A Theory of Attribute Equivalence in Databases with Application to Schema Integration

JAMES A. LARSON, MEMBER, IEEE, SHAMKANT B. NAVATHE, AND RAMEZ ELMASRI, MEMBER, IEEE

Abstract—Schema integration is necessary in designing a logical database schema or a conceptual schema from multiple views. Schema integration is also necessary to define a global schema that describes all of the data in several existing databases participating in a distributed database management system.

This paper unifies our previous work in schema integration. That work described how to integrate groups of entity sets and groups of relationship sets from different schemas specified using the entity-relationship data model. We present a common foundation for integrating pairs of entity sets, pairs of relationship sets, and an entity set with a relationship set. This common foundation is based on the basic principle of integrating attributes. Any pair of objects whose identifying attributes can be integrated can themselves be integrated.

Several definitions of attribute equivalence are presented. These definitions can be used to specify the exact nature of the relationship between a pair of attributes. Based on these definitions, several strategies for attribute integration are presented and evaluated.

Index Terms—Attributes, database design, database integration, data equivalence, schema integration.

I. INTRODUCTION

Schema integration is becoming increasingly important because it is needed in the increasingly prevalent contexts of logical database design and global schema design. In logical database design, each class of users designs a view of part of a proposed database that they need to access. The objective is to design a conceptual schema (or logical integrated schema) that represents the contents of all of these views. User queries and transactions specified against each view are mapped to the logical integrated schema. In global schema design, several databases already exist and are in use. The objective is to design a single global schema that represents the contents of all of these databases. This global schema can then be used as an interface to the diverse databases. User queries and transactions specified against the global schema are mapped to the existing schemas supported by the relevant databases.

Fig. 1 illustrates the four major tasks performed during integration:

1) Schema Translation: Converts schemas to a canonical data model. Intraschema transformations are applied to the schemas to be integrated, converting them into an easy-to-integrate form.

2) Assertion Specification: Specifies assertions among objects in two or more schemas. Assertions are specified among the attributes, object classes, and relationships of the schemas to be integrated.

3) Schema Integration: Object classes in two or more schemas are integrated based on the specified assertions.

4) Mapping Generation: Mappings between schemas are generated for use in translating database requests within either of two contexts: 1) from user views to the integrated logical schema (for logical database design), or 2) from the global schema to the underlying database schemas (for global schema designs).

These four tasks are based on an extension of the methodology for view integration in [1], one of several approaches to schema integration compared in [2]. We have found that many of the conflicts that arise in schema integration, such as conflicts in naming, scale, structure, and abstraction [3], can be phrased in terms of equivalence of attributes, equivalence of object classes [4], and equivalence of relationships [5].

This paper concentrates on the equivalence of attributes and how these equivalences can be exploited. In particular, we show how relationships and entity sets can be integrated if their identifying attributes can be integrated. The discussion in our paper therefore revolves around the tasks performed during assertion specification (task 2) and schema integration (task 3).

This paper is structured as follows. Section II describes the data model we use as a basis for discussion. Section III defines several notions of attribute equivalence. Section IV extends our previous work on object integration and relationship set integration by defining object equivalence and relationship set equivalence in terms of attribute equivalence. Section V describes and evaluates several strategies for integrating attributes. Section VI presents our conclusions and future work.

The four main contributions of this paper are: 1) refinement of our previous concept of object equivalence and relationship equivalence, 2) formal definitions of types of attribute equivalence, 3) a formulation of different strategies for attribute integration, and 4) an application of attribute integration to both object integration and relationship integration.

Manuscript received May 30, 1986; revised November 8, 1988.

J. A. Larson is with Honeywell Corporate Systems Development Division, 1000 Boone Avenue North, Minneapolis, MN 55427.
S. B. Navathe is with the Department of Computer and Information Sciences, University of Florida, Gainesville, FL 32611.
R. Elmasri is with the Department of Computer Science, University of Houston—University Park, Houston, TX 77004.

IEEE Log Number 8826220.
II. THE ECR MODEL OF DATA

To explain the concepts involved in schema integration, we need to use a data model. The conceptual data model we use in our work [4], [6], [7] is the entity-category-relationship (ECR) model [8], which is an extension of the entity-relationship (ER) model [9]. However, our work may be easily adapted to other data models. The motivation for using the ECR model is threefold: 1) The ER model is a popular tool for logical database design [2], [10]. An extension to the ER model is needed in our work because of the need to represent subclasses. This extension is provided by a single new concept, called category, in the ECR model. 2) The ECR model is rigorously defined in [8] and has a formal, well-defined query language (GORDAS [11]). 3) Mappings between the three popular models used in database systems (network, relational, hierarchical) and the ECR model are straightforward [8].

The ECR data model extends the ER model by using the category concept to represent subclasses [12], which will play a critical role during schema integration. (In the ECR data model [8], a category can also be used to represent a subset of the union of object classes. This use of categories is not required in the present paper.) In addition, the ECR data model also extends the ER model by specifying cardinality and dependency constraints on relationships that are useful in capturing data semantics. These constraints are analyzed during schema integration. The ECR model uses the constructs of entity set, relationship set, category, and attribute. The term object class refers to either an entity set or a category. An entity set represents a set of entities that have the same attributes. A category can be used to model a subset of an object class. A category inherits the attributes of the object class of which it is a subset [13], [14]. An example of an ECR schema is shown in Fig. 2: the diagrammatic notation is shown in Fig. 2: the diagrammatic notation is an extension of the ER diagram [9]. Rectangular boxes represent entity sets, hexagonal boxes represent categories, and diamond-shaped boxes represent relationship sets.

Constraints on the number of instances of a relationship in which an entity may participate are represented by a pair of integers (min, max); each entity must participate
in at least min and at most max relationship instances. In Fig. 2, each vehicle instance must participate in at least one and at most n instances of the ownership relationship.

Our strategy for schema integration [4]-[7] using the ECR data model is to integrate object classes (entity sets and categories) first and then integrate relationships. In our methodology, equivalence specification plays an important part in both of these steps. In the first step, object class integration is accomplished by determining the equivalence of identifying attributes of the respective object classes. This is followed by relationship integration. This paper unifies our previous work by presenting a common foundation for integrating two or more relationship sets. This common foundation is based on the basic principle of integrating attributes.

III. Attribute Equivalence

Mannino and Effelsberg [15] have classified attribute equivalence into local and global based on the properties of semantic equivalence and scope in the context of a domain conversion function. Whereas some of their ingredients of equivalence appear similar to our list of attribute characteristics and their concept of domain conversion is similar to our concept of mapping function, we have attempted a much more detailed treatment of equivalence.

To determine an equivalence among attributes, various characteristics of the attributes must first be established. Subsection III-A discusses the various characteristics of attributes that will have a bearing on attribute equivalence. Attributes that are equivalent have several characteristics in common. We refer to these common characteristics as basic equivalence properties and define them in Subsection III-B. Based on these basic equivalence properties, various types of equivalence among attributes can be defined. Subsection III-C defines the notion of STRONG equivalences among attributes. STRONG equivalences imply that both updates and retrievals can be supported. Subsection III-D defines the notion of WEAK equivalences among attributes. WEAK equivalences imply that retrievals can be supported but not necessarily updates. Subsection III-E defines the notion of DISJOINT equivalences, which may apply to attributes that are neither STRONG nor WEAK equivalent but can be integrated anyway.

