forall x_1,...,\forall x_n$ (where $x=<x_1,...,x_n>$).

Formally, a set of ERL assertions is simply a first order theory (i.e., a first order language, plus a set of sentences in that language), as specified by the following definition.

Definition 1 An ERL-theory is a first order theory <L,A>, where L is a first order language consisting of a set of constant, variable and predicate symbols, and A, the axioms of the theory, is a set of formulas of L, with the constraint that each element of A has one of the following forms:

$\forall x (\neg \Phi_1(x) \vee \Phi_2(y))$ where **y** is a subtuple of **x**;

$\forall z (\neg \Phi_1(x) \vee \neg \Phi_2(y))$ where **x** and **y** are subtuples of **z**;

$\forall x \exists y (\neg \Phi_1(x) \vee \Phi_2(z))$ where **z** contains **x** and all the variables in **y**;

$\exists x (\Phi_1(x) \wedge \neg \Phi_2(y))$ where **y** is a subtuple of **x**;

$\exists z (\Phi_1(x) \wedge \Phi_2(y))$ where **x** and **y** are subtuples

of z;

$\exists x \forall y \ (\Phi_1(x) \wedge \neg \Phi_2(z))$) where z contains x and all the variables in y;

It is well known that the notion of interpretation is used for assigning semantics to any logical formula (see [Me79]). As far as ERL-theories are concerned, the notion of interpretation specialises as follows.

Definition 2 An interpretation for an ERL-theory T is a pair <DOM,EXT>, where DOM is a finite set, called the domain of the interpretation, and EXT is a mapping assigning an element of DOM to each constant symbol of T, and a subset of DOM^n to each predicate symbol P of T (n is the arity of P).

Taking into account the semantics of the axioms expressible in ERL-theories, we now show how they are used to express typing, subsetting, disjointness, existence, and negative assertions.

Axioms of the type:

$$\forall x \ (\neg \Phi_1(x) \ \vee \ \Phi_2(y)),$$

can be used to express both typing and subsetting. In particular, if Φ_2 is a 1-place predicate symbol representing an entity ($|y| = 1$), Φ_1 is a n-place predicate symbol representing a relationship, and $x^i = y$, then the above axiom specifies that every i-component of Φ_1 instances, is an instance of Φ_2, i.e. that the i-components of the relationship Φ_1 are typed with entity Φ_2. For example, the relationship Tutoring can be typed with entities Person and Course as follows:

$$\forall x_1, x_2 \ (\neg \text{Tutoring}(x_1, x_2) \vee \text{Person}(x_1))$$

$$\forall x_1, x_2 \ (\neg \text{Tutoring}(x_1, x_2) \vee \text{Course}(x_2)).$$

Observe that the above axiom allows a more general form of typing than the one discussed in Section 2. Indeed, the cardinality of y needs not to be 1 and, therefore, the axiom may express a dependency between two relationships.

If x = y, then the axiom expresses subsetting, on either entities ($|x| = 1$), or relationships ($|x| > 1$). For example, in order to express the subset relationship between entities Professor and Person, and between relationships Summer_Tutoring and Tutoring, we can use the following axioms:

$$\forall x_1 \ (\neg \text{Professor}(x_1) \vee \text{Person}(x_1)),$$

$$\forall x_1, x_2 \ (\neg \text{Summer\_Tutoring}(x_1, x_2) \vee \text{Tutoring}(x_1, x_2)).$$

Axioms of the type:

$$\forall z \ (\neg \Phi_1(x) \ \vee \neg \Phi_2(y))$$

are disjointness axioms. If z = x = y, then the axiom expresses the disjointness relationship between either two entities ($|x| = 1$), or two relationships ($|x| > 1$). For example, disjointness between the entities Graduate_Student and Undergraduate_Student, and between the relationships Tutoring and Attends, is expressed as:

$$\forall x_1 \ (\neg \text{Graduate\_Student}(x_1) \vee \neg \text{Undergraduate\_Student}(x_1))$$

$$\forall x_1, x_2 \ (\neg \text{Tutoring}(x_1, x_2) \vee \neg \text{Attends}(x_1, x_2)).$$

