Understanding Schemas*

R. J. Miller† Y. E. Ioannidis‡ R. Ramakrishnan§
Dept. Computer Sciences, Univ. Wisconsin
1210 W. Dayton St., Madison, WI 53706 USA
{rmiller, yannis, raghu}@cs.wisc.edu

Abstract

Before the problem of schema integration and translation can be adequately addressed, a precise understanding of schemas is needed. We present an analysis of the notion of schema as used by existing integration methodologies. We show how inherent ambiguities and imprecision in traditional definitions of schema can hamper the development of formal schema integration methodologies. Specifically, traditional notions of schema contain data in the form of metadata, as well as superfluous structuring information that is not semantically meaningful. We argue that it is important to cleanly separate structural information from data, and remove from consideration artifacts of a specific data model or design methodology. To this end, we introduce the notion of a schema intension to capture the semantic content of a schema.

1 A Closer look at Schemas

Current work on schema integration uses the notion of schema as defined by a specific data model or design methodology. These notions vary in such areas as the kinds of structural associations (and constraints) that may be expressed, in the way constraints are grouped to form data model constructs, and in the forms of names that may be associated with schema constructs. The choice of a data model or design methodology, as well as the view of metadata, may impact what constitutes a schema.

The rich semantic models (favored within the schema integration community) provide a plethora of constructs that permit the same information to be represented in different constructs of the data model. In addressing the problem of schema integration it is important to understand when a choice of construct in a given schema (for example, modeling information as an entity or a relationship) conveys semantic information and when a choice is arbitrary. In the former case, the choice of construct should be used in reasoning about a schema. In the latter, the choice is an artifact of the data model or of the freedom in representation allowed by the data model. This dichotomy is fundamental to the problem of schema integration and it is one that has been largely ignored in existing integration methodologies.

Also inherent to the problem of understanding and defining schemas is the fact that traditional schemas may include data in the form of names for structures within the schema. In Figure 1, three relational schemas are shown for stock information.[1] All three schemas intuitively model similar information. Yet information that is captured by data in the first schema (as values in a specific schema instance) is expressed within the schema itself (as either names of tables or names of attributes) in the second and third schemas. Examples such as this are not uncommon since names for entities often capture some intuitive semantic information. [2]

Indeed, it is not just the names of entities that may be viewed as data. Statistics about tables, access patterns, and even comments about units of quantities (e.g., whether prices are in dollars or francs) are forms of metadata that may be viewed as data. In fact, any information within a traditional database catalog may be viewed as data.

Schema design methodologies influence what data

*Appeared in: Research Issues in Data Engineering: Interoperability in Multidatabase Systems, 1993.

†Partially supported by NSF Grant IRI-9157368.

‡Partially supported by NSF Grants IRI-9113736 and IRI-9157368 (PYI Award) and by grants from DEC, HP, and AT&T.

§Partially supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering, by the NSF under a PYI Award and under grant IRI-9011563, and by grants from DEC, Tandem, and Xerox.

[1] Variations of this same example have been used in [1, 5].

[2] Other researchers have recognized that schemas often contain data in the form of names for schema constructs. However, work has been focused on providing second order reasoning capabilities over names of schema constructs [1, 5, 8]. In contrast, our goal is to understand formally what data is being represented by a schema, not to address language requirements or allow queries over metadata.

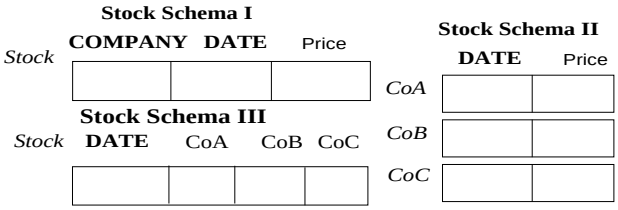

Figure 1: **Three stock schemas. Table names are in italics, key attributes are capitalized.**

is seen as metadata and define (implicitly perhaps) a common view of precisely what comprises a schema. Many integration methodologies have taken advantage of this fact to simplify the problem of integration. The focus is on merging structures within the various schemas making incremental, rather than fundamental, changes to the structures themselves [2, 3, 6, and others]. The assumption is that a common design methodology will ensure that the structures chosen to represent similar information in different schemas are inherently compatible. As a result the semantics implied by the choice of a certain structure is rarely captured explicitly within these methodologies other than by the goal of changing the initial schemas as little as possible.