A. Characteristics of Attributes

To determine an equivalence among attributes, various characteristics of the attributes must be established. These characteristics describe the attribute with respect to the object class to which it belongs. The correspondence of the characteristics of two attributes provides a basis to establish the equivalence of the attributes.

Definition 1: The characteristics of attribute a of object class C are as follows.

Uniqueness: A constraint that specifies that two objects belonging to the object class cannot have the same values for a set of attributes. Let UN(a) be the set of attributes, including a, that together uniquely identifies an instance of the object class.

Cardinality: A constraint on the number of values of attribute a that can be present in each instance of object class C. The cardinality will be expressed as upper and lower bounds. A lower bound of zero indicates that the attribute may have no values, i.e., be null. An upper bound greater than one indicates an attribute with multiple values. Integers UB(a) and LB(a) denote the upper and lower bounds on the number of values that a may take for each instance of object class C.

Domain: A set of values the attribute a may take for individual instances of object class C. DOM(a) denotes the domain of attribute a.

Static Semantic Integrity Constraints: Constraints involving the attribute and possibly other attributes from the same or different object classes. Functional dependencies, multivalued dependencies, and existence dependencies such as the referential integrity constraint are examples of semantic integrity constraint expressions involving attribute a.

Dynamic Semantic Integrity Constraints: These describe constraints on changes to which attribute values must adhere. For example, OLD.salary less-than-or-equal NEW.salary is a state-change constraint on the salary attribute. SCC(a) denotes the set of state-change constraint expressions involving attribute a.

Security Constraints: These describe restrictions on the use of values of an attribute. For example, user Jones can read but not modify the salary attribute value. SEC(a) denotes the set of the security constraint expressions involving attribute a.

Allowable Operations: This is a collection of allowable operations associated with the domain of each attribute. As examples, scalar values such as weight may be compared and added, while dates may only be compared. OP(a) is the set of allowable operations on the domain of attribute a.

Scale: This specifies the interpretation of values of attributes. For numeric attributes, scale specifies the unit of measurement. As examples, the height attribute may be measured in meters, and the speed attribute may be measured in kilometers per hour. For nonnumeric attributes, scale may specify the language or code used to express the values. For example, the code value of 13 or the character string “verde” can be interpreted as the color green. SCALE(a) denotes the unit of measurement of attribute a. In our discussion on equivalence of attributes a and b, we make use of a mapping function f which relates the scales of DOM(a) and DOM(b). The function f enables us to hide any differences in scale.

To illustrate these characteristics, Example 1 describes the characteristics for attributes in two different object classes.

Example 1: EMPLOYEE is an object class with the three attributes social-security-number, height-in-inches and degrees. Social-security-number is a unique identifier of EMPLOYEE. Social-security-number functionally determines height-in-inches, and also functionally deter-
mines degrees. A semantic integrity constraint requires that values of height-in-inches can only be increased. There is only one allowable computational operation, average, which is an aggregate function defined over values of height-in-inches. A security restriction requires that only the personnel office may update the degrees values. The first three columns of Table I summarize the characteristics of these attributes.

Example 2: Like EMPLOYEE, PERSON is an object class with the three attributes employee-number, height-in-centimeters, and education. The attribute employee-number is a unique identifier. Employee-number functionally determines height-in-centimeters; it also functionally determines education. A semantic constraint restriction requires that height-in-centimeters can only be increased in value. Average, an example of an aggregate function, is defined over the values of height-in-centimeters. A security restriction requires that only the personnel officer can change values of the education attribute. The last three columns of Table I summarize the characteristics of the three attributes.

These characteristics of attributes, and others that the database administrator feels are important, are used to determine the equivalence of two attributes. We will need the notation introduced in Definition 2 in the following section.

Definition 2: The values of an attribute $a$ of object class $C$, denoted by VALUES$(a)$, is a subset of DOM$(a)$. It equals the set of $a$ values of all occurrences of $C$ at some point in time.

B. Basic Attribute Equivalence Properties

In the next three subsections, we will define three types of attribute equivalence:-strong attribute equivalence, weak attribute equivalence, and disjoint attribute equivalence. These types of attribute equivalence have several properties in common, which we refer to as basic equivalence properties. The difference between the several types of attribute equivalence is the manner in which the mapping function, defined below, is applied.

Definition 3 (Basic Equivalence Properties): Let $a_i$ be an attribute of object class $A$ and $b_i$ be an attribute of object class $B$. Let $D_i$ be the largest non-null subset of DOM$(a_i)$ and $R_i$ be the largest non-null subset of DOM$(b_i)$ such that there exists a mapping function $f_i: D_i \rightarrow R_i$ and its inverse, $f^{-1}_i: R_i \rightarrow D_i$. The functions $f_i$ are defined by the database administrator and are used in the retaining conditions that define the basic equivalence properties of $a_i$ and $b_i$. The properties of $f_i$ are as follows.

1) The function $f_i$ is an isomorphism; there exists a one to one correspondence between $D_i$ and $R_i$. (This is a restatement of the fact that $f_i$ has an inverse.)

2) Each allowable operation on $a_i$ has an equivalent allowable operation on $b_i$, and vice versa. For every allowable operation $g$ in OP$(a_i)$, there exists an allowable operation $g'$ in OP$(b_i)$ such that $f_i \circ g = g' \circ f_i$; that is, the result of applying operation $g$ on some value $x$ in DOM$(a_i)$ followed by the mapping function $f_i$ gives the same result as applying the mapping function $f_i$ on $x$ and then applying the operation $g'$ on $f_i(x)$. For example, if
g is unary, then for every \( x \) in \( D_i, f_i(g(x)) = g'(f_i(x)) \).
If \( g \) is binary, then for every \( a' \) and \( a'' \) in \( D_i, f_i(g(a', a'')) = g'(f_i(a'), f_i(a'')) \), and so on.

3) All semantic integrity constraints hold under the \( f_i \)
and \( f_i^* \) mappings. Let \( f_1, f_2, \cdots, f_n \) be the mapping
functions associated with attributes \( a_1, a_2, \cdots, a_n \). Each
constraint in \( \text{SIC}(a_i) \) involving attributes \( a_1, a_2, \cdots, a_n \)
should be implied by \( \text{SIC}(b_i) \) after replacing \( a_i \) by \( f_i(a_i) \).
Similarly, each constraint in \( \text{SIC}(b_i) \) involving attributes
\( b_1, b_2, \cdots, b_m \) should be implied by \( \text{SIC}(a_i) \) after replacing
\( b_i \) by \( f_i^*(b_i) \) for \( i = 1, \cdots, m \).

In short, \( \text{SIC}(a_i) \Rightarrow \text{SIC}(b_i) \) under the mapping functions
\( f_1, f_2, \cdots, f_n \).

4) All state change constraints hold under the \( f_i \)
and \( f_i^* \) mappings. \( \text{SCC}(a_i) \Rightarrow \text{SCC}(b_i) \).