Disjointness relationships between entities and relationships can be expressed by imposing $|x| = 1$, and z = y. For example the disjointness relationship between Undergraduate_Student and Tutoring in the 1-component, is expressed by means of the following axiom:

$$\forall x_1, x_2 \ (\neg \text{Undergraduate\_Student}(x_1) \vee \neg \text{Tutoring}(x_1, x_2)).$$

Axioms of the type:

$$\forall x \exists y \ (\neg \Phi_1(x) \ \vee \ \Phi_2(z)),$$

express existence assertions, where Φ_1 represents an entity ($|x| = 1$) and Φ_2 represents a relationship ($|z| > 1$). For example, the mandatory participation of the entity Course in the relationship Teaching as 1-component, is expressed by:

$$\forall x \exists y \ (\neg \text{Course}(x) \ \vee \ \text{Teaching}\ (x,y)).$$

Up to now, we have discussed the first three forms of allowable axioms in ERL-theory. With ragard to negative assertions, it is easy to see that they correspond to the last three forms reported in definition 1. As an example, the fact that it is not true that Tutoring is typed with the entity Professor as 1-component, can be expressed by the following axiom:

$$\exists x \exists y \ (\text{Tutoring}(x,y) \ \wedge \neg \text{Professor}(y)).$$

After the formal definition of ERL, we are now ready to address the main point of this paper, which is the problem of computing inference on ERL-theories. Such a problem is dealt with in the next section, in which we present a sound and complete inference algorithm for ERL-theories.

4 Checking ERL-theories for satisfiability

Our method for computing inference on ERL-theories exploits the notion of satisfiability in logic. Recall that a theory is said to be satisfiable, if it admits at least one model (i.e. an interpretation which satisfies all the axioms of the theory). A theory which is not satisfiable is said to be unsatisfiable. Logical implication is the problem of determining whether a given sentence τ is true in every model of a theory T. Such a problem can be reduced to the one of checking a theory for unsatisfiability: T logically implies τ, just in case there cannot be models of T which satisfy $\neg\tau$, i.e. $T \cup \{\neg\tau\}$ is unsatisfiable. Notice that if T is an ERL-theory, and τ is any formula which has the form of an admissible axiom for T, then $T \cup \{\neg\tau\}$ is an ERL-theory as well.

The goal of this section is to present an efficient

algorithm for checking an ERL-theory for satisfiability. The above considerations show that such an algorithm can be used to perform inference on ERL-theories.

4.1 Clausal form for ERL-theories

The method we present requires an ERL-theory to be transformed into a set of clauses, which are in turn represented by means of a graph. A clause is simply a disjunction of literals, each literal being either an atomic formula (i.e. a predicate symbol with the corresponding arguments, called positive literal), or the negation of an atomic formula (negative literal). A negative (positive) clause is a clause constituted by negative (positive) literals. All the variables appearing in a clause are implicitely universally quantified. Well known results in first order logic ensure us that any set of formulas can be transformed into a set of clauses which is satisfiable if and only if the original set is satisfiable.

Table 1 shows the rules for obtaining a set of clauses from the axioms of an ERL-theory (axioms are on the left and the corresponding clauses on the right).

$\forall x\,(\neg\,\Phi_1(x)\ \vee\ \Phi_2(y))$	$\neg\,\Phi_1(x) \vee \Phi_2(y)$
$\forall z\,(\neg\,\Phi_1(x)\ \vee \neg\,\Phi_2(y))$	$\neg\,\Phi_1(x) \vee \neg\,\Phi_2(y)$
$\forall x\,\exists y\,(\neg\,\Phi_1(x)\ \vee\ \Phi_2(z))$	$\neg\,\Phi_1(x) \vee \Phi_2(v)$
$\exists x\,(\Phi_1(x)\ \wedge \neg\,\Phi_2(y))$	$\Phi_1(c),\ \neg\,\Phi_2(d)$
$\exists z\,(\Phi_1(x)\ \wedge \Phi_2(y))$	$\Phi_1(c),\ \Phi_2(d)$
$\exists x\,\forall y\ (\Phi_1(x)\ \wedge \neg\,\Phi_2(z)))$	$\Phi_1(c),\ \neg\,\Phi_2(w)$

