1993

On Object Semantic similarity in Heterogeneous Database Integration

Xiangning Lui

Omran A. Bukhres

Report Number:
93-045

On Object Semantic Similarity in Heterogeneous Database Integration

Xiangning Liu and Omran A. Bukhres

CSD-TR-93-045
July 1993

On Object Semantic Similarity in Heterogeneous Database Integration

Xiangning Liu and Omran A. Bukhres

Department of Computer Science
Purdue University
West Lafayette, IN 47907
e-mail: {xl,bukhres}@cs.purdue.edu

Abstract

To provide users with a uniform interface to access federated database systems, local database schemas must be integrated to form a global schema. Before the integration, objects to be integrated need to be defined. Only semantically similar objects are integrated. In our paper, we define semantic similarity via a method which considers the semantics of objects in a database application context. The context of database is treated rigorously. We propose a measurement of semantic similarity and provide a taxonomy of semantic similarity. The advantage of this approach is that it establishes a foundation for formulating methods to determine which object schemas are semantically similar and need to be integrated.

1 Introduction

With the advances in the computer network technology, applications to access multidatabase systems in a uniform way have emerged. Users want to share data stored on isolated databases. This need is especially urgent for large organizations and companies. But the requirement has been hampered by three dimensions of characteristics of multidatabase systems which are: *heterogeneity, autonomy*, and *distribution* [ÖV91].

While earlier research focused on the distribution aspect of databases, current work in the database community now emphasizes the heterogeneity aspect of databases [She91]. The function of database integration is to integrate multiple databases in order to reduce the heterogeneity among component databases and to provide a global view of the federated database for users [SL90].

Database integration consists of system integration, schema integration, and semantics integration. *System integration* operates to hide the system heterogeneity such as hardware, operating system, and transaction management heterogeneities among component databases. *Schema integration* functions to provide a global view of the databases by integrating local schemas if the objects are semantically similar. *Semantics integration* acts to resolve semantic conflicts among data stored on multiple databases and to maintain consistency among interdependent data.

To integrate databases, many commercial and prototype federated database systems [BOT86, ANRS92, LR82, TLW87, BCD+93] have been developed. Research on federated database systems has also made major advances. Regarding system integration, significant understanding of transaction management in multidatabase systems has been achieved [BGMS92, DE89]. In many papers, such as [KS91, DH84, BOT86], database heterogeneity has been investigated from the viewpoint of reducing schematic conflicts. Compared with literature on systemic and schematic issues [BLN86], much less progress has been achieved on semantic issues [She91]. Not enough research work on semantic heterogeneity has been done, although semantic integration is one of the oldest and most difficult problems in multidatabase systems [ACM90].

Before schema integration, objects to be integrated need to be defined. With the achievement in schema integration, as well as high interconnectivity and access to many information sources, the primary issue in the future will not be the efficiency of data processing but the relevancy of the data [She91]. We only integrate semantically similar objects. Three tasks are involved in finding semantically similar objects: 1) resolving schematic heterogeneity; 2) identifying semantic similarity; and 3) recognizing

semantic discrepancy.

With consideration to both schematic and semantic aspects of multiple databases, a semantic equivalence theory was established in the papers [LNE89, EN84, NSE84]. The theory concerns attribute equivalence, object equivalence, and class equivalence. In regards to our work, semantic equivalence is a special case of semantic similarity. [SM91] describes a rule-based approach to semantic specification that can be used to establish semantic agreement between a database and applications. [FKN91] investigated the similarity of classes which utilizes *fuzzy* and *incomplete* terminological knowledge as well as schematic knowledge. [GPN91] introduced a mathematically based distinction between the structural and semantic aspects of class specification.

Recently, [SK92] introduced the concept of semantic proximity to specify degrees of semantic similarities among objects based on their real world semantics and used such a concept to propose a taxonomy of semantic similarity and schematic conflicts. Semantic proximity considered database semantics with respect to database *context*. The context was described by enumeration of some examples with no precise definition given. Additional work is needed to further clarify the nature and structure of the context to which two objects can belong.

In comparison to previous works, the major contributions of our paper are: 1) We present a method to define the nature of *context* of database applications. An object-oriented data model is used for the presentation. Database semantics is defined with respect to context. 2) We define precisely semantic similarity based on database semantics within the database context. 3) We propose a way to measure semantic similarity between objects. 4) We give a taxonomy of semantic similarity according to our definitions. The treatment of these topics is formal and generic.