5) All security constraints hold under the \( f_i \) and \( f_i^* \)
mappings. \( \text{SEC}(a_i) \) implies and is implied by \( \text{SEC}(b_i) \).

6) The mapping functions preserve functional dependences.
(This property is a special case of property 3, but
is listed here separately because functional dependences are
often the only semantic integrity constraints explicit in
database schemas.) Consider attributes \( a_i \) and \( a_j \) to be
attributes of object class \( A \) such that \( a_j \) is functionally depend-
on on \( a_i \). Thus there exists a function \( f_{aij}: \text{DOM}(a_i) \rightarrow \text{DOM}(a_j) \).
Consider attributes \( b_i \) and \( b_j \) to be attributes of
object class \( B \) such that \( b_j \) is functionally dependent on
\( b_i \). Thus there exists a function \( f_{bij}: \text{DOM}(b_i) \rightarrow \text{DOM}(b_j) \)
such that for each value \( a_i \) of \( a_i \), \( f_i(f_{aij}(a_i)) = f_{bij}(f_i(a_i)) \).

7) The mapping functions preserve unique identifiers.

The database administrator must determine all of the
mapping functions such that the invariants, state-change
constraints, security constraints, and allowable operations also
hold under the mapping.

Example 3: Given the attributes of EMPLOYEE and
PERSON in Examples 1 and 2, and the functions between
social-security-number and employee-number (\( f_1 \)),
between height-in-inches and height-in-centimeters (\( f_2 \)),
and between degrees and education (\( f_3 \)). \( D_i \) stands for
the domain and \( R_i \) stands for the range of the functions:

\[
\begin{align*}
f_1 &: D_1 = \text{DOM}(\text{social-security-number}); \\
    & R_1 = \text{DOM}(\text{employee-number}); \\
    & f_1(111-11-1111) = 1, \\
    & f_1(222-22-2222) = 2, \\
    & f_1(333-33-3333) = 3, \\
    & f_1(444-44-4444) = 4, \\
    & f_1(555-55-5555) = 5. \\
f_2 &: D_2 = \text{DOM}(\text{height-in-inches}); \\
    & R_2 = \text{DOM}(\text{height-in-centimeters}); \\
    & f_2(x) = 2.54 \times x \\
f_3 &: D_3 = \text{DOM}(\text{degrees}) \text{ MINUS } \{1\} \\
    & R_3 = \text{DOM}(\text{education}) \text{ MINUS } \{\text{MD}\} \\
    & f_3(1) = \text{not defined}, \\
    & f_3(2) = \text{BS}, \\
    & f_3(3) = \text{MS}, \\
    & f_3(4) = \text{PhD}. 
\end{align*}
\]

(Reason: the code 1, which represents a high school,
does not exist in \( \text{DOM}(\text{education}) \);
Reason: the education attribute does not have a
code corresponding to \( \text{MD} \);

Conclusion: These mappings are isomorphic. The al-
lowable aggregate operation, average, is equivalent on
both height-in-inches and height-in-centimeters. The sec-
urity constraints about updating degrees and education
are equivalent. Both functional dependencies are pre-
served by the mappings. The unique identifiers are equiv-
lent under the \( f_i \) mapping. Therefore, the mapping func-
tions \( f_1, f_2, \cdots, f_n \) satisfy all of the basic equivalence
properties.

Theorem 1 gives the conditions under which basic
equivalence property \( 1 \) on a mapping function \( f_2 \) implies
the remaining equivalence properties.

Theorem 1: Given the following:

1) \( f_i \), a mapping function from \( D_i \) contained in
\( \text{DOM}(a_i) \) to \( R_i \) contained in \( \text{DOM}(b_i) \) that satisfies
the basic equivalence properties,
2) \( f_{aij} \), a functional dependency from attribute \( a_i \) to at-
ttribute \( a_j \), and
3) \( f_{bij} \), a functional dependency from attribute \( b_i \) to \( b_j \), and
4) there exists a mapping, say \( f_j \), from \( D_i \) contained in
\( \text{DOM}(a_i) \) to \( R_j \) contained in \( \text{DOM}(b_i) \) that satisfies
the basic equivalence property \( 1 \),

then the mapping function \( f_j \) also satisfies all of the basic
equivalence properties (proof of this theorem is found in
[16]).

We next define three types of attribute equivalence,
STRONG, WEAK, and DISJOINT. If two attributes \( a \) and \( b \) are either STRONGLY or WEAKLY equivalent,
then they may be integrated to form attribute \( c \). If \( a \) and \( b \)
are STRONGLY equivalent, then values of \( c \) may be
updated. Values of \( c \) may not be updated if \( a \) is WEAKLY
equivalent but not STRONGLY equivalent to \( b \). Thus,
the basic difference between STRONG and WEAK equiva-
ience is the ability to update. We also define another type
of attribute equivalence, DISJOINT equivalence, that can
be used when neither STRONG nor WEAK equivalence
holds.

C. Strong Attribute Equivalence

Let \( a \) be a (possibly multivalued) attribute of object
class \( A \) and \( b \) be a (possibly multivalued) attribute of
object class \( B \). We define several classes of STRONG rela-
tionships between a and b depending upon the context of object classes A and B.

**STRONG α equivalences** relate the values of one (possibly multivalued) attribute in one instance of an entity set with the values of another (possibly multivalued) attribute in one instance of another entity set at some point in time. **STRONG α equivalences** depend upon D, the largest non-null subset of current values, and upon R, the largest non-null subset of values, on which the mapping f and its inverse are defined. There are four types of **STRONG α equivalences**, depending upon how many values of attribute a and values of attribute b can be related by the mapping function.

Informally, a STRONG α EQUAL b implies that there is a one-to-one correspondence between the set of all values of a and the set of all of values of b. a STRONG α CONTAINS b implies that there is a one-to-one correspondence between a subset of the values of a and all of the values of b. a STRONG α CONTAINED-IN b implies that there is a one-to-one correspondence between all of the values of a and a subset of the values of b. a STRONG α OVERLAPS b implies that there is a one-to-one correspondence between a subset of the values of a and a subset of the values of b. A more formal definition follows.

**Definition 4 (STRONG α Equivalences):** Given an attribute a of object class A and an attribute b of object class B at some point in time, and f: D → R:

1) if a and b obey the basic equivalence properties of definition 3, D = VALUES (a), and R = VALUES (b), then a STRONG α EQUAL b.

2) if a and b obey the basic equivalence properties, D = VALUES (a) and R is properly contained in VALUES (b), then a STRONG α CONTAINS b.

3) if a and b obey the basic equivalence properties, D is properly contained in VALUES (a), and R = VALUES (b), then a STRONG α CONTAINED-IN b.

4) if a and b obey the basic equivalence properties, D is properly contained in VALUES (a), and R is properly contained in VALUES (b), then a STRONG α OVERLAPS b.