Table 1

The first two axiom forms are easily translated into clauses. Due to the presence of existential quantifiers, the remaining four forms require the introduction of Skolem symbols (constants and functions). In the third clause form, v indicates a tuple that is obtained from z by leaving unchanged the x variables, and substituting each y^i with $f_i^j(x)$, where f_i^j is a Skolem function and j is a number identifying the original axiom (we are assuming that the axioms of the ERL-theory are numbered). Concerning the fourth and fifth axiom forms, each of them is transformed into two clauses forms, that are simply obtained by substituting x and y with tuples of Skolem constants, c and d, of the same arity. Finally, in the sixth form, c is a tuple of Skolem constants of the same arity of x, and w is obtained from z by substituting the x variables with the coresponding constants of c.

Example 1 We show how to transform a simple ERL-theory into the corresponding set of clauses:

$\forall x\,(\neg$Professor(x) $\vee$ Person(x)$)$
$\forall x\,(\neg$Student(x) $\vee$ Person(x)$)$
$\forall x\forall y\,(\neg$Undergraduate_Student(x) $\vee \neg$Tutoring(x,y)$)$
$\exists x\exists y\,($Tutoring(x,y) $\wedge \neg$Professor(y)$)$
$\forall x\exists y\,(\neg$Course(x) $\vee$ Tutoring(x,y)$)$

$\neg$Professor(x) $\vee$ Person(x)
$\neg$Student(x) $\vee$ Person(x)
$\neg$Undergraduate_Student(x) $\vee \neg$Tutoring(x,y)
Tutoring(d_1,d_2)
$\neg$Professor(d_2)
$\neg$Course(x) $\vee$ Tutoring(x,f_2^5(x)).

In what follows, we call ERL-clause any clause obtained from an axiom of an ERL-theory by means of the above transformation. Notice that ERL-clauses are Horn clauses, i.e. they contain at most one positive literal. Moreover, due to the presence of function symbols, the Herbrand universe of a set of ERL-clauses is infinite.

The following lemma allows us to transform a set of ERL-clauses into another set, in which every Skolem function is replaced by a constant symbol. Obviously, the Herbrand universe of the resulting set is finite.

Lemma 1 Let C be a set of ERL-clauses, and C' the set of ERL-clauses obtained from C by substituting each $f_i^j(x)$ with a new constant symbol z_i^j. Then C is satisfiable if and only if C' is satisfiable.

In the following, we take advantage of Lemma 1, and consider only ERL-clauses in which Skolem functions have been eliminated.

4.2 The algorithm for satisfiability checking

Our algorithm for satisfiability checking is based on a variant of the resolution. Resolution is an inference rule that has been shown to be sound and complete for the clausal form of logic (see [CL73]): a resolution refutation (i.e. a derivation of the empty clause obtained by repeated applications of resolution steps) exists for a set of clauses if and only if such a set is unsatisfiable. Positive unit resolution (see [CL73]) is a special form of resolution, requiring every step to be applied to at least one positive unit clause, i.e. a clause containing just one positive literal. For example, a positive unit resolution step can be applied to A(c) and $(\neg$A(x) $\vee$ B(x)$)$, yielding the positive clause B(c).

Definition 3 Let C be a set of ERL-clauses. A positive derivation chain (or simply a chain) of C from P(c) to Q(d) is a sequence $\langle\sigma_1,\ldots,\sigma_n\rangle$ of literals, with $\sigma_i \neq \sigma_j$ for $i \neq j$, where σ_1 is P(c), σ_n is Q(d), and for each i (i=2,...,n), σ_i is obtained by positive unit resolution from a clause of C of the form $(\neg\Phi_1(w_1) \vee \Phi_2(w_2))$ and from

σ_{i-1}; n is called the length of the chain $<\sigma_1,...,\sigma_n>$.