Only with a strict, formal approach, can it be possible to establish an automatic or even a semi-automatic way to recognize semantic similarity between objects. A formal treatments serves several purposes:

- Facilitates in understanding of database and data semantics;
- Assists in federated database standardization;
- Supports federated database design;
- Aids in writing database integration tools;
- Accommodates the application verification and software reliability;
- Provides a model for database integration specification.

Formalization is not a panacea and should be used properly. Its major limitation is that real world applications are still difficult with the current technology [Mey90].

The paper is organized as follows: In section 2, we describe the notions and related concepts for defining database semantics with respect to the context. In section 3, we define semantic similarity according to the semantics defined in context and provide a taxonomy of semantic similarity. We also present a model to measure semantic similarity. In section 4, we delineate our conclusions and list some ideas for future work.

2 Database Semantics in Context

Basic significance of databases is to store and manipulate data by computers in an economical fashion. Data in databases is worthless if it does not mean anything or has no use. With respect to one database, data is manipulated according to certain semantic rules, and users usually interpret the data according to some implicit assumptions [Ken91]. When more than one database is considered, the rules and assumptions from different databases may conflict. Data stored in these databases can be interpreted in different ways by applications or by human users under different contexts. To capture the semantics of these data, context of database must be taken into account explicitly.

2.1 Modeling the Reality

Entities in the real world are modeled by data in computers to reflect the real world. Figure 1 shows steps to map from entities in the real world to the computer stored data. We regard the *real world* as one consisting of *entities.* An entity can be concrete, such as a book; or it can be abstract, such as 1, 2, ∞, big, or happy. The first step is to map the real world to the *abstract world.* The abstract world mathematically consists of sets. Every real world entity can be modeled as a set. The modeling is an isomorphism. Therefore, we can use an entity to refer either to a real world entity or to a corresponding set in the abstract world.

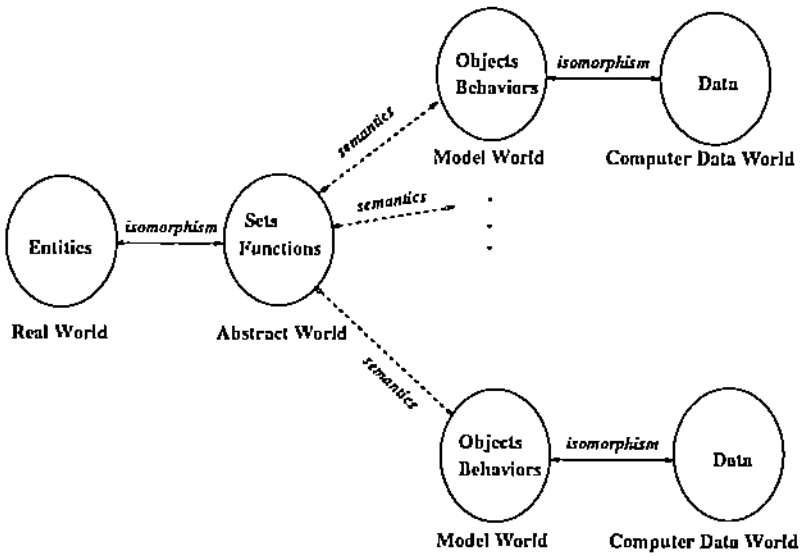

Figure 1: Graph of Worlds

In the abstract world, functions are defined on sets. This is the behavior aspect of entities. A function f is related to an entity e if e is involved in the definition domain of f as :

$$f : ... \times D \times ... \rightarrow R_1 \times R_2 \times ...$$

or e is involved in the definition range of f as :

$$f : D_1 \times D_2 \times ... \rightarrow ... \times D \times ...$$

where entity $e \in$ domain D and D_i, R_j are arbitrary sets.

In the second step, for convenient computer processing, we model the abstract world by a *model world.* There is more than one way to model the abstract world by a model world. In an object-oriented data model, the model world consists of abstract objects which can be represented in computers. Objects with the same behaviors can be grouped into classes.

We define the behavior of an object as follows: the *behavior* of an object O corresponds to a function f in the abstract world related to the entity e which is in correspondence to the object O. Intuitively, behaviors are those activities or associations in which the object participates. A behavior can be a user application, called external behavior, or it can be a method or an attribute, etc. publically accessible to users, called internal behavior. The objects of the same class share the same behaviors. Therefore behaviors are usually defined with the class. The *environment* of an object O is the set of all behaviors of O. The *context* of an object O is the subset of the environment consisting of behaviors which users are concerned with.

In the last step, the objects are represented as computer data in computers which form the *computer data world.* The representation mapping is also isomorphic. The model world and the computer data world become indistinguishable.

Therefore, from that point on, all our discussion is just on the model world and the abstract world. Database syntax is defined in the model world. The semantics is to define the mapping between the model world and the abstract world.