2 Redefinition of Schema

Given the above concerns, a definition of schema that is appropriate for the task of schema integration is needed. This definition should capture the information about how data is related together or structured and should clearly separate structuring information from the data and the metadata that form parts of traditional notions of schema.

We will refer to what has traditionally been called a schema (that is, a schema, expressed within a given data model) as a *schema extension*. Schema extensions carry with them all the ambiguity and imprecision that has been explored in the previous section and may include data in the form of names for constructs. A *schema intension*, on the other hand, is defined to be a "pure" data structure. A schema intension serves to structure information that in a traditional schema extension may be expressed as both metadata and data. To ensure that a schema intension is a "pure" data structure, constructs within a schema intension are referred to by tokens with no semantic content.

We present a simple, intuitive formalism for capturing the essence of schema intensions called *schema intension graphs* (SIGs). We present only the features of SIGs necessary to present the example of Section 3 (SIGs are fully defined in [7]). The notion of a schema intension is independent of the specific model we present below. Our exposition will therefore focus on the use of this model in demonstrating the principles underlying the analysis of schemas using schema intensions.

The formalism we present uses annotated binary relations expressed over domains represented by nodes in a graph. An annotated binary relation is a mathematical relation constrained to satisfy certain properties. The set of properties allowed will determine, to a large extent, the power of this formalism in expressing structural information. Our proposal uses only four, rather natural properties. Specifically, an annotated binary relation may be classified as being total, surjective, injective or functional, or any combination of these properties. If r is a binary relation defined on two sets A and B, denoted $r : A - B$, the inverse of r is defined as $r^\circ : B - A$ where $(b, a) \in r^\circ$ if and only if $(a, b) \in r$. If r and s are two binary relations such that $r : A \leftrightarrow B$ and $s : B \leftrightarrow C$, their composition is defined by $s \circ r : A \leftrightarrow C$ where $(a, c) \in s \circ r$ if and only if there exists an element $b \in B$ such that $(a, b) \in r$ and $(b, c) \in s$. A binary relation $r : A - B$ is said to be *total*, denoted $r : A \!\!+\!\!\!- B$, if for all $a \in A$, there exists some $b \in B$ such that $(a, b) \in r$. Similarly, r is said to be *surjective*, denoted $r : A \!\!-\!\!\!+ B$, if its inverse r° is total. A binary relation $r : A - B$ is *functional*, denoted $r : A \longrightarrow B$, if whenever $(a, b_1) \in r$ and $(a, b_2) \in r$ then $b_1 = b_2$, so an element a uniquely determines a single element b. Finally, a binary relation r is *injective*, denoted $r : A \longleftarrow B$, if r° is functional. Note that all four properties are independent in that no subset of the properties expressed on a binary relation between arbitrary sets A and B logically implies any property not in that subset.

Let Λ be an infinite set of symbols that will serve as labels. Let $\mathcal{T}$ be a finite set of abstract types. Each type $\tau \in \mathcal{T}$ is an infinite set of symbols. All types are pairwise disjoint and disjoint from the set of labels Λ. A *schema intension graph* (SIG) is a graph $G = (N, E)$ defined by two finite sets N and E. Let $M \subseteq \Lambda$ be a finite set of symbols and M^* be the closure of M under finite products and sums. Then $N \subseteq M^*$ and $N \supseteq M$. For $X \in N$, if $X \in M$ then X is called a *simple node*, otherwise X is a *constructed node*. Each simple node is assigned a (not necessarily unique) type, $\tau(X) \in \mathcal{T}$. Each element $e \in E$ is a labeled edge between two nodes of N, where $\lambda(e) \in \Lambda$. An edge e is denoted $e : X - Y$ indicating it is an edge between nodes X and Y. The set E contains special projection edges

from each product node $X \times Y$, $p_1 : X \times Y - X$ and $p_2 : X \times Y - Y$. An annotation of a SIG $G = (N, E)$ is a function $\mathcal{A}$ whose domain is the set of simple paths in G where $\mathcal{A}(e) \subseteq \{f, i, s, t\}$.

