Inferring Complex Semantic Mappings Between Relational Tables and Ontologies from Simple Correspondences

Yuan An[1], Alex Borgida[2], and John Mylopoulos[1]

[1] University of Toronto, Canada
{yuana,jm}@cs.toronto.edu
[2] Rutgers University, USA
borgida@cs.rutgers.edu

Abstract. There are many problems requiring a semantic account of a database schema. At its best, such an account consists of mapping formulas between the schema and a formal conceptual model or ontology (CM) of the domain. This paper describes the underlying principles, algorithms, and a prototype of a tool which infers such semantic mappings when given *simple correspondences* from table columns in a relational schema and datatype properties of classes in an ontology. Although the algorithm presented is necessarily heuristic, we offer formal results stating that the answers returned are "correct" for relational schemas designed according to standard Entity-Relationship techniques. We also report on experience in using the tool with public domain schemas and ontologies.

1 Introduction and Motivation

A number of important database problems have been shown to have improved solutions by using a conceptual model or an ontology (CM) to provide he *precise semantics* of the database schema. These include federated databases, data warehousing [1], and information integration through mediated schemas [7]. (See survey [15].) Since much information on the web is generated from databases (the "deep web"), the recent call for a Semantic Web, which requires a connection between web content and ontologies, provides additional motivation for the problem of associating semantics with data (e.g., [6]). In almost all of these cases semantics of the data is captured by some kind of *semantic mapping* between the database schema and the CM. Although sometimes the mapping is just a *simple* association from terms to terms, in other cases what is required is a *complex* formula, often expressed in logic or a query language.

For example, in both the Information Manifold data integration system presented in [7] and the study of data integration in data warehousing presented in [1], Horn formulas in the form $T(\overline{X})$:- $\Phi(\overline{X}, \overline{Y})$ are used to connect a relational data source to a CM described by some Description Logic, where $T(\overline{X})$ is a single predicate representing a table in the relational data source and $\Phi(\overline{X}, \overline{Y})$ is a conjunctive formula over the predicates representing the concepts and relationships in the CM. In the literature, such a formalism is called local-as-view (LAV).

So far, it has been assumed that *humans* specify the mapping formulas – a difficult, time-consuming and error-prone task. In this paper, we propose a tool that assists users

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1152–1169, 2005.

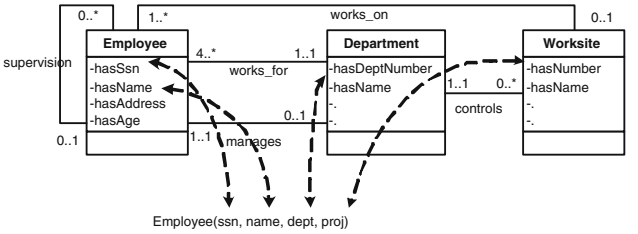

Fig. 1. Relational table, Ontology, and Correspondences

in specifying LAV mapping formulas between relational databases and ontologies. Intuitively, it is much easier for users to draw the *simple correspondences* from the columns of the tables in the database to datatype properties of classes in the ontology – manually or through some existing schema matching tools (e.g., [3, 13]) – than to compose the logic formulas. Given the set of correspondences and following the LAV formalism, the tool is expected to reason about the database schema and the ontology, and to generate a ranked list of candidate Horn formulas for each table in the relational database. Ideally, one of the formulas is the right one capturing the user's intention underlying the specified correspondences. The following example illustrates the input/out behavior of the tool we seek.

Example 1. An ontology contains concepts (classes), attributes of concepts (datatype properties of classes), and relationships between concepts (object properties of classes). Graphically, we use the UML notations to represent the above information. Given the ontology in Figure 1 and a relational table *Employee(ssn, name, dept, proj)* with key ssn, a user could draw the simple correspondences as the arrowed dash-lines shown in Figure 1. Using prefixes $\mathcal{T}$ and $\mathcal{O}$ to distinguish predicates in the relational schema and the ontology, we represent the correspondences as follows:

$\mathcal{T} : Employee.ssn \leftrightsquigarrow \mathcal{O} : Employee.hasSsn$
$\mathcal{T} : Employee.name \leftrightsquigarrow \mathcal{O} : Employee.hasName$
$\mathcal{T} : Employee.dept \leftrightsquigarrow \mathcal{O} : Department.hasDeptNumber$
$\mathcal{T} : Employee.proj \leftrightsquigarrow \mathcal{O} : Worksite.hasNumber$

Given the above input, we may expect the tool generate a mapping formula of the form

$\mathcal{T}$:Employee(ssn, name, dept, proj) :-
$\mathcal{O}$:Employee(x_1), $\mathcal{O}$:hasSsn(x_1,ssn), $\mathcal{O}$:hasName(x_1,name), $\mathcal{O}$:Department(x_2),
$\mathcal{O}$:works_for(x_1,x_2), $\mathcal{O}$:hasDeptNumber(x_2,dept), $\mathcal{O}$:Worksite(x_3), $\mathcal{O}$:works_on(x_1,x_3),
$\mathcal{O}$:hasNumber(x_3,proj). □

An intuitive and naive solution (inspired by early work of Quillian in [12]) gives rise to finding the minimum spanning trees or Steiner trees[1] among the classes that have datatype properties corresponding to table columns and encoding the trees into logic formulas. However, the problem is that a spanning/Steiner tree may not match