2.2 Database Context

Database context is the set of objects' behaviors which users are concerned with. We can divide behaviors of context into internal and external behaviors. Internal behaviors are defined by the system and are absolute for the outside world. They do not depend on any exterior elements. Publically accessible components, such as object attributes, methods, etc., all belong to the internal context. External behaviors

are applications defined by users and vary from user to user.

Without context but with the same syntax, an object can have several meanings or it can mean nothing. Context is a restriction of semantic mapping when an object has ambiguous semantics. When an object has no prior meaning, context can provide its semantics.

According to the location of the data being accessed, in multidatabases there are local and global application behaviors. Local behaviors are defined to a local database. Global behaviors are globally defined to more than one database.

A behavior which can create values of an object is called a *producer* of the object. A behavior which retrieves values of an object is called a *consumer* of the object. Usually, the producer of an object defines the semantics of the object. Consumers of the object interpret the object with the semantics defined by the producer. If there is no communication barrier between the producers and the consumers, semantic problems will not exist at all. For multidatabases, usually the consumers can not grasp the meaning defined by the producers easily.

Very often, context plays its role in a guise of other terms, such as places, databases, etc.. But contexts are still collections of behaviors.

Example 2.1 Location can determine semantics. For instance, in some databases, a value of an attribute money is 150. In the context of London, when we consider applications for London, it means 150 British pounds. In the context of New York, it means 150 U.S. dollars. The activities in which 150 is involved imply its meaning as either British pounds or U.S. dollars. To be more specific, the producers of the data value 150 determine the semantics. In London, for instance, the producer of 150 assumes that it is 150 pounds and all consumers should interpret it as 150 pounds. In computers, sometimes it is very convenient to store some rules or additional attribute items for this purpose. The rules or attributes can imply (or restrict) the meaning of some syntax structure [SM91].

Example 2.2 Database names can determine semantics. In a database DB_1, education degrees Ph.D, Master, Bachelor and High School diploma are represented as 1, 2, 3, 4, respectively. But in another database DB_2, they are represented as 4, 3, 2, 1, respectively. In the context of these different databases, the values represent different degrees. Although the context in this case is in the guise of the database names, again the semantics of the data are defined by the producer of the data. The context is actually the application behaviors.

In paper [SK92], many other instances of contexts are discussed. They all fit our definition of the context if viewed from the perspective of object behaviors.

2.3 Database Syntax and Semantics

In this section we discuss the concepts of syntax and semantics of databases formally and generically. Based on this discussion, we define semantic similarity in the next section.

1. Syntax

Syntax is the symbols and the structures in which the symbols are put together to form the model world objects in representations of entities in the real world. Computers process these symbols according to some rules and human beings interpret the meaning of the results. In database systems, schemas are used to describe the syntax of databases.

Objects with common properties and behaviors are grouped into classes. Therefore objects of the same class can be defined by their class description.

Definition 2.1 Syntax of class C is defined as

$S_C = \{$`Class Identifier, Properties, Internal Behavior Description`$\}$,

where `Internal Behavior Description` includes all the behaviors, such as attributes and methods of the class defined by the system, publically accessible to the outside

world. The external behavior of a class is the applications using the class defined outside the definition of the class.

An object is an instance of a class. Syntax of an object can be defined by the object identifier, the class it belongs to, and the instance state which retains the effect of the behaviors.

Definition 2.2 Syntax of an object O is defined as

$$S_O = \{\texttt{Object Identifier, Class Identifier, State}\},$$

Example 2.3 A class with class identifier `Employee` has properties {`name, photo, birth_date, salary`} and behaviors {`IncreaseSalary, ShowPhoto`}. An object of class `Employee` can have object identifier `E` and state {`Mark Twain, photo_image, 08/04/45, 90`}. From this syntax, the semantics of `90` of Mark's salary is not so obvious here. It can be 90 thousand dollars per year, or 90 dollars per week.

2. Semantics

Semantics is the scientific study of the relations between syntax and what it denotes[Woo85]. In our model introduced in the early part of this paper, semantics is the mapping between the conceptual world and the abstract world. For one database, data is manipulated according to some rules and users interpret the data according to some implicit assumptions. These rules and implicit assumptions are consistent within one database. When more than one database is considered, these rules and assumptions may conflict. The semantics of data has to be defined with respect to context explicitly.

Definition 2.3 Semantics of a class C in external context C^e is :

$$Sem_{C^e} : S_C \mapsto (D_C, C^e, C^i)$$

Where S_C is the syntax of class C. C^i is the internal context consisting of functions corresponding to the internal behaviors of class C. The context of class C is $C^e + C^i$. D_C is the semantic domain of class C.

