

On Applying Classification to Schema Integration[#]

Ashoka Savasere^{*} Amit Sheth^{**} Sunit Gala^{*}
Shamkant Navathe^{*} Howard Marcus^{**}

Bellcore^{**}
444 Hoes Lane, Rm 1J-210
Piscataway, NJ 08854 USA
amit@ctt.bellcore.com

Database R & D Center^{*}
University of Florida
Gainesville FL 32611 USA
sunit@beach.cis.ufl.edu

Abstract

In this paper, we discuss how the concepts of subsumption and classification (as described in the KL-ONE family of systems) can be adapted to automatically identify various relationships among classes. Classes can overlap, or be equivalent or disjoint, or one class may include another class. Computing such relationships is based on the subsume function, which determines whether a given class is a superclass of another class. Relationships thus identified can be used to merge classes of two or more schemas (i.e., integrating these schemas) with the help of the classification function and other schema restructuring operators.

1. Introduction

Many approaches and techniques for schema integration have been reported in literature. The survey paper of Batini et al. [1986] discusses and compares twelve methodologies for schema integration. It divides the process of schema integration into five steps: preintegration, comparison, conformation, merging, and restructuring. The most important and difficult step is comparison which involves determining relationships among objects in one schema with the objects in another.

In our approach, two classes (such as entity types in the E-R model), $C1$ and $C2$, may be related by one of following relationship types: $C1$ equals $C2$, $C1$ includes (or is-included-in) $C2$, $C1$

overlaps $C2$, and $C1$ is-disjoint-with $C2$. While classes with equivalence and inclusion can always be merged (possibly creating new classes), the classes that are disjoint or overlapping may be merged or not, depending on the integrator's decision (which may be dependent on the intended use of the integrated schema). In earlier methodologies, all class relationships are determined and specified by a human (the schema integrator). For example, Elmasri et al. [1986] and Sheth et al. [1988] use the relationships among attributes in a heuristic algorithm to identify pairs of entity types and relationship types that may be related by equivalent, inclusion, and disjoint types of relationships. The actual relationships are specified by the integrator. However, we believe that determining these relationships automatically (i.e., without human involvement), whenever possible, is very desirable.

In our approach, we exploit the notions of subsumption and classification (as described in the family of KL-ONE systems [Brachman and Schmolze 1985]) to automatically determine relationships among classes. Using the CANDIDE semantic model [Beck et al. 1989] was particularly appealing since it adapts the KANDOR knowledge representation system [Patel-Schneider 1984] (a KL-ONE derivative) to traditional data modeling. In this paper, however, we shall limit our attention to discussing how to exploit the

[#]This project was funded by and performed at Bellcore.

notions of classification and subsumption for schema integration. Details of our approach can be found in [Sheth and Gala 1990], and the same techniques can be applied for integrating extended E-R schemas [Savasere 1990, and Sheth and Gala 1990].

2. Subsumption and Classification

A class in CANDIDE is defined by its attributes, and the cardinality and domain constraints on these attributes. Type constructors are also provided to define complex domains [Beck et al 1989].

A class f subsumes a class g if and only if every instance of g is also an instance of f , i.e., f is a superclass of g . This subsumption relationship is computed on the basis of whether the attribute constraints for class g logically imply the attribute constraints for class f . Computing subsumption can be automated because a class definition provides necessary and sufficient conditions for deciding class membership.

Classification can be viewed as the process of correctly locating a given class in an existing taxonomy¹. The correct location is immediately below the most specific classes which *subsume* the new class and immediately above the most general classes *subsumed* by this new class.

3. Operators to Determine Class Relationships

As mentioned above, it is important to automatically compute various class relationships such as *equivalent*, *includes*, *is-included-in*, *overlaps*, and *disjoint*. The *subsume* function computes *includes* and *is-included-in* in an obvious

¹One way of computing classification is to take the transitive reduction over a boolean matrix generated by computing the subsumption relationship between all possible pairs of classes in the database schema, i.e., class taxonomy. Since there are n^2 such pairs, classification is an $O(n^2)$ algorithm where the fundamental unit of computation is the subsumption operation. However, computing subsumption is at least co-NP-hard [Nebel 1988].

manner. We now give a few definitions which show how these relationships can be computed with the help of the *subsume* function. Let $E[f]$, $E[g]$ represent the extensions of classes f and g , respectively.