**Example 4:** The following table gives values of the degrees and education attributes for the instances of PERSON and EMPLOYEE objects from Example 3:

<table>
<thead>
<tr>
<th>Point</th>
<th>Values of Degrees</th>
<th>Values of Education</th>
<th>Type of Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Time</td>
<td>BS, MS</td>
<td>{2, 3}</td>
<td>STRONG α EQUAL</td>
</tr>
<tr>
<td>BS, MS, MD</td>
<td>{2, 3}</td>
<td>STRONG α CONTAINS</td>
<td></td>
</tr>
<tr>
<td>BS, MS</td>
<td>{1, 2, 3}</td>
<td>STRONG α CONTAINED-IN</td>
<td></td>
</tr>
<tr>
<td>BS, MS, MD</td>
<td>{1, 2, 3}</td>
<td>STRONG α OVERLAPS</td>
<td></td>
</tr>
<tr>
<td>MD</td>
<td>{1}</td>
<td>no STRONG α equivalence</td>
<td></td>
</tr>
</tbody>
</table>

There is another type of equivalence called **STRONG β equivalence** that is defined in terms of **STRONG α equivalence**. **STRONG β equivalences** relates the values of one (possibly multivalued) attribute of an entity set with the values of a (possibly multivalued) attribute of another entity set over all points in time. By all points in time we mean all states of the database. Just as there are four types of **STRONG α equivalences**, there are four types of **STRONG β equivalences**:

**Definition 5 (STRONG β Equivalences):** Given an attribute a of object class A and an attribute b of object class B:

1) if a STRONGLY α EQUAL b holds for every point in time for which both A and B exist, then a STRONGLY β EQUAL b.

2) if for every point in time, either a STRONG α EQUAL b, or a STRONG α CONTAINS b holds, then a STRONG β CONTAINS b.

3) if for every point in time, either a STRONG α EQUAL b, or a STRONG α CONTAINED-IN b holds, then a STRONG β CONTAINED-IN b.

4) if a STRONG β EQUAL b, a STRONG β CONTAINS b, or a STRONG β CONTAINED-IN b hold at different time instances, then a STRONG β OVERLAP b.

**Example 5:** Fig. 3 illustrates the characteristics of six attributes. Attributes CR1–CR5 contain class-rank values using two coding schemes: numeric (1, 2, ...), and string (Freshman, Sophomore, ...). The mapping function relating these two coding schemes is summarized in Fig. 4. Using this f mapping function, the **STRONG β equivalence** types are summarized in Fig. 5. (Typically, the equivalence would be determined by the DBA after consulting the users and database designers. We shall assume this throughout.) Because there is no mapping between the values of the sex-code attribute and the CR attributes, the sex-code attribute is DISJOINT from each of the CR attributes.

**STRONG β equivalence** permits users to perform updates on values in D and R. This is possible because the mapping function defines the correspondence between values of D and R. The function can convert any value in D into the corresponding value in R. Because the mapping has an inverse, any value in the range R of the mapping function can be converted into the corresponding value in the domain D. If the integrated attribute is expressed as values of D, then the values in D can be converted to values of R. If the integrated attribute is expressed as values of R, values in R can be converted to values of D.

When inserting an object A that contains attribute a, the user may assign values from D to be values for a. If the attribute a is physically represented using values in D, then these values may be directly stored into the database. If the attribute a is physically represented using values in R, then the values of a can be converted to equivalent values in R and then stored in the database. However, the user may not assign values from DOM (a) – D to attribute a if the attribute is physically represented using values in R because there is no mapping function from DOM (a) – D to R. Similarly, the user may not assign values from DOM (b) – R to attribute b.
Fig. 3. Some characteristics of six attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Unique</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
<th>Domain</th>
<th>Invariants</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>no</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CR2</td>
<td>no</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CR3</td>
<td>no</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CR4</td>
<td>no</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CR5</td>
<td>no</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sex-code</td>
<td>no</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 4. Correspondences between coding schemas.

Fig. 5. Strong \( \beta \) equivalence types among the five attributes of Fig. 4 (e.g., CR1 \( \beta \) contained in CR3.)

Full update capability is possible if a \( \text{STRONG} \ \beta \) \text{EQUAL} \( b \) because the \( f \) mapping function and its inverse are both total with respect to \( \text{DOM} (a) \) and \( \text{DOM} (b) \). When \( \text{DOM} (a) = D \) or \( \text{DOM} (b) = R \) is not null, that is, the domain and range of the mapping function fall short of the possible values that attributes \( a \) and \( b \) may take, the full update capability may not exist. The above situation occurs when a \( \text{STRONG} \ \beta \ \text{CONTAINED-IN} \) or \( a \) \( \text{STRONG} \ \beta \ \text{CONTAINS} \) \( b \), respectively. If the containing domain (\( \text{DOM} (a) \)) if a \( \text{STRONG} \ \beta \ \text{CONTAINS} \) \( b \), \( \text{DOM} (b) \) if a \( \text{STRONG} \ \beta \ \text{CONTAINED-IN} \) \( a \) exists as the physical representation of the integrated attribute, then full update capability is possible. In the case of \( a \) \( \text{STRONG} \ \beta \ \text{OVERLAP} \) \( b \), full update is not possible.

In the above examples we illustrated two categories of attribute equivalences: \( \text{STRONG} \ \alpha \) and \( \text{STRONG} \ \beta \). Under each, five cases occur. The word \( \text{STRONG} \) is used to indicate that updates are possible. \( \alpha \) is used if the equivalence holds for some point in time whereas \( \beta \) is used if the equivalence holds for all points in time.

D. Weak Attribute Equivalence

\( \text{STRONG} \) equivalence implies that updates are possible against the integrated schema. A weaker form of equivalence can be defined if only retrieval is permitted. Using an alternative for Definition 3 (basic attribute equivalence) as given below, we can apply Definitions 4 and 5 to define the weak equivalences (by replacing \( \text{STRONG} \) with \( \text{WEAK} \) in definitions 4 and 5).

Definition 6 (Alternative Basic Attribute Equivalence): Attributes \( a \) and \( b \) are \( \text{WEAK} \) equivalent if all conditions of \( \text{STRONG} \) equivalence hold with the following exceptions: a) no inverse function need exist, and b) the basic equivalence properties 3, 4, and 5 of Definition 3 are changed as follows.

Let \( f_1, f_2, \cdots, f_n \) be the mapping functions associated with attributes \( a_1, a_2, \cdots, a_n \). Each constraint in \( \text{SIC} (a) \) involving attributes \( a_1, a_2, \cdots, a_n \) should hold in \( \text{SIC} (b) \) after replacing \( a_i \) by \( f_i (a) \), \( i = 1, \cdots, n \). Likewise, each constraint in \( \text{SCC} (a) \) and \( \text{SEC} (a) \) hold in \( \text{SCC} (b) \) and \( \text{SEC} (b) \).

The \( \text{WEAK} \) equivalence relationship is not reflexive because the function \( f \) need not have a complete inverse.