Definition 2.4 Semantics of an object O in external context C^e is :

$$Sem_{C^e} : S_O \mapsto (value, D_O, C^e, C^i)$$

Where S_0 is the syntax of O. D_O is the semantic domain of O and $value \in D_O$. C^i is the same as that in the definition of class semantics.

The definition shows that the semantics of an object is related with the domain, the corresponding value in the domain, as well as external and internal behaviors. Example 2.1, Example 2.2, and the following example show how behaviors determine semantics in different situations.

Example 2.4 The number 2 may be an integer or a floating number. `2/3.0` implies 2 is the floating number `2.0`. But `list[2]` implies that 2 is an integer. Operations `/` and `[ ]` determine the semantics of 2. Without these behaviors, we can not have the exact semantics of the object 2.

3 Semantic Similarity

In this section, we define semantic similarity between objects using database semantics in application context defined in the previous section. When two objects represent the same real world entity, they will have the same semantics. When they represent different types of real world entities, they often do not share every property. Therefore, when two objects share some common property in a context, we say that they are similar, and we can integrate them according to their common behaviors.

3.1 Semantic Equivalence and Semantic Similarity

When two objects represent the same real world entity, their behaviors will be the same although the entity was modeled in different ways. In this case, the two objects are semantically equivalent. The theory of attribute equivalence, object equivalence, and class equivalence [LNE89, EN84, NSE84] is compatible with our semantic equivalence concept.

When two objects represent different types of real world entities, they can still share common attributes. Under some context, we want to integrate similar objects although they do not represent the same entity in the real world. For example, `Car` and `Truck` are two different kinds of objects. But they share some common attributes. In this case, we say they are semantically similar. We can generalize the classes of the objects and integrate them.

Some objects do not have any common attribute to share in a context. But in some extended context, they can have some properties in common. For example, if we consider `Car` and `Book` both as items we spend money on, they can be integrated as `(item name, price)`. In this case, the object similarity is very obviously context related.

3.2 Measurement of the Semantic Similarity

To know the strength of semantic similarity, we define a measurement of semantic similarity. The measurement of degrees of semantic similarity can help the decision of database integration and the optimization of global queries.

We measured semantic similarity according to the amount of the common semantic behaviors in all behaviors of the two objects. The weight function in which the frequencies are taken into account make the measurement more precise. Semantics which are often used take more weight in the measurement.

Definition 3.1 For two objects O_1 and O_2 in context C, we define the strength of semantic similarity between them as the shared behaviors in all behaviors of O_1 and O_2 with respect to some weight function $w(x)$ which reflects the frequency of the usage of behaviors :

$$sim_C(O_1, O_2) = \frac{\mid w(Sem_C(O_1) \cap f(Sem_C(O_2))) \mid}{\mid w(Sem_C(O_1)) \mid + \mid w(Sem_C(O_2)) \mid}$$

where $w(x)$ can be obtained by statistical data or in advance by the user's estimation. f is the mapping from semantic domain of object O_2 to semantic domain of object

O_1.

When $sim_C(O_1, O_2) > 0$, we say object O_1 and O_2 are *semantically similar* in context C.

Since we consider the frequency of behaviors being used, the measurement is more accurate than those described in [SK92]. The strength of semantic similarity is useful for optimization of data storage. For highly similar objects, the global view can be materialized for fast access.

Example 3.1 If objects O_1 and O_2 are semantically equivalent, all their behaviors are the same, i.e. $sim(O_1, O_2) = 1$. In this situation, the objects can be totally integrated. We will give the strict definition of semantic equivalence in the next section.

Example 3.2 If there is no common behavior among objects or their common behaviors are never used, then the similarity between them is 0. Then there is no need to integrate the objects.

Example 3.3 A car C and a truck T have common attributes {`plate_num`, `maker`, `year`}. Car C has a distinct attribute `style`, and truck T has a distinct attribute `tonnage`. The applications accessing the common behaviors and attributes are, say, 60 % in all accesses to the behaviors and attributes of the objects. Then $sim(C, T) = 0.60$.

Assumption: For simplicity in our presentation, we assume that all behaviors of an object are used by local and global users. Otherwise, if a behavior is not used by any user, it can be treated as nonexistent. If a behavior is not used by global users, it is equivalent to not being exported to global users.

3.3 A Taxonomy of Semantics Similarity

Figure 2 summarizes the taxonomy of semantic similarity which we will discuss in this section. Our work is based on [SK92]. Since our definition of semantic similarity is based on the behavior aspects of objects within the same context, the taxonomy is therefore different from that presented in [SK92] where semantic similarity was classified according to abstraction mappings and context cases.