[1] A Steiner tree for set M of nodes in graph G is a minimum spanning tree of M that contains nodes of G which are not in M.

the semantics of the given table due to their constraints. For example, consider the relational table $Project(\underline{name}, supervisor)$, with $name$ as its key and corresponding to $\mathcal{O}$:$Worksite.hasName$, plus $supervisor$ corresponding to $\mathcal{O}$:$Employee.hasSsn$ in Figure 1. The minimum spanning tree consisting of $Worksite$, $Employee$, and the edge `works_on` does not match the semantics of table $Project$ because there are multiple $Employee$s working on a $Worksite$. In this paper, we turn to a database design process to uncover the connections between the constraints in relational schemas and ontologies. In contrast to the graph theoretic results which show that there might be too many minimum spanning/Steiner trees between a fixed set of nodes (for example, there are already 5 minimum spanning trees among *Employee, Department*, and *Worksite* in the very simple graph in Figure 1, considering each edge has the same weight,) we propose to generate a limited number of "reasonable" trees and formulas.

Our approach is directly inspired by the Clio project [10, 11], which developed a successful tool that infers mappings from one set of relational tables and/or XML documents to another, given just a set of correspondences between their respective attributes. Without going into further details at this point, we summarize the contributions which we feel are being made here:

- The paper identifies a new version of the data mapping problem: that of *inferring* complex formulas expressing the semantic mapping between relational database schemas and ontologies from simple correspondences.
- We propose an algorithm to find a "reasonable" tree connection in the ontology graph. The algorithm is enhanced to take into account information about the schema (key and foreign key structure), the ontology (cardinality restrictions), and standard database schema design guidelines.
- To gain theoretical confidence, we describe formal results which state that if the schema was designed from a CM using techniques well-known in the Entity Relationship literature (which provide a natural semantic mapping for each table), then the tool will report essentially all and only the appropriate semantics. This shows that our heuristics are not just shots in the dark: in the case when the ontology has no extraneous material, and when a table's schema has not been denormalized, the algorithm will produce good results.
- To test the effectiveness and usefulness of the algorithm in practice, we implemented the algorithm in a prototype tool and applied it to a variety of database schemas and ontologies. Our experience has shown that the user effort in specifying complex mappings by using the tool is significantly less than that by manually writing formulas from scratch.

The rest of the paper is structured as follows. Section 2 discusses related work, and Section 3 presents the necessary background and notation. Section 4 describes an intuitive progression of ideas underlying our approach, while Section 5 provides the mapping inference algorithm. In Section 6 we report on the prototype implementation of these ideas and experience with the prototype. Finally, Section 7 concludes and discusses future work.

2 Related Work

As mentioned earlier, the Clio tool [10, 11] discovers formal queries describing how target schemas can be populated with data from source schemas. The present work could be viewed as extending this to the case when the source schema is a relational database, while the target is a ontology. For example, in Example 1, if one viewed the ontology as a relational schema made of unary tables, e.g., $Employee(x_1)$, $Department(x_2)$, binary tables, e.g., $hasSsn(x'_1, ssn)$, $hasDeptNumber(x'_2, dept)$, $works\_for(x''_1, x''_2)$, and foreign key constraints, e.g., x'_1 and x''_1 referencing x_1, x'_2 and x''_2 referencing x_2, where x_i, x'_i, x''_i $(i = 1, 2)$ are object identifiers available in the ontology, one could in fact try to apply directly the Clio algorithm to it, pushing it beyond its intended application domain. The desired mapping formula from Example 1 would not be produced for several reasons: (i) Clio [11] does not make a so-called logical relation connecting $hasSsn(x'_1, ssn)$ and $hasDeptNumber(x'_2, dept)$, since the chase algorithm of Clio only follows foreign key references *out* of tables. Specifically, there would be three separate logical relations, i.e., $Employee(x_1) \bowtie_{x_1=x'_1} hasSsn(x'_1, ssn)$, $Department(x_2) \bowtie_{x_2=x'_2} hasDeptNumber(x'_2, dept)$, and $works\_for(x''_1, x''_2) \bowtie_{x''_1=x_1} Employee(x_1) \bowtie_{x''_2=x_2} Department(x_2)$. (ii) The fact that *ssn* is a key in the table $\mathcal{T}$:$Employee$, leads us to prefer (see Section 4) a many-to-one relationship, such as *works_for*, over some many-to-many relationship which could have been part of the ontology (e.g., $\mathcal{O}$*:previouslyWorkedFor*); Clio does not differentiate the two. So the work to be presented here analyzes the key structure of the tables and the semantics of relationships (cardinality, IsA) to eliminate *unreasonable* options that arise in mapping to ontologies.

The problem of *data reverse engineering* is to extract a CM, for example, an ER diagram, from a database schema. Sophisticated algorithms and approaches to this have appeared in the literature over the years (e.g., [8, 5]). The major difference between data reverse engineering and our work is that we are given an existing ontology, and want to interpret a legacy relational schema in terms of it, whereas data reverse engineering aims to construct a new ontology.

Schema matching (e.g., [3, 13]) identifies semantic relations between schema elements based on their names, data types, constraints, and schema structures. The primary goal is to find the one-to-one simple correspondences which are part of the input for our mapping inference algorithms.