Example 6: Given \( \text{DOM} (\text{CR} 3) = \{ \text{Freshman}, \text{Sophomore}, \text{Jr}, \text{Sr}, \text{MS}, \text{PhD} \} \) and \( \text{DOM} (\text{CR} 6) = \{ \text{undergrad}, \text{grad} \} \), the function \( f \) where \( f(\text{Freshman}) = f(\text{Sophomore}) = f(\text{Jr}) = f(\text{Sr}) = \text{undergrad} \) and \( f(\text{MS}) \)
Example 7: Suppose \( \text{DOM}(a) = \{ \text{Freshman}, \text{Sophomore}, \text{Jr}, \text{Sr}, \text{MS}, \text{PhD} \} \), \( \text{DOM}(b) = \{ 1, 2, 3, 4, 5 \} \) and mapping function \( f: \text{DOM}(a) \rightarrow \text{DOM}(b) \) defined by: \( f(\text{Freshman}) = 1, f(\text{Sophomore}) = 2, f(\text{Jr}) = 3, f(\text{Sr}) = 4, f(\text{MS}) = 5, f(\text{PhD}) = 5 \).

Because \( f \) maps all values of \( \text{DOM}(a) \) into all values of \( \text{DOM}(b) \), we have a WEAK \( \beta \) EQUAL \( b \). However a STRONG \( \beta \) CONTAINS \( b \) does not hold because there is no inverse of this mapping.

Another mapping function defined by: \( f_1(\text{Freshman}) = 1, f_1(\text{Sophomore}) = 2, f_1(\text{Jr}) = 3, f_1(\text{Sr}) = 4, f_1(\text{MS}) = 5 \); maps a subset of \( \text{DOM}(a) \) into all of the values of \( \text{DOM}(a) \). This mapping function does have an inverse; thus a STRONG \( \beta \) CONTAINS \( b \).

Using function \( f \) in Example 7, which produces WEAK equivalence, both MS and PhD are mapped to 5; thus it is possible to retrieve \( a \) with MS and PhD values, but these values cannot be distinguished because both are mapped to the same value 5. If a value of the integrated attribute is changed from 4 to 5, it is not possible to determine if \( a \) has a value of MS or PhD; so updates are not allowed.

On the other hand, using function \( f_1 \), which yields STRONG equivalence, updates are allowed. For example, if a value of \( c \) changed from 4 to 5, then \( a \) is assigned a value of MS. (It is not possible to assign a value of PhD because it is not included in the \( f \) mapping.)

From the above example, it is clear that the basic difference between STRONG and WEAK equivalence is the ability to update. Under STRONG equivalence, it is possible to update values related by the \( f \) mapping as discussed at the end of Section III-D. Under WEAK equivalence, the mapping may not have an inverse; thus updating may not be possible. After the database administrators have determined whether the integrated schema will be used for updating or not (and have thus determined whether STRONG or WEAK equivalences must be used), similar object classes from different component schemas should be integrated. The same integration procedure can be used for both STRONG and WEAK equivalences. Thus for the remainder of this paper, we will drop the indication of STRONG and WEAK when specifying the equivalence of attributes.

E. Disjoint Attribute Equivalence

When neither STRONG nor WEAK equivalence exists, it may still be possible to combine attributes. This occurs when the roles of the two attributes are the same, yet there is no mapping function \( f \) that can map values of one attribute to values of the other attribute (i.e., the case when the domain and range of mapping functions are null).

Example 8: Let \( \text{DOM}(\text{CR7}) = \{ \text{Freshman}, \text{Sophomore}, \text{Jr}, \text{Sr} \} \) and \( \text{DOM}(\text{CR8}) = \{ \text{MS}, \text{PhD} \} \). There is no reasonable mapping function between \( \text{DOM}(\text{CR7}) \) and \( \text{DOM}(\text{CR8}) \). However, a new attribute, CR9, can be generated where \( \text{DOM}(\text{CR9}) = \text{DOM}(\text{CR7}) \cup \text{DOM}(\text{CR8}) \).

Definition 7: Attributes \( a \) and \( b \) are disjoint equivalent, denoted \( a \alpha \) DOMAIN-DISJOINT-ROLE-EQUAL \( b \), whenever the following two conditions hold:

1) the domains of \( a \) and \( b \) are disjoint, hence no STRONG or WEAK equivalence holds, and
2) \( a \) and \( b \) have the same roles. By that we mean that they satisfy characteristics 1, 2, 3, 7, and 8 from Definition 1.

Note that with the previous meaning attached to \( \alpha \) (at some point in time) and \( \beta \) (at all points in time), \( \alpha \) DISJOINT equivalence implies \( \beta \) DISJOINT equivalence, because if the domains are disjoint, then the instances will have disjoint values at every time instance.

In the next section we define the equivalences of objects and the equivalences of relationship sets in terms of equivalences of attributes.

IV. OBJECT AND RELATIONSHIP EQUIVALENCE

A. Introduction

Object classes can be related in the same basic ways that attributes can be related. A pair of object classes (like a pair of attributes) can have the following equivalences: EQUAL, CONTAINS, CONTAINED-IN, OVERLAP, and DISJOINT [3]. While we use \( \alpha \) and \( \beta \) to denote equivalences among attributes, we will use \( \rho \) to denote equivalences among object classes. Also, like attributes, object classes may be related STRONG or WEAK. Our approach to determining the equivalence between two object classes is to determine the equivalence between the values of the identifier (key) attributes of those object classes. The object classes will be equivalenced in the same manner as the identifier attributes. This is described in Subsection IV-B. Relationship sets can also be equivalenced based on the equivalences of the identifier attributes of the participating object classes. This is described in Subsection IV-D.

B. Definition of Object Equivalence

Let object class \( A \) have unique identifier \( k_1 \) and attributes \( a_1, a_2, \ldots, a_m \). Let object class \( B \) have unique identifier \( k_2 \) and attributes \( b_1, b_2, \ldots, b_n \). Let \( \text{RWS}(A) \) and \( \text{RWS}(B) \) (called real-world state) refer to the set of real-world instances of object classes \( A \) and \( B \) at a given moment in time. \( \text{RWS} \) is a concept similar to database extension, except that it refers to the underlying real world instances of the object being represented. When two different extensions of objects \( A \) and \( B \) in two different databases actually represent the instances of the same real world objects we say that extension (\( A \)) is not equal to extension (\( B \)). However, \( \text{RWS}(A) = \text{RWS}(B) \).

Let \( T(k_i) \) denote the values of \( k_i \) for all instances of object class \( A \) at a point in time. Likewise, let \( T(k_j) \) denote the values of \( k_j \) for all instances of the object class \( B \) at the same point in time. It is then possible to define a
mapping function \( f : D \rightarrow R \) at each point in time, where 
\( D \) is the largest non-null subset contained in \( T(k_1) \) and \( R \) is the largest non-null subset contained in \( T(k_2) \). The function \( f \) at each moment in time can be derived from a hypothetical function \( f \) between \( \text{DOM}(k_1) \) and \( \text{DOM}(k_2) \). \( \text{DOM}(k_i) \) can be considered to be the union of all the \( T(k_i) \)'s over all possible time instances, and similarly for \( k_2 \). Using this mapping, the equivalence between instances of object class \( A \) and object class \( B \) can be defined. Extending the work of [4], we define five types of \( \rho \) equivalences between \( \text{RWS}(A) \) and \( \text{RWS}(B) \), as illustrated in Fig. 6. Depending upon the relationship of \( k_1 \) and \( D \) and the relationship of \( k_2 \) and \( R \), we define each type of \( \rho \) equivalence and explain how the object classes are integrated.