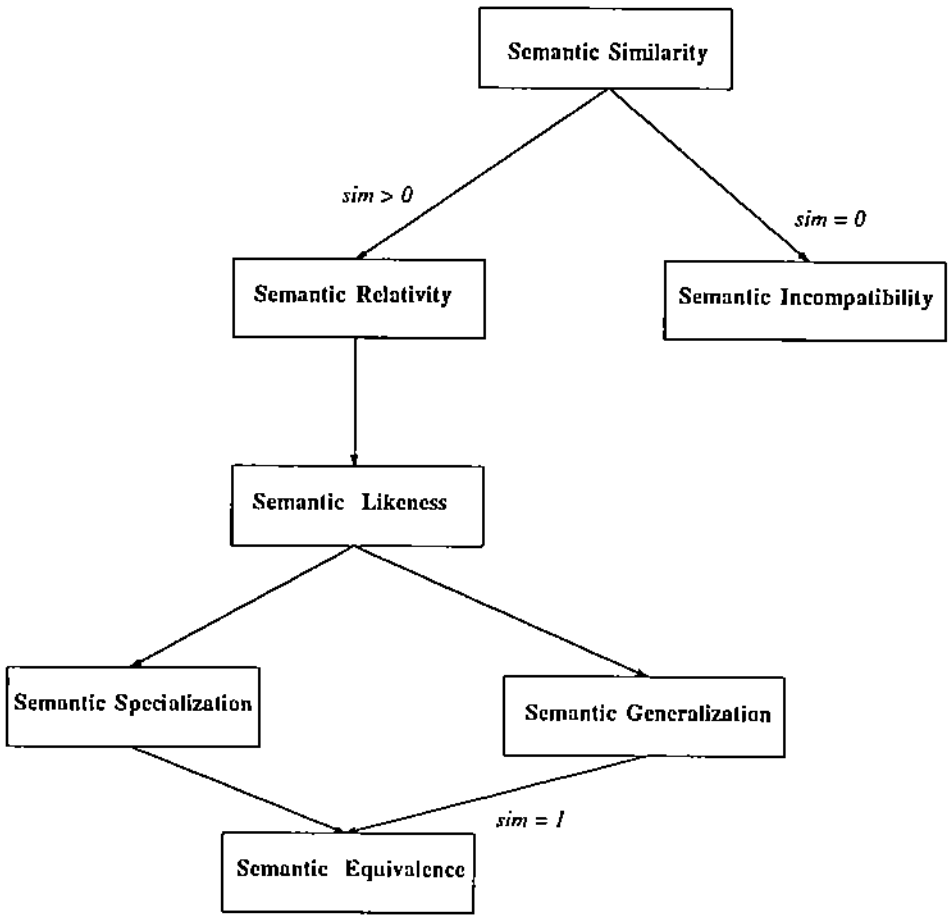

Figure 2: A Taxonomy of Semantic Similarity

When two objects are unrelated, they have *semantic incompatibility*. Otherwise they have *semantic relativity* if common behaviors exist in an extended database context. Two objects have *semantic likeness* if common behaviors exist in an extant database context. If the behaviors of one object is a subset of those of another, the semantic similarity is more specific as *semantic specialization* or *semantic general-*

ization. When two objects represent the same entity in the real world sharing the same behaviors, they have *semantic equivalence.*

1. Semantic Equivalence

Two objects have *state semantic equivalence* in a database context if they represent the same entity in the real world. Therefore, their behaviors and instance states are exactly the same. Two objects have *class semantic equivalence* if they represent entities of the same class in the real world, i.e. their behaviors are the same but the instance states may be different. By *(object) semantic equivalence*, we mean either state semantic equivalence or class semantic equivalence. Object semantic equivalence is also called object semantic *isomorphism.* A formal definition of state semantic equivalence is the following:

Definition 3.2 Object O_1 and O_2 have *state semantic equivalence* if and only if $\exists$ a bijective mapping f between the definitions of O_1 and O_2 such that $f(O_1) = O_2$, and $\exists$ a bijective mapping $\tilde{f} : Sem_1 \rightarrow Sem_2$ where Sem_j is the semantics of O_j (j=1,2) such that for context C^e

$$Sem_{C^e}(O_2) = Sem_{C^e}(f(O_1)) = \tilde{f}(Sem_{C^e}(O_1)). \quad (1)$$

When object O_1 and O_2 have state semantic equivalence, we denote it as $O_1 \overset{s}{\equiv}_{C^e} O_2$. When the external context C^e is obvious, it can be omitted. In the definition, f is the mapping between equivalent objects. $\tilde{f}$ is the mapping between semantics. Equation(1) means that the corresponding objects have the equivalent semantic behaviors. Therefore all database constraints, security control, etc. are the same for the two objects.