The following theorem establishes an important property of ERL-clauses, namely that we can check any set C of ERL-clauses for unsatisfiability, by looking for particular positive derivation chains of C.

Theorem 2 A set C of ERL-clauses is unsatisfiable if and only if it includes a negative clause $\Gamma = (\neg\Phi_1(w_1) \vee \neg\Phi_2(w_2))$ of C such that there is a chain of C from $\psi_1(c_1)$ to $\Phi_1(d_1)$, and a chain of C from $\psi_2(c_2)$ to $\Phi_2(d_2)$, where $\psi_1(c_1)$ and $\psi_2(c_2)$ are positive unit clauses of C, and $(\neg\Phi_1(d_1) \vee \neg\Phi_2(d_2))$ is a ground instance of Γ.

We are now in a position to define the concept of graph associated with a set of ERL-clauses, which is used by our algorithm for satisfiability checking.

Definition 4 Given a set of ERL-clauses C, we associate with it a directed graph G(P,D,I,A), called ERL-graph, labeled on nodes and arcs. The set of nodes is partitioned into three subsets, called P, D, and I; A is the set of arcs. P-nodes, I-nodes, and D-nodes are called predicate nodes, implication nodes, and disjointness nodes, respectively. The ERL-graph G(P,D,I,A) is defined according to the following rules:

1. for each predicate Q of C define a P-node, labeled with Q;
2. for each clause of the form $(\neg\Phi(w))$, define a D-node d and an arc $e=<\Phi,d>$, labeled with $u=<1,...,n>$, where $n=|w|$;
3. for each clause of the form $(\neg\Phi_1(x) \vee \neg\Phi_2(y))$, define a D-node d and arcs $e_1=<\Phi_1,d>$, and $e_2=<\Phi_2,d>$; the label u of e_1, and the label v of e_2 are defined as follows: u is constituted by all the variables of x that appear in y, whereas v is such that $|v| = |u|$, and for each i, $v^i = j$, where j is the integer such that $y^i = x^k$, where $k=u^i$.
4. for each clause of the form $(\neg\Phi_1(x) \vee \Phi_2(y))$, define an I-node m and arcs $<\Phi_1,m>$, labeled with u, and $<m,\Phi_2>$, labeled with v; u and v are defined as follows:

$u = <1,...,n>$, where $n = |x|$;
$v = <v^1,...,vP>$, where $p = |y|$, and, for each i:

$$v^i = \begin{cases} j & \text{if } y^i = x^j; \\ z_h^k & \text{if } y^i = z_h^k. \end{cases}$$

Example 2 The ERL-graph associated with the set of clauses of example 1 is shown in figure 1.

For an arc e, head(e) denotes the reaching node of e; for example, if $e=<n,m>$, then head(e)=m.

We now present an algorithm whose purpose is to store the information about the derivation chains of C (where C is a set of ERL-clauses) from a given literal P(c).

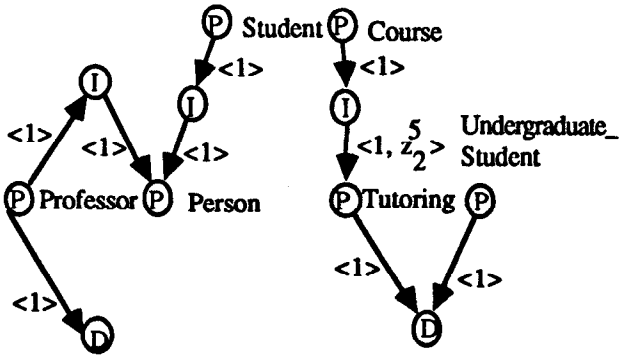

Figure 1

We assume that associated with each P-node Q of an ERL-graph, there is a set of tuples, denoted by TERM_SET(Q). The goal of the algorithm is to insert the tuple **d** into TERM_SET(Q) whenever there is a chain of C from P(**c**) to Q(**d**). The algorithm makes use of a function *select* which, given a constant tuple **c**, and an arc **e**, with label(e) = **u**, returns a new constant tuple **r** of the same arity as **u**, according to the following rule; select(c,e) = **r** where:

$$r^i = \begin{cases} c^j & \text{if } u^i=j \text{ (j integer)} \\ u^i & \text{if } u^i \text{ is a constant} \end{cases}$$