1. Subsume:

$$\text{subsume}(f, g) = \text{true iff } E[f] \supseteq E[g]$$

However, as explained earlier, subsumption is computed on the basis of class definitions and not the actual extensions. This means that the *subsume* function returns true if and only if the constraints on each attribute of g logically imply the constraints on the corresponding attribute of f . Therefore, $\text{subsume}(f, g) \equiv f$ includes $g \equiv g$ is-included-in f .

2. Equivalence:

$$\text{equivalent}(f, g) = \text{true iff } E[f] \equiv E[g]$$

This function can be computed using subsumption as follows:

$$\text{equivalent}(f, g) = \text{subsume}(f, g) \wedge \text{subsume}(g, f)$$

3. Overlap:

Overlap is defined and computed as follows:

$$\text{overlap}(f, g) = \text{true iff } \neg \text{subsume}(f, g) \wedge \neg \text{subsume}(g, f) \wedge \neg \text{disjoint}(f, g)$$

That is, two classes are overlapping if they do not subsume each other, and are not disjoint. As an example, consider the two classes *Instructor* and *Student*. They can overlap because an instance of *Instructor* may also be an instance of *Student*, such as a teaching assistant.

4. Disjoint:

Two classes are defined as disjoint if the intersection of all possible extensions of the two classes is empty.

$$\text{disjoint}(f, g) = \text{true iff } E[f] \cap E[g] = \emptyset$$

Disjoint can be computed as follows:

$$\text{disjoint}(f, g) = \text{true iff } \text{incoherent}(\text{conjunction}(f, g))$$

Here `conjunction` is a function which returns a new class from two given classes such that the extension of the new class is the intersection of the extensions of the given classes.

$$E[\text{conjunction}(f, g)] = E[f] \cap E[g]$$

`Incoherent` is a boolean function which tests for logical inconsistency in constraints on the attributes of a given class. For example, the maximum cardinality cannot be less than the minimum cardinality on any attribute (for more details see Patel-Schneider 1984). This means that there cannot be any instances for an incoherent class: $E[\text{incoherent}(f)] = \emptyset$. We compute incoherency of a class by checking if the class is subsumed by a known incoherent class: $\text{incoherent}(f) = \text{subsume}(i, f)$ where i is a known incoherent class.

For example, the class *Part-time-student* from schema *sc1* with the constraint that a part time student can register for a maximum of 9 credit hours, and the class *Full-time-student* from schema *sc2* with the constraint that a full time student must register for at least 12 credit hours clearly show that there cannot be a student who is both part time and full time. As another example, if a class *Student-organization* has the constraint that it must have at least 2 part-time students among its members and the class *Honor-society* has the constraint that only a full-time student can be its member, then evidently these two classes do not have a common instance. A conjunction of these two classes will lead to an incoherency. Therefore, the classes are identified as disjoint.

The functions defined above are based on the ability to compute subsumption, i.e., they are based on the semantics of class definitions. However, it is possible to define boolean functions which return true or false based on more syntactic criteria such as attribute relationships. Therefore, we define two additional operators.

5. `attr_overlap(f,g)`: This function returns true if and only if there exists at least one pair of attributes a_1 and a_2 such that $a_1 \equiv a_2$ or $a_1 \subset a_2$ or $a_2 \subset a_1$ where a_1 is any attribute of f and a_2 is any attribute of g .

6. `attr_disjoint(f,g)`: This function returns true if and only if there does not exist any pair of attributes a_1 and a_2 such that $a_1 \equiv a_2$ or $a_1 \subset a_2$ or $a_2 \subset a_1$ where a_1 is any attribute of f and a_2 is any attribute of g .

All these functions can be computed automatically. The user can now restructure the global schema based on his/her perspective of the domain of discourse with the help of these functions.

4. Schema Restructuring Operators

Depending on the class relationships determined by using the operators discussed above, another set of operators can be used to create new classes from existing ones and/or delete existing classes.