3 Formal Preliminaries

For an ontology, we do not restrict ourselves to any particular ontology language in this paper. Instead, we use a generic conceptual modeling language (CML), which contains *common* aspects of most semantic data models, UML, ontology languages such as OWL, and description logics. In the sequel, we use CM to denote an ontology prescribed by the generic CML. Specifically, the language allows the representation of *classes/concepts* (unary predicates over individuals), *object properties/relationships* (binary predicates relating individuals), and *datatype properties/attributes* (binary predicates relating individuals with values such as integers and strings); attributes are single valued in this paper. Concepts are organized in the familiar **is-a** hierarchy. Object properties, and their inverses (which are always present), are subject to constraints such

as specification of domain and range, plus the familiar cardinality constraints, which here allow 1 as lower bounds (called *total* relationships), and 1 as upper bounds (called *functional* relationships). We shall represent a given CM using a directed and labeled *ontology graph*, which has concept nodes labeled with concept names C, and edges labeled with object properties p; for each such p, there is an edge for the inverse relationship, referred to as p^-. For each attribute f of concept C, we create a separate attribute node denoted as $N_{f,C}$, whose label is f, and with edge labeled f from node C to $N_{f,C}$.[2] For the sake of simplicity, we sometimes use UML notations, as in Figure 1, to represent the ontology graph. Note that in such a diagram, instead drawing separate attribute nodes, we place the attributes inside the rectangle nodes. Readers should not be confused by this compact representation.

If p is a relationship between concepts C and D (or object property having domain C and range D), we propose to write in text as C ---p--- D (If the relationship p is functional, we write C ---p->-- D.) For expressive CMLs such as OWL, we may also connect C to D by p if we find an existential restriction stating that each instance of C is related to *some* or *all* instance of D by p.

For relational databases, we assume the reader is familiar with standard notions as presented in [14], for example. We will use the notation $T[\underline{K}, Y]$ to represent a relational table T with columns KY, and key K. If necessary, we will refer to the individual columns in Y using $Y[1], Y[2], \ldots$, and use XY as concatenation. Our notational convention is that single column names are either indexed or appear in lower-case. Given a table such as T above, we use the notation key(T), nonkey(T) and columns(T) to refer to K, Y and KY respectively. (Note that we use the terms "table" and "column" when talking about relational schemas, reserving "relation(ship)" and "attribute" for aspects of the CM.) A foreign key (fk) in T is a set of columns F that *references* table T', and imposes a constraint that the projection of T on F is a subset of the projection of T' on key(T').

In this paper, a *correspondence* $T.c \rightsquigarrow D.f$ will relate column c of table T to attribute f of concept D. Since our algorithms deal with ontology graphs, formally a correspondence L will be a mathematical relation $L(T, c, D, f, N_{f,D})$, where the first two arguments determine unique values for the last three.

Finally, we use Horn-clauses in the form $T(X)$:- $\Phi(X, Y)$, as described in Introduction, to represent *semantic mappings*, where T is a table with columns X (which become arguments to its predicate), and Φ is a conjunctive formula over predicates representing the CM, with Y existentially quantified as usual.

4 Principles of Mapping Inference

We begin with the set of *concept nodes*, M, such that for each node in M some of the attribute nodes connected to it are corresponded by some of the columns of a table, and M contains all of the nodes singled out by all of the correspondences from the columns of the table. We assume that the correspondences have been specified by users. To seek LAV mapping, it is sufficient to only focus on the connections among nodes in M

[2] Unless ambiguity arises, we will use "node C", when we mean "concept node labeled C".

by stripping off the attribute nodes[3]. Note that attribute nodes, which we can attach them back at any time, are important when encoding trees into formulas for proving the formal results. The primary principle of our mapping inference algorithm is to look for *shortest* "reasonable" trees connecting nodes in M. In the sequel, we will call such a tree *semantic tree*.

As mentioned before, the naive solution of finding min-spanning trees or Steiner trees does not give us good results. The semantic tree we seek is not only shortest but "reasonable". Although the "reasonableness" is vague at this moment, we will lay out some principles according to the semantics carried by the relational schemas and ontologies; and we will show that our principles have a solid foundation that the "reasonableness" can be formally proved in a very strict but useful setting.

Consider the case when $T[\underline{c}, b]$ is a table with key c, corresponding to an attribute f on concept C, and b is a foreign key corresponding to an attribute e on concept B. Then for each value of c (and hence instance of C), T associates at most one value of b (instance of B). Hence the semantic mapping for T should be some formula that acts as a function from its first to its second argument. The semantic trees for such formulas look like functional edges, and hence should be preferred. For example, given table $Dep[\underline{dept}, ssn, \ldots]$, and correspondences which link the two named columns to $hasDeptNumber$ and $hasSsn$ in Figure 1, respectively, the proper semantic tree uses $\texttt{manages}^-$ (i.e., `hasManager`) rather than $\texttt{works\_for}^-$ (i.e., `hasWorkers`).

Conversely, for table $T'[\underline{c, b}]$, an edge that is functional from C to B, or from B to C, is likely not to reflect a proper semantics since it would mean that the key chosen for T' is actually a super-key – an unlikely error. (In our example, consider a table $T[\underline{ssn, dept}, \ldots]$, where both named columns are foreign keys.) To deal with such problems, an algorithm should work in two stages: first connecting the concepts corresponding to key columns into somehow a *skeleton tree*, then connecting the rest nodes corresponding to other columns to the skeleton by, preferably, functional edges.