Algorithm chain(G,c,P)
Input an ERL-graph G corresponding to a set C of ERL-clauses, a function TERM_SET, a constant tuple **c**, and a node P of G
Output the algorithm inserts **d** in TERM_SET(Q), for each **d** and Q such that there is a chain of C from P(**c**) to Q(**d**)

```
begin
  if   P is a P-node and d∉ TERM_SET(P)
  then begin
         insert c into TERM_SET(P)
         for each outgoing arc e of P
         do chain(G, select(c,e), head(e))
       end
  else if P is an I-node with outgoing arc e
       then chain(G, select(c,e), head(e))
end
```

Theorem 3 shows that algorithm *chain* is correct.

Theorem 3 Let C be a set of ERL-clauses, G the associated graph, and P,Q two of its P-nodes. During the execution of chain(G,c,P), **d** is inserted into TERM_SET(Q), if and only if there is a chain of C from P(**c**) to Q(**d**).

The algorithm *chain* is the basis of our method for checking a set C of ERL-clauses for unsatisfiability. Given the ERL-graph G, we set TERM_SET(P) to empty, for each P-node of G. Then we execute the algorithm *chain* for each c and P such that P(c) is a clause of C. Finally, we check a special condition on the D-nodes of G, as specified by the following algorithm.

Algorithm Unsatisfiable?(C)
Input a set C of ERL-clauses
Output true, if C is unsatisfiable, false otherwise
begin
construct the ERL-graph G associated with C
for each P-node P *do* set TERM_SET(P) to {}
for each positive clause (P(c)) in C *do* chain(G,c,P)
if there is a D-node D of G, with ingoing arcs e_1=<Q_1,D> and e_2=<Q_2,D> such that there is a h_1 in TERM_SET(Q_1) and a h_2 in TERM_SET(Q_2) such that select(h_1,e_1) = select(h_2,e_2)
or
there is a D-node D of G, associated with ¬Q(w), with one ingoing arc e=<Q,D> such that there is a h in TERM_SET(Q) such that select(h,e)=w', where w' is a ground instance of w
then return (true)
else return (false)
end

Theorem 4 Let C be a set of ERL-clauses. Then *Unsatisfiable?*(C) returns true if and only if C is unsatisfiable.

As far as the computational complexity of our method is concerned, the next theorem shows that algorithm *Unsatisfiable?* is polynomial in the size of the graph.

Theorem 5 Let C be a set of ERL-clauses. Algorithm *Unsatisfiable?* takes $O(E \cdot K \cdot N \cdot \log(K \cdot N))$ time, where E is the number of clauses of C, K is the number of constants of C, and N is the number of predicates of C.

5. An environment for conceptual modeling

The purpose of this section is to demonstrate that, using the language presented in Section 3 and the algorithm presented in Section 4, one can define an effective environment supporting conceptual modeling. The basic observation is that our method for satisfiability checking can be directly used to perform inferences on ERL-theories. Therefore, we can easily devise a deductive environment which allows the designer to check wether a given sentence is logically implied by an ERL-theory.

In what follows, we shall consider several crucial problems in conceptual modeling, and analyse the features provided by our approach in order to deal with these problems.

5.1 Consistency

All the methodologies and tools for conceptual modeling insist on the need for checking the consistency of the conceptual schema. However, such a concept is often defined using informal arguments, and the methodologies provide only some intuitive guidelines for achieving the consistency of the representation. In our approach, consistency can be naturally defined in terms of satisfiability. In particular, if we allow the ERL-theory to be built incrementally, the consistency of the schema can be automatically checked as follows: every time a new axiom α is added to an ERL-theory T, the system automatically checks T∪{α} for satisfiability, and the axiom is rejected if T∪{α} is unsatisfiable. Notice that in this case the system is also able to provide the designer with useful information about the contradictory axioms, in particular using the positive chains which lead to the contradiction (see [DL89]). In this case, the algorithm *Unsatisfiable?* is used for ensuring the global consistency of the representation.