An *instance* of a SIG $G = (N, E)$ is a function $\Im$ whose domain is the sets N and E. For each simple node $X \in N$, $\Im(X)$ is a finite subset of $\tau(X)$. For each product node $X \times Y \in N$, $\Im(X \times Y)$ is the full cross product of the sets $\Im(X)$ and $\Im(Y)$. For each sum node $X + Y \in N$, $\Im(X + Y)$ is the disjoint sum of the sets $\Im(X)$ and $\Im(Y)$. For each edge $e : X - Y \in E$, $\Im(e)$ is a binary relation over $\Im(X)$ and $\Im(Y)$ (i.e., a subset of the cross product of $\Im(X)$ and $\Im(Y)$). An instance of a projection edge $p : X \times Y - X$ is constrained so that $((x_1, y), x_2) \in \Im(p)$ iff $x_1 = x_2$. For an annotation function $\mathcal{A}$, a *valid instance* of $(G, \mathcal{A})$ is an instance that assigns a total (respectively functional, surjective or injective) binary relation to each path where $t \in \mathcal{A}(e)$ (respectively $f, s,$ or $i \in \mathcal{A}(e)$).

3 Expressing Schema Intensions

In this section, we present an example (using the schemas of Figure 1) of how the SIG formalism can be used to capture the essential structuring properties of schema extensions and to reason about all data represented by a schema extension.

In describing the intension of a given schema extension, different nodes must be assigned to each subset of an abstract type that can act in a different structural role. Restrictions on the values of these nodes (including equality of nodes) are then captured by annotations on edges of the SIG. In Schema I of Figure 1, the three abstract types κ, δ, and π represent company values, date values and price values, respectively. There is a single node for each of these types, C, D and P, each corresponding to a column of the stock table. The final grouping of data, the stock table itself, is not semantically significant in this example and is not assigned an abstract type or node.

Existing integration approaches provide mechanisms for expressing and integrating data that is structured *by* the schema extension. It is the data captured *within* the schema extension that is often omitted. Hence, this metadata must also be typed and grouped into nodes. In Schema I of Figure 1, the table name and attribute names are labels, rather than data. In Schema III, however, the three attribute names, *CoA*, *CoB* and *CoC* are all values of the abstract type κ (company values). These attribute names are assigned to a node C'. The Date column is assigned the abstract type δ (date values) and node D'. The remaining three columns are all assigned the abstract type π (price values). To simplify the example we assign a single node P' to represent all three columns of price values. In a full development of this example separate nodes would be assigned to these columns. However, because the three columns play the same structural role, this simplification can be made without loss of generality.

Given an assignment of all data to typed nodes, annotated binary relations capturing relevant structuring information can be derived. A formalization of this derivation process would include, for example, rules to express the constraint that the node for a relational table name and the nodes for key attributes of the table together functionally determine the node of each nonkey attribute. Inference rules allow annotated binary relations to be combined and simplified. (Derivation rules for classes of the relational model and inference rules on SIGs are presented elsewhere [7].) In the following we focus on the intuition behind the derivation of annotated binary relations.

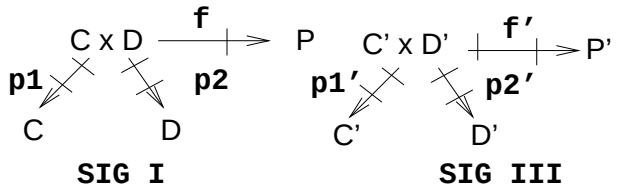

Figure 2: **The SIGs for Schemas I and III.**

The structure on nodes implied by the stock table of Schema I can be captured by SIG I of Figure 2. The edges p_1 and p_2 are the projections from $C \times D$ of C and D values respectively. The surjective functional edge f expresses the constraint that a company and data value determine a single price, and that every price value is associated with at least one company, date pair. This edge is not total since every value in the cross-product of company and date values is not guaranteed to appear in a valid instance of Schema I. However, every company value is associated with some price, so the additional constraint that $p_1 \circ f^\circ$ is surjective must be added. Similarly, $p_2 \circ f^\circ$ must be surjective.