**Definition 8 (\( \rho \) Equivalences Between Two Object Classes):** \( \rho \) equivalences hold for all points in time, i.e., for all real-world states, and are defined as follows.

**Case 1:** If \( k_1 \) is a \( \text{STRONG} \ \beta \ \text{EQUAL} \ k_2 \), then \( \text{RWS}(A) \ \rho \ \text{EQUAL} \ \text{RWS}(B) \). In this case the object classes \( A \) and \( B \) are integrated into a single class \( AB' \) with unique identifiers \( k_1 \) or \( k_2 \); \( \text{RWS}(AB') := \text{RWS}(A) \cup \text{RWS}(B) \). (Note that \( \text{RWS}(A) = \text{RWS}(B) \) here.)

**Case 2:** If \( k_1 \) is a \( \text{STRONG} \ \beta \ \text{CONTAINS} \ k_2 \), then \( \text{RWS}(A) \ \rho \ \text{CONTAINS} \ \text{RWS}(B) \). In this case the object class \( A \) is made into object class \( A' \) and class \( B \) is made a category (subclass) \( B' \) of object class \( A' \):

\[
\text{RWS}(A') := \text{RWS}(A) \\
\text{RWS}(B') := \text{RWS}(B)
\]

The new object class \( A' \) is an abstraction of object class \( B \).

**Case 3:** If \( k_1 \) is a \( \text{STRONG} \ \beta \ \text{CONTAINED IN} \ k_2 \), then \( \text{RWS}(A) \ \rho \ \text{CONTAINED-IN} \ \text{RWS}(B) \). In this case the object class \( B \) is made into object class \( B' \) and object class \( A \) is made a category (subclass) \( A' \) of object class \( B' \):

\[
\text{RWS}(B') := \text{RWS}(B) \\
\text{RWS}(A') := \text{RWS}(A)
\]

**Case 4:** If \( k_1 \) is a \( \text{DOMAIN-DISJOINT-ROLE-EQUAL} \ k_2 \), then \( \text{RWS}(A) \ \rho \ \text{DISJOINT} \ \text{RWS}(B) \). In this case, integrate as in Case 1 or Case 5.

**Case 5:** If \( k_1 \) is a \( \text{STRONG} \ \beta \ \text{OVERLAP} \ k_2 \), then \( \text{RWS}(A) \ \rho \ \text{OVERLAP} \ \text{RWS}(B) \). In this case, a new object class \( AB' \) is created. Object class \( A \) and object class \( B \) are made categories \( A' \) and \( B' \) of the new object class \( AB' \):

\[
\text{RWS}(A') := \text{RWS}(A) \\
\text{RWS}(B') := \text{RWS}(B) \\
\text{RWS}(AB') := \text{RWS}(A) \cup \text{RWS}(B).
\]

These restrictions are summarized in the following three theorems. (Proofs are found in [16].)

**Theorem 2:** Given \( a \) is an attribute of \( A \), and \( b \) is an attribute of \( B \), if \( \text{RWS}(A) \ \rho \ \text{EQUAL} \ \text{RWS}(B) \), then the only possible equivalence between \( a \) and \( b \) is \( a \ \beta \ \text{EQUAL} \ b \).

**Theorem 3:** If \( \text{RWS}(A) \ \rho \ \text{CONTAINS} \ \text{RWS}(B) \), then either

1. \( a \ \beta \ \text{CONTAINS} \ b \), or
2. \( a \ \beta \ \text{EQUALS} \ b \).

**Theorem 4:** If \( \text{RWS}(A) \ \rho \ \text{CONTAINED-IN} \ \text{RWS}(R) \), then either

1. \( a \ \beta \ \text{CONTAINED-IN} \ b \), or
2. \( a \ \beta \ \text{EQUAL} \ b \).

**D. Definition of Relationship Equivalence**

Relationship sets can be equivalenced in the same basic ways that object classes can be equivalenced. While we use \( \rho \) to denote equivalences among object classes, we will use \( \gamma \) to denote equivalences among ECR relationships.

Let \( A_1, A_2, \ldots, A_n \) be \( n \) object classes that participate in the relationship set RA of degree \( n \). Let \( B_1, B_2, \ldots, B_m \) be \( m \) object classes that participate in the relationship set RB of degree \( m \).

Let \( a_1, a_2, \ldots, a_n \) denote the identifying attributes of object classes \( A_1, A_2, \ldots, A_n \), respectively, and let \( b_1, b_2, \ldots, b_m \) denote the identifying attributes of object classes \( B_1, B_2, \ldots, B_m \), respectively. An instance of a relationship can be identified by the aggregation of the identifying attribute values of the object classes that participate in the relationship. Thus instances of RA can be identified by values of \( (a_1, a_2, \ldots, a_n) \) and instances of RB can be identified by values of \( (b_1, b_2, \ldots, b_m) \).

The equivalences of relationship sets are defined in a
The basic difference is that in object equivalence we analyze the equivalence of the unique identifiers of the objects being integrated, whereas in relationship set equivalence we analyze the aggregation of the identifying attributes of the objects participating in the relationship sets being integrated. Here we assume that a relationship can be thought of as an aggregation of the identifier attributes of the participating entities.

Let \( T(a_1, a_2, \ldots, a_n) \) denote the values of \((a_1, a_2, \ldots, a_n)\) for all possible instances of relationship set \( RA \) at some point in time. Likewise let \( T(b_1, b_2, \ldots, b_m) \) denote the values of \((b_1, b_2, \ldots, b_m)\) for all possible instances of relationship set \( RB \) at the same point in time.

It is then possible to define a mapping function \( f: D \rightarrow R \), with \( D \) as the largest subset contained in \( T(a_1, a_2, \ldots, a_n) \) and \( R \) as the largest subset contained in \( T(b_1, b_2, \ldots, b_m) \). As in the case of entity sets, the function \( f \) at each moment in time can be derived from a hypothetical function \( fr \) between \( DOM(a_1, a_2, \ldots, a_n) \) and \( DOM(b_1, b_2, \ldots, b_m) \). Normally \( n = m \); however, \( m \) may be different from \( n \). This occurs, for example, when two attributes, such as employee number and company number (where both are necessary to identify the employee uniquely) map to a single attribute such as social security number. Using this mapping, the equivalence between instances of relationship set \( RA \) and relationship set \( RB \) can be defined.

Extending the work of [5], we can define five \( \gamma \) relations between \( RWS(RA) \) and \( RWS(RB) \). Depending upon the relationship of \( T(a_1, a_2, \ldots, a_n) \) to \( D \), and \( T(b_1, b_2, \ldots, b_m) \) to \( R \), we define each type of \( \gamma \) equivalence and explain how the relationship sets are integrated.

**Definition 7 (\( \gamma \) Equivalences Between Two Relationship Sets):**

**Case 1:** If \( D = T(a_1, a_2, \ldots, a_n) \) and \( R = T(b_1, b_2, \ldots, b_m) \) for all possible instances of relationship set \( RA \) and \( RB \) at the same point in time, then \( RWS(RA) \gamma \) EQUAL \( RWS(RB) \).