Besides the global consistency, other consistency checks can be performed by the system. For example, an entity-relationship schema which is satisfiable may suffer from another anomaly, called class unsatisfiability ([LN87]). A class S (either an entity or a relationship) is said to be unsatisfiable in an ERL-theory T, if S has no instances in all the models of T. For example, in the following ERL-theory:

$\forall x \exists y (\neg A(x) \vee R(x,y))$
$\forall x, y (\neg B(x) \vee \neg R(x, y))$
$\forall x (\neg S(x) \vee A(x))$
$\forall x (\neg S(x) \vee B(x))$

the object class S is unsatisfiable, although the theory is satisfiable. In general, we can check if a class S is usatisfiable in T as follows: if C is the set of clauses corresponding to T, and z is a constant symbol not appearing in C, then S is unsatisfiable in T if and only if C∪{S(z)} is unsatisfiable. Notice that C∪{S(z)} is a set of ERL-clauses.

Finally, our environment can support prototyping facilities, which are very important for allowing the designer to directly test and verify the schema. To this end, it is easy to extend ERL with axioms of the type: Φ(c) and $\neg \Phi$(c), where c is a constant tuple. Such axioms assert that c (does not) belongs to an entity (relationship) Φ. Clearly the resulting set of clauses is still a set of ERL-clauses, and the method presented in section 4 can be used to perform inference on such a set.

In summary, the designer can incrementally build an ERL-theory, insert instances into classes, assert that an object is not an instance of a class, and pose queries to the system anytime she/he needs to know whether a certain

sentence logically follows from the defined ERL-theory.

5.2 Minimality

Another important quality of object descriptions is minimality. In our approach minimality can be formally defined in terms of derivability of axioms. An ERL-theory Σ is minimal if no axiom α of Σ can be derived from $\Sigma - \{\alpha\}$. Thus, it is possible to test if an ERL-theory Σ is minimal by simply testing if for all axioms $\alpha \in \Sigma$, $\Sigma - \{\alpha\} \cup \{\neg(\alpha)\}$ is satisfiable.

Example 3 Consider the fragment of entity relationship schema shown in figure 2, and suppose that the following properties hold for such a schema:

- the Scientific_Faculty_Enrollment relationship is a subset of the Enrollment relationship (represented by an arrow in the figure);
- Mandatory_Tutoring is a subset of Tutoring;
- relationship Tutoring is typed with Scientific_Faculty_Student in 1-component;
- Suspended_Student cannot participate in the relationship Enrollment;
- Scientific_Faculty_Student must participate in the relationship Scientific_Faculty_Enrollment (minimum cardinality equal to 1);
- Scientific_Faculty_PhD_Student must participate in the relationship Mandatory_Tutoring.

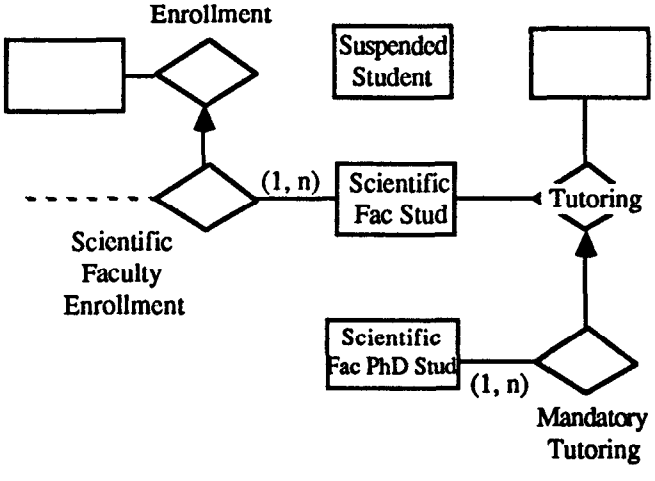

Figure 2

The schema and the above properties can be represented by the following ERL-theory T:

$\forall x_1, x_2 (\neg$ Scientific_Faculty_Enrollment$(x_1, x_2) \vee$ Enrollment$(x_1, x_2))$

$\forall x_1, x_2 (\neg$ Mandatory_Tutoring$(x_1, x_2) \vee$ Tutoring$(x_1, x_2))$

$\forall x_1, x_2 (\neg$ Tutoring$(x_1, x_2) \vee$ Scientific_Faculty_Student$(x_1))$

$\forall x_1, x_2 (\neg$ Suspended_Student$(x_2) \vee \neg$Enrollment$(x_1, x_2))$

$\forall x_1 \exists x_2 (\neg$ Scientific_Faculty_Student$(x_2) \vee$ Scientific_Faculty_Enrollment$(x_1, x_2))$

$\forall x_1 \exists x_2 (\neg$ Scientific_Faculty_PhD_Student$(x_1) \vee$ Mandatory Tutoring$(x_1, x_2))$.

Now, suppose we want to assert that the two entities Suspended_Student and Scientific_Faculty_PhD_Student are disjoint, i.e. we want to add to the theory the axiom :

$\forall x_1 (\neg$ Scientific_Faculty_PhD_Student $(x_1) \vee \neg$Suspended_Student$(x_1))$.

In order to maintain the theory minimal we have to check if such an assertion is already implied by the theory T, that is done by adding its negation to T, and checking the resulting theory T' for unsatisfiability. The negation of the above axiom is:

$\exists x_1$ (Scientific_Faculty_PhD_Student$(x_1) \wedge$ Suspended_Student(x_1)),

which in clausal form yields:

Scientific_Faculty_PhD_Student(c_1), and

Suspended_Student(c_1).

The ERL-graph corresponding to T' is shown in figure 3. Algorithm *chain* propagates the Skolem constant c_1 to the nodes Suspended_Student and Enrollment, which are connected to the disjointness node D. Therefore, the algorithm *Unsatisfiable?* returns true as result. Hence, we can conclude that the above disjointness axiom is impled by T.

Other types of minimality of the description can be considered by using ERL. For example, one may want to obtain the minimal set of subset relations which is implied by an ERL-theory T. In this case, we can first compute the set of all the subset relations implied by T, and then compute the transitive reduction of such a set. Again, this can be done by using the deductive capabilities of the system.

A further example of how minimality can be dealt with in ERL, concerns the existence of equivalent entities or relationships in the schema. Two classes A and B are said to be equivalent in T if they have the same set of instances in all the models of T. This condition can be checked by testing if both $\forall x (\neg B(x) \vee A(x))$ and $\forall x (\neg A(x) \vee B(x))$ are derivable from T.

6 Conclusions and further research

We have presented ERL, a simple object modeling language that allows one to express positive and negative assertions concerning several semantic relations among object classes.