Most importantly, we must deal with the assumption that the relational schema and the CM were developed independently, which implies that not all parts of the CM are reflected in the database schema and vice versa. This complicates things, since in building the semantic tree we may need to go through additional nodes, which end up not being corresponded by any columns in the relational schema. For example, Consider again the $Project(\underline{name}, supervisor)$ table and its correspondences mentioned in Introduction. Instead of the edge `works_on`, we prefer the *functional path* $\texttt{controls}^-.\texttt{manages}^-$ (i.e., `controlledBy` followed by `hasManager`), passing through node *Department*. Similar situations arise when the CM contains detailed *aggregation* hierarchies (e.g., *city* part-of *township* part-of *county* part-of *state*), which are abstracted in the database (e.g., a table with columns for *city* and *state* only).

We have chosen to flesh out the above principles in a systematic manner by considering the behavior of our proposed algorithm on relational schemas designed from Entity Relationship diagrams — a topic widely covered in even undergraduate database courses [14]. (We call this *er2rel schema design*.) One benefit of this approach will be to allow us to prove that our algorithm, though heuristic in general, is in some sense

[3] In the sequel, we will say "a concept corresponded by some columns of a table" without mentioning its attributes.

"correct" for a certain class of schemas. Of course, in practice such schemas may be "denormalized" in order to improve efficiency, and, as we mentioned, only parts of the CM are realized in the database. We emphasize that our algorithm uses the general principles enunciated above even in such cases, with relatively good results in practice.

To reduce the complexity of the algorithms which is inherently a tree enumeration, and the size of the answer set, we modify the graph by collapsing multiple edges between nodes E and F, labeled $p_1, p_2, \ldots$ say, into a single edge labeled $'p_1; p_2; \ldots'$ The idea is that it will be up to the user to choose between the alternative labels after the final results have been presented by the tool, though the system may offer suggestions, based on additional information, such as heuristics concerning the identifiers labeling tables and columns, and their relationship to property names.

5 Mapping Inference Algorithms

As stated before, the algorithm is based on the relational database design methodology from ER models. We will introduce the details of the algorithm in a gradual manner, by repeatedly adding features of an ER model that appear as part of the CM. We assume that the reader is familiar with basics of ER modeling and database design [14], though we summarize the ideas.

5.1 An Initial Subset of ER Notions

We start with a subset of ER that contains the notions such as *entity set* E (called just "entity" here), with attributes referred as $\mathsf{attribs}(E)$, and *binary relationship set*. In order to facilitate the statement of correspondences and theorems, we assume in this section that attributes in the CM have globally unique names. (Our implemented tool does not make this assumption.) An entity is represented as a concept/class in our CM. A binary relationship set corresponds to two relationships in our CM, one for each direction, though only one is mapped to a table. Such a relationship will be called *many-many* if neither it nor its inverse is functional. A *strong entity* S has some attributes that act as identifier. We shall refer to these using $\mathsf{unique}(S)$ when describing the rules of schema design. A *weak entity* W has instead $\mathsf{localUnique}(W)$ attributes, plus a functional total binary relationship p (denoted as $\mathsf{idRel}(W)$) to an identifying owner entity (denoted as $\mathsf{idOwn}(W)$).

Note that information about general identification cannot be represented in even highly expressive languages such as OWL. So functions like unique are only used while describing the er2rel mapping, and are not assumed to be available during semantic inference. The er2rel design methodology (we follow mostly [8, 14]) is defined by two components: To begin with, Table 1 specifies a mapping $\tau(O)$ returning a relational table schema for every CM component O, where O is either a concept/entity or a binary relationship. In this subsection, we assume that no pair of concepts is related by more than one relationship, and that there are no so-called "recursive" relationships relating an entity to itself. (We deal with these in Section 5.3.)

In addition to the schema (columns, key, fk's), Table 1 also associates with a relational table $T[V]$ a number of additional notions:

Table 1. er2rel Design Mapping

ER Model object O	Relational Table τ(O)	
Strong Entity S	*columns:*	X
	primary key:	K
Let X=attribs(S)	*fk's:*	none
Let K=unique(S)	*anchor:*	S
	semantics:	$T(X)$:- $S(y)$,hasAttribs(y, X).
	identifier:	identify$_S(y, K)$:- $S(y)$,hasAttribs(y, K).
Weak Entity W	*columns:*	ZX
let	*primary key:*	UX
E = idOwn(W)	*fk's:*	X
P = idrel(W)	*anchor:*	W
Z=attribs(W)	*semantics:*	$T(X, U, V)$:- $W(y)$, hasAttribs(y, Z), $E(w)$,$P(y, w)$,
X = key($\tau(E)$)		identify$_E(w, X)$.
U =localUnique(W)	*identifier:*	identify$_W(y, UX)$:- $W(y)$,$E(w)$, $P(y, w)$, hasAttribs(y, U),
$V = Z - U$		identify$_E(w, X)$.
Functional Relationship F	*columns:*	X_1X_2
	primary key:	X_1
E_1 --F-> E_2	*fk's:*	X_i references $\tau(E_i)$,
let X_i = key($\tau(E_i)$)	*anchor:*	E_1
for $i = 1, 2$	*semantics:*	$T(X_1, X_2)$:- $E_1(y_1)$,identify$_{E_1}(y_1, X_1)$, $F(y_1, y_2)$, $E_2(y_2)$,
		identify$_{E_2}(y2, X_2)$.
Many-many Relationship M	*columns:*	X_1X_2
	primary key:	X_1X_2
E_1 --M-- E_2	*fk's:*	X_i references $\tau(E_i)$,
let X_i = key($\tau(E_i)$)	*semantics:*	$T(X_1, X_2)$:- $E_1(y_1)$,identify$_{E_1}(y_1, X_1)$, $M(y_1, y_2)$,$E_2(y_2)$,
for $i = 1, 2$		identify$_{E_2}(y2, X_2)$.