When f is the identity mapping in Definition 3.2, there is no schema heterogeneity between O_1 and O_2. O_1 and O_2 are identical for all contexts.

Object state semantic equivalence only exists when the objects in different databases represent exactly the same entities in the real world. If there is any inconsistency be-

tween equivalent objects, either the objects should be updated to the most recent value with consistency, or some relaxed correctness criteria should be adapted.

In reality, different objects of the same class are usually stored in different databases. In this case, we want to integrate the database schema since they represent the same class although different objects are stored in different databases. We define class semantic equivalence below.

Definition 3.3 Object O_1 and object O_2 have *class semantic equivalence* if O_1 is an instance of C_1 and O_2 is an instance of C_2, where it is possible that, $\exists$ O_3 which is an instance of C_2 (or C_1), such that O_1 (or O_2) is state semantic equivalent to O_3.

We denote class semantic equivalence of object O_1 and O_2 as $O_1 \equiv O_2$. For classes C_1 and C_2, where O_1 is an instance of C_1 and O_2 is an instance of C_2 and $O_1 \equiv O_2$, then C_1 and C_2 have the same structure and behaviors. So we can integrate them by one schema.

Example 3.4 Two synonym objects are semantically equivalent. For instance, we have

```
STUDENT : (Sno, Sname, Age)
STU_REC : (Sid, Snm, S_age).
```

Then we can say, an object (36, John, 20) in `STUDENT` is *class* semantic equivalent to (21, Mary, 18) in `STU_REC`, and (36, John, 20) in `STUDENT` is *state* semantic equivalent to (36, John, 20) in `STU_REC` since they represent the same person. State semantic equivalence maps the objects that represent the exact same real world entity to each other.

We can see from the example that semantically equivalent objects have the same inner structure and behaviors. They belong to the same class in an object-oriented data model or have the same relation schema in a relational data model. State equivalent objects are exactly the same objects which have been stored over several databases.

Theorem 3.1 Object semantic equivalence is a symmetric, transitive relation.

1. Symmetric : A_1 is equivalent to A_2, if and only if A_2 is equivalent to A_1. i.e. $A_1 \stackrel{s}{\equiv} (or \equiv) A_2 \Longleftrightarrow A_2 \stackrel{s}{\equiv} (or \equiv) A_1$

2. Transitive : If A_1 is equivalent to A_2 and A_2 is equivalent to A_3 , then A_1 is equivalent to A_3. i.e. $A_1 \stackrel{s}{\equiv} (or \equiv) A_2$ *and* $A_2 \stackrel{s}{\equiv} (or \equiv) A_3 \Longrightarrow A_1 \stackrel{s}{\equiv} (or \equiv) A_3$

The proofs are trivial from the Definition 3.2 and Definition 3.3.

By the theorem, if we have semantically equivalent classes, we can use one schema to represent all of the classes by integration since they are semantically equivalent to each other.

2. Semantic Likeness

When two objects represent different entities, they might share some common properties, therefore object semantic similarity can be defined. If the behaviors of an object are a subset of those of another object, then we say that the latter is the specialization of the former.

Definition 3.4 Object O_1 is a *state semantic specialization* of object O_2 if and only if $\exists$ a mapping f between the definitions of O_1 and O_2 such that $f(O_1) = O_2$, and $\exists$ a mapping $\tilde{f}$: $Sem_1 \rightarrow Sem_2$ where Sem_j is the semantics of O_j (j=1,2) such that for external context C^e

$$Sem_{C^e}(O_2) = Sem_{C^e}(f(O_1)) \subseteq \tilde{f}(Sem_{C^e}(O_1)) \qquad (2)$$

where f need not to be a bijection. It can be many-to-one and partial. Equation(2) shows that semantic behaviors of one object is a subset of the other's behaviors.

We denote state semantic specialization of object O_1 and O_2 as $O_1 \prec_s O_2$. We can also define class semantic specialization.

Definition 3.5 Object O_1 is a *class semantic specialization* of object O_2 if and only if for O_1 as an instance of class C_1 and O_2 as an instance of class C_2, $\exists$ O_3 in C_2 (or O_3 in C_1) , such that O_1 (or O_2) is a state semantic specialization of O_3. We denote class semantic equivalence as $O_1 \prec O_2$.

The anti-relation of semantic specialization is semantic generalization. The definition follows.