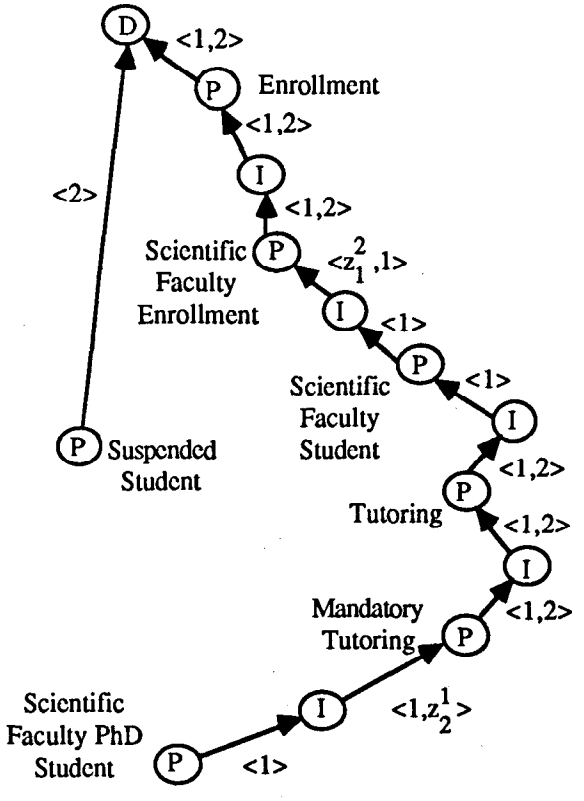

Figure 3

The semantics of such a language has been defined in terms of first order logic. A polynomial time algorithm has been provided for computing inference in ERL, which has been proven to be sound and complete with respect to the semantics. Our work extends previous results on containment inference in object-based models, by considering aggregation relations among classes. Moreover, we have shown how ERL and the associated inference capabilities can be used as a kernel for a well-founded environment for conceptual modeling.

In the future, we aim at extending the expressive power of ERL by allowing different levels of aggregation among classes, and by considering other meaningful semantic relations, with the constraint that such extensions do not affect the computational tractability of the inference problem.

References

[AM86] Arisawa H., and Miura T., "On the Properties of Extended Inclusion Dependencies", *Proc. of the 12th Very Large Data Bases (VLDB) Conference*, Kyoto, 1986.

[AS87] Atzeni P., and Stott Parker D., "Set Containment Inference", *Proc. of the International Conference on Database Theory, LNCS*, N.243, Springer-Verlag New York Inc., 1987.

[BLN86] Batini C., Lenzerini M., and Navathe S.B., "A Comparative Analysis of Methodologies for Database Schema Integration", *ACM Computing Surveys*, Vol. 18, N. 4, 1986.

[CFP84] Casanova M.A., Fagin R., and Papadimitriou C.H., "Inclusion Dependencies and Their Interaction with Functional Dependencies", *Journal of Computer and System Sciences*, 28, 1984.

[CaVi83] Casanova M.A., and Vidal V.M.P., "Towards a Sound View Integration Methodology", *Proc. of the 2nd ACM Symposium on Principles of Database Systems*, 1983.

[Ce83] Ceri S. (ed.), *Methodology and Tools for Database Design*, North Holland, 1983.

[CV85] Chandra A.K., and Vardi M.Y., "The Implication Problem for Functional and Inclusion Dependencies is Undecidable", *SIAM Journal on Computing*, Vol. 14, 1985.

[CL73] Chang C.L., and Lee R.C.T., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

[DL89] Di Battista G., and Lenzerini M., "A Deductive Method for Entity-Relationship Modeling", Tech. Rep. Dipartimento di Informatica e Sistemistica, Universita' di Roma "La Sapienza", 1989 .

[HK87] Hull R., and King R., "Semantic Database Modeling: Survey, Applications, and Research Issues", *ACM Computing Surveys*, Vol. 19, 1987.

[KCV83] Kannellakis P.C., Cosmadakis S.S., Vardi M.Y., "Unary Inclusion Dependencies Have Polynomial Time Inference Problems", *Proc. of the 15th ACM Symposium on Theory of Computer Science* , 1983.

[Le87a] Lenzerini M., "Covering and Disjointness Constraints in Type Networks", *Proc. of the 3rd IEEE International Conference on Data Engineering*, Los Angeles, Ca, February 1987.

[Le87b] Lenzerini M., "Class Hierarchies and their Complexity", *Proc. of the Workshop on Database Programming Languages* , Roscoff, 1987.

[LN87] Lenzerini M., and Nobili P., "On the Satisfiability of Dependency Constraints in Entity-Relationship Schemata", *Proc. of the 13th Conference on Very Large Data Bases* (VLDB), Brighton, UK, 1987.

[MZ86] McAllester D., and Zabih R., "Boolean Classes", *Proc. of ACM Object Oriented Prog. Systems, Languages and Applicatons (OOPSLA) Conference*, 1986.

[Me64] Mendelson E., *Introduction to Mathematical Logic*, Van Nostrand Reinhold, New York, 1964.

[TYF86] Teorey T.J., Yang D., and Fry J.P., "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", *ACM Computing Surveys*, Vol. 18, 2, 1986.

[Ul87] Ullman J.D., "Database Theory: Past and Future", *Proc. of ACM Conf. on Principles of Database Systems*, 1987.