– an *anchor*, which is the central object in the CM from which T is derived, and which is useful in explaining our algorithm (it will be the root of the semantic tree);
– a formula for the semantic mapping for the table, expressed as a Horn formula with head $T(V)$ (this is what our algorithm should be recovering); in the body of the Horn formula, the function hasAttribs(x, Y) returns conjuncts $attr_j(x, Y[j])$ for the individual columns Y[1], Y[2],... in Y, where $attr_j$ is the attribute name corresponded by column $Y[j]$.
– the formula for a predicate identify$_C(x, Y)$, showing how object x in (strong or weak) entity C can be identified by values in Y[4].

Note that τ is defined recursively, and will only terminate if there are no "cycles" in the CM (see [8] for definition of cycles in ER).

The er2rel methodology also suggests that the schema generated using τ can be modified by (repeatedly) *merging* into the table T_0 of an entity E the table T_1 of some functional relationship involving the same entity E (which has a foreign key reference to T_0). If the semantics of T_0 is $T_0(K, V)$:- $\phi(K, V)$, and of T_1 is $T_1(K, W)$

[4] This is needed in addition to hasAttribs, because weak entities have identifying values spread over several concepts.

:- $\psi(K, W)$, then the semantics of table T=merge(T_0,T_1) is, to a first approximation, $T(K, V, W)$:- $\phi(K, V), \psi(K, W)$. And the anchor of T is the entity E.

Please note that one conceptual model may result in several different relational schemas, since there are choices in which direction a one-to-one relationship is encoded (which entity acts as a key), and how tables are merged. Note also that the resulting schema is in Boyce-Codd Normal Form, if we assume that the only functional dependencies are those that can be deduced from the ER schema (as expressed in FOL).

Now we turn to the algorithm for finding the semantic trees between nodes in the set M singled out by the correspondences from columns of a table. As mentioned in the previous section, because the keys of a table functionally determine the rest of the columns, the algorithm for finding the semantic trees works in several steps:

1. Determine a skeleton tree connecting the concepts corresponding to key columns; also determine, if possible, a unique anchor for this tree.
2. Link the concepts corresponding to non-key columns using shortest functional paths to the skeleton anchor.
3. Link any unaccounted-for concepts corresponding to some other columns by arbitrary shortest paths to the tree.

More specifically, the main function, getTree(T,L), will infer the semantics of table T, given correspondence L, by returning an semantic tree S. Encoding S into formula yields the conjunctive formula defining the semantics of table T.

Function getTree(T,L)
input: table T, correspondences L for columns(T)
output: set of semantic trees [5]
steps:

1. Let L_k be the subset of L containing correspondences from key(T);
   compute (S', Anc')=getSkeleton(T,L_k).
2. If onc(nonkey(T))[6]- onc(key(T)) is empty, then return (S', Anc'). */*if all columns correspond to the same set of concepts as the key does, then return the skeleton tree.*/*
3. For each foreign key F_i in nonkey(T) referencing $T_i(K_i)$:
   let $L^i_k = \{T_i.K_i \rightsquigarrow L(T, F_i)\}$, and compute (Ss''_i, Anc''_i)= getSkeleton(T_i,L^i_k). */*recall that the function $L(T, F_i)$ is derived from a correspondence $L(T, F_i, D, f, N_{f,D})$ such that it gives a concept D and its attribute f ($N_{f,D}$ is the attribute node in the ontology graph.)*/*
   find π_i=shortest functional path from Anc' to Anc''_i; let $S =$ combine[7]$(S', \pi_i, \{Ss''_i\})$.
4. For each column c in nonkey(T) that is not part of an fk, let $N =$ onc(c); find π=shortest functional path from Anc' to N; update $S :=$ combine(S, π).
5. In all cases above asking for functional paths, use a shortest path if a functional one does not exist.
6. Return S.

[5] To make the description simpler, at times we will not explicitly account for the possibility of multiple answers. Every function is extended to set arguments by element-wise application of the function to set members.

[6] onc(X) is the function which gets the set M of concepts corresponded by the columns X.

[7] Function combine merges edges of trees into a larger tree.

The function getTree(T, L) makes calls to function getSkeleton on T and other tables referenced by fks in T, in order to get a set of (skeleton tree, anchor)-pairs, which have the property that in the case of er2rel designs, if the anchor returned is concept C, then the encoding of the skeleton tree is the formula for identify$_C$.