**Case 2:** If \( T(b_1, b_2, \ldots, b_m) \) is properly contained in \( R \) or \( T(b_1, b_2, \ldots, b_m) \) is equal to \( R \), and \( D = T(a_1, a_2, \ldots, a_n) \) for all time instants, then \( RWS(RA) \gamma \) CONTAINS \( RWS(RB) \).

**Case 3:** If \( a_n = b_m \) for all instances of relationship set \( RA \) and \( RB \), then \( RWS(RA) \gamma \) CONTAINED-IN \( RWS(RB) \).

**Case 4:** If \( D \cap R \neq \emptyset \) for all possible instances of relationship set \( RA \) and \( RB \), then \( RWS(RA) \gamma \) DISJOINT \( RWS(RB) \).

**Case 5:** If any of the constraints in Cases 1–4 can hold at different time instants then \( RWS(RA) \gamma \) OVERLAPS \( RWS(RB) \). In this case, a new relationship set \( RAB' \), is created, where \( RWS(RAB') := RWS(RA) \cup RWS(RB) \).

Relationship sets subjected to integration would generally be both binary (\( m = n = 2 \)). However, it is possible to integrate relationship sets of different degrees. An example follows.

**Example 9 (Fig. 7):** The relationship set Owns is of degree 2; the other relationship set CAR-OWNERSHIP is of degree 1. (By degree 1, we mean that the relationship set is an entity class itself.) Because there is a one-to-one mapping between the values of the identifier attributes (SSN and LIC#) of Car-Ownership and the aggregation of the identifier attributes (SSN and LIC#) of the object classes (Person and Car) of the Owns relationship set, the relationship set Owns can be integrated with the entity class CAR-OWNERSHIP. When integrating relationship sets of different degree, we propose that the higher degree relationship set always appear in the integrated schema.

Thus the result of the integration is a relationship set of degree 2 (with whatever name).

Example 9 illustrates the solution to another problem sometimes encountered when integrating ECR schemas. The problem is how to integrate an entity set with a relationship set. The above approach to integrating relationships of different degrees solves this problem.

V. STRATEGIES FOR ATTRIBUTE INTEGRATION

We have used the concept of attribute (\( \beta \) equivalences to define object class (\( \rho \) equivalences and relationship set (\( \gamma \) equivalences. In Subsection IV-C we described some restrictions to attribute equivalence that are imposed by the equivalences of object classes containing the attributes. However, within these constraints there are several approaches for integrating attributes given their equivalences. These approaches are described in this section.

A. Four Strategies

We now describe four approaches to integrating object classes \( A \) and \( B \) and their resulting situation regarding their attributes as illustrated in Figs. 8–11. Each row of a table represents one of the possible relationships of object
Fig. 8. Attribute integration strategy.
### Attribute Integration Strategy 3

<table>
<thead>
<tr>
<th>CASE 1: RWS (A) p-EQUAL RWS (B)</th>
<th>a (\Delta) EQUAL b</th>
<th>a (\Delta) CONTAINS b</th>
<th>a (\Delta) CONTAINED-IN b</th>
<th>a (\Delta) OVERLAP b</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>A</td>
<td>C</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Attribute Integration Strategy 4

<table>
<thead>
<tr>
<th>CASE 1: RWS (A) p-EQUAL RWS (B)</th>
<th>a (\Delta) EQUAL b</th>
<th>a (\Delta) CONTAINS b</th>
<th>a (\Delta) CONTAINED-IN b</th>
<th>a (\Delta) OVERLAP b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Fig. 10. Attribute integration strategy 3.

Fig. 11. Attribute integration strategy 4.
classes $A$ and $B$ illustrated in Fig. 6. Each column represents one of the four possible equivalences between attributes $a$ of $A$ and $b$ of $B$ presented in Section III. The intersection of each row and column indicates how attributes $a$ and $b$ should be integrated. Each figure illustrates only two equivalent attributes: $a$ of object class $A$ and $b$ of $B$. Additional attributes of $A$ and $B$ are not shown but are handled in the same way. $A'$, $B'$, and $AB'$ denote the object classes resulting from the integration of $A$ and $B$. Figs. 8-11 use $a'$, $b'$, and $c'$ to denote the attributes of the integrated object types. The blank positions in Figs. 8-11 indicate cases that will never occur due to Theorems 2, 3, and 4.

Using either the strong or weak definition of attribute equivalence, there are at least four strategies for merging attributes. Note that these strategies deal with multivalued attributes in general. This is important because we will use these strategies to integrate relationships; this is similar to integrating multivalued attributes.

Strategy 1 (Integrate All Nondisjoint Attributes): If attributes $a$ and $b$ are not disjoint, then integrate them to form attribute $c'$ (except for those cases prohibited by Theorems 2, 3, and 4). When two object classes are integrated, the equivalent attributes are always integrated to form a new attribute $c'$ by assigning $\text{DOM}(c')$ as indicated in Fig. 8. $AB'$ represents an object class created in the integrated schema from the source object classes $A$ and $B$. Instances of $AB'$ account for $\text{RWS}(A) \cup \text{RWS}(B)$. $A'$ and $B'$ represent the target object classes corresponding to the source object classes $A$ and $B$. Note that unlike subsequent strategies, here the outcome of integration in a given row is the same regardless of the column. Of all the strategies we consider, Strategy 1 results in the integrated schema with the smallest number of attributes. Users of a schema derived by Strategy 1 do not need to choose between closely related attributes that are not disjoint because these attributes have been integrated. This schema may be suitable for infrequent and novice users who want to avoid the complexity of many similar attributes.

Strategy 2 (Integrate Only Attributes That Are $\beta$ Equal and Indicate Relationships Between Nonintegrated Similar Attributes): This strategy captures containment constraints among multivalued attributes of the integrated object class and is illustrated in Fig. 10. For $a \beta \text{EQUAL} b$ (column 1), the result is the same as in Strategy 2. For $a \beta \text{CONTAINS} b$ (column 2) and $a \beta \text{CONTAINED-IN} b$ (column 3), there is a new constraint between the values of attributes $a'$ and values of $b'$ that was missing in Strategy 2. For $a \beta \text{OVERLAP} b$ (column 4), there is both a new attribute $c'$ and new constraints between the values of attribute $a'$ and $c'$ and between the values of $b'$ and $c'$. In column 2 the containment condition holds after applying the inverse mapping function to attribute $b$. In column 3 the containment condition holds after applying the mapping function to attribute $b$.

An integrated schema derived using Strategy 3 may have a few more attributes than the integrated schema derived using Strategy 2. Unlike Strategy 2, the integrated schema derived using Strategy 3 explicitly describes the equivalences of attributes. The additional attributes and the explicit equivalences between similar but not integrated attributes gives the user much semantic information, which also makes integrated schemas derived using Strategy 3 the most complex.