The structure on nodes implied by the stock table of Schema III can be captured by SIG III of Figure 2. Again, the edges p'_1 and p'_2 are projections. The edge f' expresses the constraint that a date value together with an attribute name (a company value) determines a single price. Again, every price is associated with at least one company, date pair so the edge f' is surjective. However, unlike in Schema I, the structure of Schema III ensures that every company has a price

recorded for *every* date (assuming null values are not allowed). Hence, edge f' is also total. As with Schema I, the structure of the relational table ensures that every company is associated with some price value and every data is associated with some price. These constraints are again captured by requiring that $p'_1 \circ f'^{\circ}$ and $p'_2 \circ f'^{\circ}$ be surjective.

SIGs I and III are isomorphic except for the total annotation on f' in SIG III. Schema III is therefore more restrictive than Schema I in that the set of valid instances of this schema will be a strict subset of the set of valid instances of Schema I. Continuing this case study, a SIG for Schema II may be derived. This SIG would be identical to that of Schema I. This corresponds to our natural intuition that these two schemas have the same "information capacity" and can express the same set of instances.

We conclude with a final note on modeling inheritance. Many authors have stressed the importance of modeling generalization and specialization when dealing with semantic heterogeneity. Specialization expresses the constraint that one set of instances or objects is a subset of another. Generalization expresses the constraint that one set is the (disjoint) union of two or more other sets. Our approach deals naturally with such constructs using annotated binary relations that may possibly be constrained to be a subset of the identity relation. Generalization is modeled as a total, surjective, injective function (a bijection) between a node and the union of other nodes. Specialization is modeled as a total, injective function from one node into another.

Specialization is often modeled using inclusion dependencies within the relational model. While space prohibits the development of a full example using generalization or specialization, the stock example may be simply extended with a few inclusion dependencies. In Schema II of Figure 1, a two-way inclusion dependency between the date columns in each of the three relational tables would ensure that all companies have stock prices recorded for the same dates. Within the schema intension, this information could be expressed by an annotated binary relation and combined with the functional dependency information to produce the total, surjective, function: $f : C \times D \twoheadrightarrow P$. It is these inclusion dependencies that are required to ensure that the second and third schemas have the same information capacity.

4 Future Work

We are currently expanding the use of this foundation in developing a sound, comprehensive integration methodology. Specifically, we are examining rules and heuristics that are appropriate for the derivation of schema intensions from existing schema extensions and ways in which these rules may be combined with interactive tools. We believe that the use of schema intensions offers an opportunity for increased automation and formalization of schema integration. To meet this expectation, the derivation of schema intensions must be based on a formal notion of equivalence. We are exploring the application of existing work on relative information capacity to schema intensions [4].

Once schema intensions are derived, formal reasoning about their equivalence is possible. We are examining whether a set of inference rules over schema intensions that is not only sound (produces logically valid inferences) but complete (produces all inferences that are sound) may be developed. We have shown this problem to be incomplete in general [7]. However, there are interesting subsets of schema intensions (of practical importance) for which a complete set of inference rules may be given. These rules may be used to formally establish the equivalence of schema intensions.

References

[1] T. Barsalou and D. Gangopadhyay. M(DM): An Open Framework for Interoperation of Multimodel Multidatabase Systems. In *Data Eng.*, pages 218–227, 1992.

[2] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, Dec. 1986.

[3] J. Biskup and B. Convent. A Formal View Integration Method. In *SIGMOD*, pages 398–407, 1986.

[4] R. Hull. Relative Information Capacity of Simple Relational Database Schemata. *SIAM Journal of Computing*, 15(3):856–886, Aug. 1986.

[5] R. Krishnamurthy, W. Litwin, and W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. In *SIGMOD*, pages 40–49, 1991.

[6] J. Larson, S. B. Navathe, and R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Trans. on Software Eng.*, 15(4):449–463, Apr. 1989.

[7] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. Foundations of Schema Translation. *Submitted for publication*, 1992.

[8] K. A. Ross. Relations with Relation Names as Arguments: Algebra and Calculus. In *PODS*, pages 346–353, 1992.