Function getSkeleton(T,L)
input: table T, correspondences L for key(T)
output: a set of (skeleton tree, anchor) pairs
steps:
Suppose key(T) contains fks F_1,...,F_n referencing tables $T_1(K_1)$,..,$T_n(K_n)$;

1. If $n \leq 1$ and onc(key(T)) is just a singleton set $\{C\}$, then return $(C, \{C\})$.[8] /*Likely a strong entity: the base case.*/
2. Else, let L_i=$\{T_i.K_i \rightsquigarrow L(T, F_i)\}$/*translate corresp's thru fk reference*/;
   compute (Ss_i, Anc_i) = getSkeleton(T_i, L_i).
   (a) If key(T) = F_1, then return (Ss_1, Anc_1). /*functional relationship of weak entities.*/
   (b) If key(T)=F_1A, where columns A are not in any foreign key of T then /*possibly a weak entity*/
      i. if $Anc_1 = \{N_1\}$ and onc(A) = $\{N\}$ such that there is a total functional path π from N to N_1, then return (combine(π, Ss_1), $\{N\}$). /*N is a weak entity.*/
   (c) Else supposing key(T) has additional non-fk columns $A[1], \ldots A[m]$, ($m \geq 0$); let Ns=$\{Anc_i\} \cup \{$onc$(A[j]), j = 1, .., m\}$, and find skeleton tree Ss' connecting the nodes in Ns, where any pair of nodes in Ns is connected by a many-many path; return (combine(Ss', $\{Ss_j\}$), Ns). /*dealing with the many-to-many binary relationships; also the default action for unaccounted-for tables, e.g., cannot find an identifying relation from a weak entity to the supposed owner entity. No unique anchor exists.*/

In order for getSkeleton to terminate, it is necessary that there be no cycles in fk references in the schema. Such cycles (which may have been added to represent additional integrity constraints, such as the the fact that an association is total) can be eliminated from a schema by replacing the tables involved with their outer join over the key. getSkeleton deals with strong entities and their functional relationships in step (1), with weak entities in step (2.b.i), and so far, with functional relationships of weak entities in (2.a). In addition to being a catch-all, step (2.c) deals with tables representing many-many relationships (which in this section have key $K = F_1F_2$), by finding anchors for the ends of the relationship, and then connecting them with paths that are not functional, even when every edge is reversed.

To get the logic formula from a tree based on correspondence L, we provide the procedure encodeTree(S, L) below, which basically assigns variables to nodes, and connects them using edge labels as predicates.

Function encodeTree(S,L)
input: subtree S of ontology graph, correspondences L from table columns to attributes of concept nodes in S.
output: variable name generated for root of S, and conjunctive formula for the tree.
steps: Suppose N is the root of S. Let $\Psi = \{\}$.

[8] Both here and elsewhere, when a concept C is added to a tree, so are edges and nodes for C's attributes that appear in L.

1. if N is an attribute node with label f, find d such that $L(\_, d, \_, f, N) = true$, return$(d, true)$. */*for leaves of the tree, which are attribute nodes, return the corresponding column name as the variable and an empty formula.*/*
2. if N is a concept node with label C, then introduce new variable x; add conjunct $C(x)$ to Ψ;
   for each edge p_i from N to N_i */*recursively get the entire formula.*/*
      let S_i be the subtree rooted at N_i,
      let $(v_i, \phi_i(Z_i))$=encodeTree(S_i, L),
      add conjuncts $p_i(x, v_i) \wedge \phi_i(Z_i)$ to Ψ;
return (x, Ψ).

To specify the properties of the algorithm, we now suppose that the correspondences L be the identity mappings from attribute names to table columns. The interesting property of getSkeleton is that if $T = \tau(C)$ according to the er2rel rules in Table 1, where C corresponds to a (strong or weak) entity, then getSkeleton returns (S, Anc), where $Anc = C$ as anchor, and encodeTree(S, L) is logically equivalent to identify$_C$. Similar property exists for $T = \tau(p)$, where p is a functional relationship originating from concept C, in which case its key looks just like an entity key. We now state the desirable properties more formally. Since the precise statement of theorems (and algorithms) is quite lengthy and requires a lot of minute details for which we do not have room here, we express the results as "approximately phrased" propositions. First, getTree finds the desired semantic mapping, in the sense that

Proposition 1. *Let table T be part of a relational schema obtained by er2rel derivation from conceptual model $\mathcal{E}$. Then some tree S returned by* getTree(T, L) *has the property that the formula returned by* encodeTree(S, L) *is logically equivalent to the semantics assigned to T by the er2rel design.*

Note that this "completeness" result is non-trivial, since, as explained earlier, it would not be satisfied by the current Clio algorithm [11], if applied blindly to $\mathcal{E}$ viewed as a relational schema with unary and binary tables. Since getTree may return multiple answers, the following converse "soundness" result is significant

Proposition 2. *If S' is any tree returned by* getTree(T, L)*, with T as above, then the formula returned by* encodeTree(S', L) *represents the semantics of* some *table T' derivable by er2rel design from $\mathcal{E}$, where T' is isomorphic*[9] *to T.*

Such a result would not hold of an algorithm which returns only minimal spanning trees, for example.