Definition 3.6 Object O_1 is a *(state) semantic generalization* of object O_2 if and only if O_2 is a *(state) semantic specialization* of O_1. We denote *(state) semantic generalization* as $(O_1 \succ_s O_2)$ $O_1 \succ O_2$.

Example 3.5 An object of `Vehicle : (Plate#, Owner)` is a semantic generalization of an object of `Car : (Plate#, Owner, Maxspeed)` . `Vehicle` is a generalization of `Car` and `Car` is a specialization of `Vehicle`.

Like semantic equivalence, semantic generalization and specialization are transitive by their definitions.

In general, the behaviors of two objects do not have inclusive relations. They only have some behaviors in common. Then we can define the concept of semantic likeness.

Definition 3.7 Object O_1 and object O_2 have *semantic likeness* if and only if $\exists$ object O_3 , such that O_3 is a semantic generalization of both O_1 and O_2. We denote it as $O_1 \sim O_2$.

Example 3.6 An object of `Truck : (Plate#, Owner, Tonnage)` has semantic likeness to an object of `Car : (Plate#, Owner, Maxspeed)` since we can have an object in `Vehicle : (Plate#, Owner)` being a semantic generalization of both a truck and a car.

Theorem 3.2 Semantic likeness is symmetric and semi-transitive.

1. Symmetric: $O_1 \sim O_2 \Longleftrightarrow O_2 \sim O_1$

2. Semi-Transitive: $O_1 \sim O_2$, *and* $O_2 \equiv O_3 \Longrightarrow O_1 \sim O_3$

Example 3.7 The semantic likeness relation is not a transitive relation. For example, let us consider pens. Pen A and Pen B both are blue in color . Pen B and Pen C both are large in size. Pen A and Pen B are similar in color. Pen B and Pen C are similar in size. But Pen A is small and blue, while Pen C is large and red, so they are not similar in the context. See Figure 3.

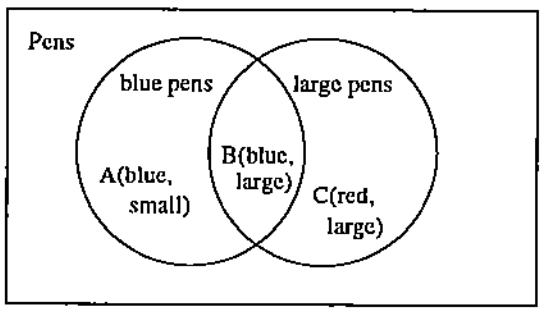

Figure 3: A $\sim$ B, B $\sim$ C, but A $\not\sim$ C

3. Semantic Discrepancy

Semantic discrepancy [She91] means there is inconsistency in the database environment. Then, either the consistency of the database should be enforced, or some weak consistency condition should be defined and maintained.

Definition 3.8 If object O_1 and object O_2 have semantic likeness, but some consistency conditions for O_1 and O_2 are violated, then the objects have *semantic discrepancy.*

Example 3.8 Two databases store employee information. For an employee, the birthday should be the same in both databases. If a birthday is not the same in the two databases, semantic discrepancy exists. This must be resolved for database integration.

4. Semantic Relativity

Sometimes, in the existing context there are no common behaviors between two objects. But when the context is extended, common behaviors may appear.

Definition 3.9 Object O_1 and object O_2 have *semantic relativity* for giving context C if and only if $\exists$ a new context C' where $C \subseteq C'$, and O_1 and O_2 have semantic likeness in C'.

Example 3.9 Classes `Car` and `Book` are unrelated in an environment. When we extend the context, there is an application to access prices of a car or a book. Then `Car` behaves the same as `Book`. They are relative in the new context. We can create a new schema

```
Goods :  (item name, price).
```

for the new application and context.

5. Semantic Incompatibility

It is impossible to integrate two semantically incompatible objects. Moreover, there is no need to integrate them. They do not share any similar properties in the applications users are concerned with.

Definition 3.10 Two objects O_1, O_2 have *semantic incompatibility* if they are not semantically relative, i.e., there is no context which users are concerned with in which the two objects have a semantic relativity.

Example 3.10 Classes `Person` and `Steel` have no common behavior in the context. No applications would instill them with common properties, i.e. no applications require extension of the database context. Then these objcets are semantically incompatible.

4 Conclusions and Future Work

In this paper, we precisely give a very general definition of database context and semantics in respect to context of database applications. We present a way to define and classify object semantic similarity.

Our work is an extension of work by Sheth et al [SK92]. In comparison, our main contributions are:

1. We investigate the nature of database context which was mentioned in [SK92] as an open problem. We define it as object's behaviors which we are concerned with. This definition encloses all the context cases mentioned in [SK92].