Strategy 4 (Integrate All Nondisjoint Attributes and Migrate Values Between Attributes): Fig. 11 illustrates yet another approach for integrating attributes. Like Strategy 1, Strategy 4 integrates all nondisjoint attributes. Unlike Strategy 1, Strategy 4 permits common values in two of the original attributes to appear only once in an integrated attribute. Values that appear in only one of the two original attributes will remain in the original attribute. Fig. 12 shows the application of this strategy to multivalued attributes. For example, consider the two object class types shown in Fig. 12(a), where $A \rho \text{OVERLAP} B$ and $a \beta \text{CONTAINS} b$ and $a$ and $b$ are both multivalued attributes. Example instances of $A$ and $B$ are shown in Fig. 12(b). The integrated object class types are shown in Fig. 12(c), with instances shown in Fig. 12(d). Because $\{a_1, a_2\}$ are common to both $A_1$ and $B_1$, $\{a_1, a_2\}$ are migrated to the new attribute $c'$ of $AB'$.

B. Another Approach for Integrating Relationship Sets

Relationship sets can be integrated by considering the relationship set to be an attribute (possibly multivalued) of each object class participating in the relationship set. This attribute is also an aggregate attribute, which con-
consists of 1) all attributes of the original relationship set, and 2) the identifying attributes of the remaining object classes that participate in the relationship. Because an object may participate in several relationships, the object may have several values for the aggregate attribute, a set of values for each relationship in which the object participates; hence, the aggregate attribute may be multivalued. For example, in Fig. 13(b), the attribute $r_1$ of object class $A$ consists of all attributes of the relationship $R_1$ and the identifying attribute of $D$, while the attribute $r_1$ of object class $D$ consists of all attributes of the relationship $R_1$ and the identifying attribute of $A$. If object class $A$ participates in three instances of relationship $R_1$, there will be a set of three aggregates of values for the aggregate attribute $r_1$ of the object class $A$.

We can then integrate the relationship sets by treating them as aggregate attributes and using either Strategy 3 or 4. We declare the two aggregate attributes $\beta$ EQUIVALENT only if each of the respective subattributes that make up the aggregate are $\beta$ EQUIVALENT. After integrating the aggregate attributes, they are replaced by the corresponding relationships. The following examples illustrate the approach using Strategy 4.

**Example 11:** Using Strategy 4 permits migration of relationship instances from one relationship to another. For example, consider the two relationships $R_1$ and $R_2$ shown in Fig. 13(a). Replace the relationships with attributes, as shown in Fig. 13(b). Assume the constraints are $A \beta$ OVERLAP $B$, $D \beta$ CONTAINS $E$, and $r_1 \gamma$ EQUAL $r_2$. Following the strategy in Fig. 11, the integrated entities are shown in Fig. 13(c). We use $r_i$ to denote the integrated attribute for $r_1$ and $r_2$. Then replacing each attribute by the corresponding relationship yields the integrated relationships shown in Fig. 13(d).

VI. CONCLUSION AND FUTURE WORK

We have extended our previous work in schema integration by identifying a single basic concept used throughout schema integration. This basic concept is attribute equivalence and a corresponding attribute integration. Using this, we have defined equivalences between object classes and between relationship sets. The definition of equivalence between relationship sets handles the special case where an object class and a relationship set are integrated.

Many of the traditional problems encountered during schema integration can be rephrased in terms of attribute equivalence:

- **Naming Conflicts:** Corresponding attributes have different names and noncorresponding attributes have the same name. By applying the criteria for (strong, weak, and disjoint) attribute equivalence, corresponding attributes with different names can be identified and merged. Noncorresponding attributes with the same name can be identified and one or both attributes renamed.
- **Scale Difference:** Scale difference is a special type of strong equivalence in which the mapping function $f$ involves a scale factor. For example, if attribute $a$ is expressed in meters, and attribute $b$ is expressed in kilometers, the mapping function $f$ can be expressed as $b = f(a) = a \cdot 10^{-3}$. Usually scale difference is a strong equivalence because there exists an inverse mapping. In the above example $a = f^{-1}(b) = b \cdot 10^3$.
- **Differences in Level of Abstraction of Attributes:** Suppose there are two attributes: telephone-number and home-telephone-number.

- Telephone-number may contain both home and office telephone number.
—Obviously, telephone-number α CONTAINS home-telephone-number.

- **Differences in Object Identifiers:** Suppose two objects are similar except that they have different identifiers. The two objects can be integrated if correspondence between values of the respective identifiers can be established. This may be established by the mapping function \( f \) and its inverse.

- **Differences in Representation:** It is possible for a concept to be represented by an entity set in one schema and a relationship set in another schema. By considering the identifying attributes of these different constructs, it is possible to determine their equivalence. The general relationship set integration strategy handles this case, as well as other cases in which the degrees of the relationship sets to be integrated are not the same.

We have designed and implemented a schema integration tool based partially on this work. This tool presents the database administrator with descriptions of the schemas to be integrated. Applying a systematic methodology described in [6], the database administrator specifies all equivalences between schema objects and interactively causes the schemas to be integrated.

Future research in this area will include an investigation into an expert system approach for schema integration driven by attribute equivalence information. Another area of investigation where this work plays an important role is the mapping of queries from global schemas into local schemas.

**REFERENCES**


James A. Larson (S’77–M’81) received the Ph.D. degree in computer science from Washington State University, Pullman, in 1977. He is a Senior Research Fellow working on distributed databases and artificial intelligence projects at Honeywell Corporate Systems Development Division in Minneapolis, MN. He is also Adjunct Professor with the Department of Computer Science, University of Minnesota, where he teaches courses in database management. Previously he was involved in the design of integrated automated office systems for Olivetti, Italy, and research in database computers at Sperry Univar.

Shamkant B. Navathe received the B.E. degree in electrical communications engineering from the Indian Institute of Science, Bangalore, India, the M.S. degree from Ohio State University, Columbus, and the Ph.D. degree from the University of Michigan, Ann Arbor.

He is with the Database Systems Research and Development Center and is a Professor of Computer and Information Sciences at the University of Florida in Gainesville, where he has been since 1979. He is known for his work on distributed databases and artificial intelligence projects at Honeywell Corporate Systems Development Division, Minneapolis, MN, Tech. Rep. CSC-86-9-8212, Apr. 1986.

Ramez Elmasri (S’74–M’80) received the B.S. degree in electrical engineering from Alexandria University, Egypt, and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA.

He is an Assistant Professor in the Department of Computer Science, University of Houston, Houston, TX, where his research interests include semantic integrity of databases, distributed databases, database design and integration, and query languages and user interfaces. He is the principal investigator on an NSF grant to study semantic integrity of database transactions. He is the author of the book, *Fundamentals of Database Systems*, with R. Elmasri. His current work addresses database integration, models and management of historical databases, CAD/CAM, and the design of intelligent interfaces to databases. Prior to his current position, he was on the faculty of New York University’s Graduate School of Business Administration and a system engineer for IBM World Trade Corporation and Electronic Data Systems.

Dr. Navathe was the Program Chairman of the 1985 ACM-SIGMOD International Conference on Management of Data in Austin, Texas, and is an associate editor of *Data and Knowledge Engineering* and *ACM Computing Surveys*. 