We would like to point out that the above algorithm performs reasonably on some non-standard designs as well. For example, consider the relational table $T(\underline{personName}, cityName, countryName)$, where the columns correspond to, respectively, attributes $pname$, $cname$, and $crname$ of concepts $Person, City$ and $Country$ in a CM. If the CM contains a path such that `Person` `-- bornIn ->-` `City` `-- locatedIn ->-` `Country`, then the above table, which is not in 3NF and was not obtained using

[9] Informally, two tables are isomorphic if there is a bijection between their columns which preserves key and foreign key structure.

er2rel design (which would have required a table for *City*), would still get the proper semantics:

T(personName,cityName,countryName) :-
Person(x_1), City(x_2),Country(x_3), bornIn(x_1,x_2), locatedIn(x_2,x_3),
pname(x_1,personName), cname(x_2,cityName),crname(x_3,countryName).

If on the other hand, there was a shorter functional path from $Person$ to $Country$, say an edge labeled `citizenOf`, then the mapping suggested would have been:

T(personName, cityName, countryName) :-
Person(x_1), City(x_2), Country(x_3), bornIn (x_1,x_2),citizenOf(x_1,x_3), ...

which corresponds to the er2rel design. Moreover, had `citizenOf` not been functional, then once again the semantics produced by the algorithm would correspond to the non-3NF interpretation, which is reasonable since the table, having only $personName$ as key, could not store multiple country names for a person.

5.2 Reified Relationships

It is desirable to also have n-ary relationship sets connecting entities, and to allow relationship sets to have attributes (called "association classes" in UML). Unfortunately, these features are not directly supported in most CMLs, such as OWL, which only have binary relationships. Such notions must instead be represented by *"reified relationships"* [2] (we use an annotation * to indicate the reified relationships in a diagram): concepts whose instances represent tuples, connected by so-called "roles" to the tuple elements. So, if *Buys* relates *Person*, *Shop* and *Product*, through roles *buyer*, *source* and *object*, then these are explicitly represented as (functional) binary associations, as in Figure 2. And a relationship attribute, such as when the buying occurred, becomes an attribute of the *Buys* concept, such as *whenBought*.

Unfortunately, reified relationships cannot be distinguished reliably from ordinary entities in normal CMLs on purely formal, syntactic grounds, yet they need to be treated in special ways during recovery. For this reason we assume that they can be distinguished on *ontological grounds*. For example, in Dolce [4], they are subclasses of top-level concepts *Quality* and *Perdurant/Event*. For a reified relationship R, we use functions roles(R) and attribs(R) to retrieve the appropriate (binary) properties.

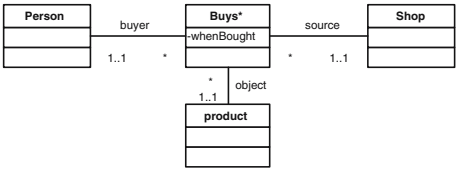

Fig. 2. N-ary Relationship Reified

The design τ of relational tables for reified relationships is shown in Table 2. To discover the correct anchor for reified relationships and get the proper tree, we need to modify getSkeleton, by adding the following case between steps 2(b) and 2(c):

– If key(T)=$F_1F_2 \ldots F_n$ and there exist reified relationship R with n roles $r_1, \ldots, r_n$ pointing at the singleton nodes in $Anc_1, \ldots, Anc_n$ respectively,
then let $S =$ combine($\{r_j\}, \{Ss_j\}$), and return $(S, \{R\})$.

Table 2. er2rel Design for Reified Relationship

ER model object O	Relational Table $\tau(O)$	
Reified Relationship R	*columns:*	$ZX_1 \ldots X_n$
if $r_1, \ldots, r_n$ are roles of R	*primary key:*	$X_1 \ldots X_n$
let Z=attribs(R)	*fk's:*	$X_1, \ldots, X_n$
X_i=key($\tau(E_i)$)	*anchor:*	R
where E_i fills role r_i	*semantics:*	$T(ZX_1 \ldots X_n)$:- $R(y), E_i(w_i)$, hasAttribs(y, Z), $r_i(y, w_i)$, identify$_{E_i}(w_i, X_i), \ldots$
	identifier:	identify$_R(y, \ldots X_i \ldots)$:- $R(y), \ldots E_i(w_i)$, $r_i(y, w_i)$, identify$_{E_i}(w_i, X_i)$,...

The main change to getTree is to compensate for the fact that if getSkeleton finds a *reified* version of a many-many binary relationship, it will no longer look for an unreified one. So after step 1. we add

– if key(T) is the concatenation of two foreign keys F_1F_2, and nonkey(T) is empty, compute (Ss_1,Anc_1) and (Ss_2, Anc_2) as in step 2. of getSkeleton; then find ρ=shortest many-many path connecting Anc_1 to Anc_2;
return $(S') \cup$ (combine(ρ, Ss_1, Ss_2))

The previous version of getTree was set up so that with these modifications, attributes to reified relationships will be found properly, and the previous propositions continue to hold.

5.3 Replication

If we allow recursive relationships, or allow the merger of tables for different functional relationships connecting the same pair of concepts (e.g., `works_for` and `manages`), the mapping in Table 1 is incorrect because column names will be repeated in the multiple occurrences of the foreign keys. We will distinguish these (again, for ease of presentation) by adding superscripts as needed. For example, if *Person* is connected to itself by the *likes* property, then the table for *likes* will have schema $T[\underline{ssn^1}, \underline{ssn^2}]$.