2. Based on the context concept, we define semantic similarity. Semantic equivalence is a special case of semantic similarity.

3. We proposed a measurement of semantic similarity. It is more accurate since the frequency of behavior usage is taken into account.

4. We present a taxonomy of the semantic similarity. For each case, we describe the integration to be made.

Future problems remaining to be solved are : 1) To represent semantics and semantic similarity as rules or additional attributes in databases. 2) To build an expert system to aid the evaluation of semantic similarity.

Acknowledgments

The authors would like to thank Prof. Ahmed Elmagarmid, the head of the InterBase project, for his valuable discussion and comments regarding this paper. We are very grateful for his encouragement and guidance of this research. The authors are also very grateful to Ms. Evaggelia Pitoura for many helpful comments.

References

[ACM90] Guest Editors' Introduction to the Special Issue on Heterogenous Databases edited by A. Elmagarmid and C. Pu. volume 22. September 1990.

[ANRS92] M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth. Using Flexible Transactions to Support Multi-System Telecommunication Applications. In *Proceedings of the 18th VLDB conference*, Vancouver, Canada, August 1992.

[BCD+93] Omran A. Bukhres, Jiansan Chen, Weimin Du, Ahmed K. Elmagarmid, and Robert Pezzoli. InterBase : An Execution Environment for Global Applications over Distributed, Heterogeneous, and Autonomous Software Systems. *IEEE Computer*, August 1993. (to appear).

[BGMS92] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *The VLDB Journal*, 1(2):181–239, October 1992.

[BLN86] C. Batini, M. Lenzerini, and S.B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):232–364, December 1986.

[BOT86] Y. Breitbart, P.L. Olson, and G.R. Thompson. Database Integration in a Distributed Heterogeneous Database Systems. In *Proceedings of the 2nd International Conference on Data Engineering*, pages 301–310, Los Angeles, CA, February 1986.

[DE89] W. Du and A. Elmagarmid. Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 347–355, Amsterdam, The Netherlands, August 1989.

[DH84] U. Dayal and H.Y. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Tran. on Software Engineering*, 10(6):628–644, 1984.

[EN84] R. Elmasri and S. B. Navathe. Object integration in database design. *Proc. IEEE COMPDEC Conf.*, March 1984.

[FKN91] P. Fankhauser, M. Kracker, and E. Neuhold. Semantic vs. Structural Resemblance of Classes. *Semantic Issues in Multidatabase: SIGMOD Record Special*, 20(4), December 1991.

[GPN91] J. Geller, Y. Perl, and E. J. Neuhold. Structure and Semantics in OODB Class Specifications. *SIGMOD Record*, 20:40–43, December 1991.

[Jac74] Nathan Jacobson. *Basic Algebra I.* W. H. Freeman and Company, 1974.

[Ken91] William Kent. The Breakdown of the Information Model in Multi-Database Systems . *SIGMOD Record*, 20(4):10–15, December 1991.

[KS91] Won Kim and Jungyun Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *Computer*, 24(12):12–18, December 1991.

[LNE89] J. A. Larson, S. B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transaction on Software Engineering*, 15(4):449–463, April 1989.

[LR82] T. Landers and R. Rosenberg. An Overview of Multibase. In H. Schneider, editor, *Distributed Data Systems.* North-Holland, 1982.

[Mey90] B. Meyer. *Introduction to the Theory of Programming Language.* Prentice Hall, 1990.

[NSE84] S. Navathe, S. Sashidhar, and T. Elmasri. Relationship merging in schema integration. In *Proceedings of 10th VLDB*, August 1984.

[ÖV91] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Inc., 1991.

[She91] A. Sheth. Semantic Issues in Multidatabase Systems - Preface by the Special Issue Editor. *SIGMOD Record*, December 1991.

[SK92] A. Sheth and V. Kashyap. So Far (Schematically) yet So Close (Semantically). In *Proceedings of the Intl. Conf. on Semantics of Interoperable Database Systems*, November 1992.

[SL90] Amit P. Sheth and James A. Larson. Federated databases systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, pages 183–236, September 1990.

[SM91] M. Siegel and S. Madnick. A Metadata Approach to Resolving Semantic Conflicts. In *Procedings of the 17th International Conference on Very Large Data Bases*, September 1991.

[TLW87] M. Templeton, E. Lund, and P. Ward. Pragmatics of Access Control in Mermaid. *IEEE Data Engineering Bulletin*, 10(3):33–38, 1987.

[Woo85] J. Wood. What's in a Link. *Readings in Knowledge Representation*, pages 217–241, 1985.