- `delete(f)`: This function will delete the class f from the integrated schema and check for consistency. If the resulting schema is found inconsistent, then the user is alerted with an appropriate message.
- $c = \text{generalize1}(f,g)$: The two classes f and g are generalized to create a new class c (the label for class c is provided by the user, e.g., $\text{teacher} = \text{generalize1}(\text{faculty}, \text{instructor})$). Only the common attributes are chosen for c , and a union (or logical disjunction) of the constraints on these attributes is defined. The resulting class is checked for incoherency.
- $c = \text{generalize2}(f,g)$: The same as `generalize1` except that f and g are now deleted from the schema.
- $c = \text{specialize1}(f,g)$: The two classes f and g are specialized to create a new class c (the label for class c is provided by the user). For

the common attributes of f and g chosen for c , the logical conjunction of the constraints on these attributes is defined. The remaining attributes are merely concatenated to the definition of c . The resulting class is then checked for incoherency.

- $c = \text{specialize2}(f,g)$: The same as specialize1 except that f and g are then deleted from the schema.

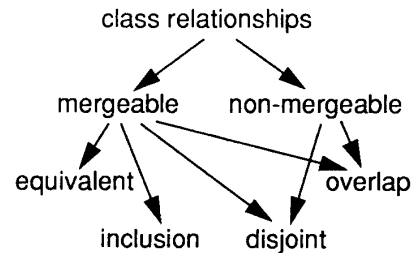
5. Applying functions to integrate schemas

To integrate two schemas, $sc1$ and $sc2$, we apply the above class comparison functions between each pair of classes f and g such that $f \in sc1$ and $g \in sc2$ (it is assumed that each individual schema is already "well" defined). Whenever two classes are identified as *equivalent*, they are merged into a single class. Similarly, when one class *subsumes* another class, the integrated schema is appropriately restructured. The user then applies the schema restructuring operators to get a final, integrated schema based on his knowledge of the universe of discourse. Disjoint or overlapping classes may or may not be merged depending on the situation. For example, "birds" and "aircraft" may be disjoint classes, but can be merged under a common superclass called "flying objects". Similarly, "instructors" and "students" may overlap, and can possibly merged to form a common subclass called "teaching assistants". On the other other hand, regardless of whether "courses" and "instructors" overlap or are disjoint, they need not be merged into a common class, even though they are related in the sense that instructors teach courses. Such decisions are made solely by the schema integrator.

All of the above functions have been implemented at Bellcore, as a part of a schema integration tool, by extending the CANDIDE system developed at the University of Florida.

References

Batini, C., Lenzerini, M., and Navathe, S. B.,



A comparative Analysis of Methodologies for Database Schema Integration, *ACM Computing Surveys*, Vol. 18, No. 4, December 1986, pp. 323-364.

Beck, H. W., Gala, S. K., and Navathe, S. B., Classification as a Query Processing Technique in the CANDIDE Semantic Data Model, *Proceedings of the Fifth International Conference on Data Engineering*, Los Angeles, February 1989.

Brachman, R. J., and Schmolze, G., An Overview of the KL-ONE Knowledge Representation System, *Cognitive Science*, Vol. 9, No. 2, April-June 1985, pp. 171-216.

Elmasri, R., Larson, J., and Navathe, S. B., Schema Integration Algorithms for Federated Databases and Logical Database Design, *Honeywell CSDD, Technical Report CSC-86-9, 8212, January 1986*.

Nebel, Bernhard, Computational complexity of terminological reasoning in BACK, *Artificial Intelligence*, Vol. 34, No. 3, 1988.

Patel-Schneider, P. F., Small can be Beautiful in Knowledge Representation, *Technical Report, No. 37, FLAIR, October 1984*.

Savasere, A., An Approach to Schema Integration Using Classification, *MS Thesis, Department of Computer and Information Sciences, University of Florida, Gainesville, 1990*.

Sheth, A. P., Larson, J. A., Cornelio, A., and Navathe, S. B., A tool for Integrating Conceptual Schemas and User views, *Proceedings of the Fourth International Conference on Data Engineering*, Los Angeles, February 1988.

Sheth, A. P., and Gala, S. K., Attribute Relationships: An Impediment in Automating Schema Integration, *Workshop on Heterogeneous Database Systems, Chicago, December 1989*.

Sheth, A. P., and Gala, S. K., On Automatic Reasoning for Schema Integration, Technical Memo, Bell Communications Research, NJ, (in preparation), 1990.