During mapping discovery, such situations are signaled by the presence of multiple columns c and d of table T corresponding to the same attribute f of concept C. In such situations, the algorithm will first make a copy C_{copy} of node C in the ontology graph, as well as its attributes. C_{copy} participates in all the object relations C did, so edges must be added. After replication, we can set onc$(c) = C$ and onc$(d) = C_{copy}$, or onc$(d) = C$ and onc$(c) = C_{copy}$ (recall that onc(c) gets the concept corresponded by column c in the algorithm). This ambiguity is actually required: given a CM with *Person* and *likes* as above, a table $T[\underline{ssn^1}, \underline{ssn^2}]$ could have alternate semantics corresponding to *likes*, and its inverse, $likedBy$. (A different example would involve a table $T[\underline{ssn}, addr^1, addr^2]$, where *Person* is connected by two relationships, *home* and *office*, to concept *Building*, which has an *address* attribute.

The main modification needed to the getSkeleton and getTree algorithms is that no tree should contain both a functional edge D --- p ->-- C and its replicate

D --- p ->-- C_{copy}, (or several replicates), since a function has a single value, and hence the different columns of a tuple will end up having identical values: a clearly poor schema.

5.4 Addressing Class Specialization

The ability to represent subclass hierarchies, such as the one in Figure 3 is a hallmark of CMLs and modern so-called Extended ER (EER) modeling.

Almost all textbooks (e.g., [14]) describe two techniques for designing relational schemas in the presence of class hierarchies

1. Map each concept/class into a separate table following the standard er2rel rules. This approach requires two adjustments: First, subclasses must inherit identifier attributes from a single super-class, in order to be able to generate keys for their tables. Second, in the table created for an immediate subclass C' of class C, its key $\mathsf{key}(\tau(C'))$ should also be set to reference as a foreign key $\tau(C)$, as a way of maintaining inclusion constraints dictated by the is-a relationship.
2. Expand inheritance, so that *all* attributes and relations involving a class C appear on all its subclasses C'. Then generate tables as usual for the subclasses C', though not for C itself. This approach is used only when the subclasses cover the superclass.
   some researchers also suggest a third possibility:
3. "Collapse up" the information about subclasses into the table for the superclass. This can be viewed as the result of $\mathsf{merge}(T_C, T_{C'})$, where $T_C[K, A]$ and $T_{C'}[K, B]$ are the tables generated for C and its subclass C' according to technique (1.) above. In order for this design to be "correct", [8] requires that $T_{C'}$ not be the target of any foreign key references (hence not have any relationships mapped to tables), and that B be non-null (so that instances of C' can be distinguished from those of C).

The use of the key for the root class, together with inheritance and the use of foreign keys to also check inclusion constraints, make many tables highly ambiguous. For example, according to the above, table $T(\underline{ss\#}, crsId)$, with $ss\#$ as the key and a foreign key referencing T', could represent at least

(a) *Faculty teach Course*
(b) *Lecturer teach Course*
(c) *Lecturer coord Course*.

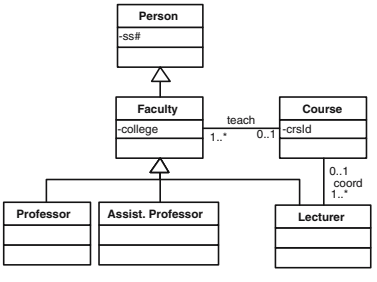

Fig. 3. Specialization Hierarchy

This is made combinatorially worse by the presence of multiple and deep hierarchies (e.g., imagine a parallel *Course* hierarchy), and the fact that not all ontology concepts are realized in the database schema, according to our scenario. For this reason, we have chosen to try to deal with some of the ambiguity relying on users, during the establishment of correspondences. Specifically, the user is supposed to provide a correspondence from column c to attribute f *on the lowest class whose instances provide data appearing in the column*. Therefore, in

the above example of table $T(\underline{ss\#}, crsId)$, *ss#* is made to correspond to *ssn* on *Faculty* in case (a), while in cases (b) and (c) it is made to correspond to *ss#* on *Lecturer*. This decision was also prompted by the CM manipulation tool that we are using, which automatically expands inheritance, so that *ss#* appears on all subclasses.

Under these circumstances, in order to capture designs (1.) and (2.) above, we do not need to modify our earlier algorithm in any way, if we first expand inheritance in the graph. So the graph would show `Lecturer -- teaches;coord ->- Course` in the above example, and *Lecturer* would have all the attributes of *Faculty*.

To handle design (3.), we can add to the graph an actual edge for the inverse of the **is-a** relation: a functional edge labeled *alsoA*, with lower-bound 0: `C --- alsoA ->- C'`, connecting superclass *C* to each of its subclasses *C'*. It is then sufficient to allow functional paths between concepts to consist of *alsoA* edges, in addition to the normal kind, in getTree.

5.5 Outer Joins

The observant reader has probably noticed that the definition of the semantic mapping for $T = \mathsf{merge}(T_E, T_p)$ was not quite correct: $T(\underline{K}, V, W) : -\phi(K, V), \psi(K, W)$ describes a join on K, rather than a left-outer join, which is what is required if p is a non-total relationship. In order to specify the equivalent of outer joins in a perspicuous manner, we will use conjuncts of the form $\lceil \mu(X, Y) \rceil^Y$, which will stand for the formula $\mu(X, Y) \vee (Y = null \wedge \neg \exists Z.\mu(X, Z))$, indicating that null should be used if there are no satisfying values for the variables Y. With this notation, the proper semantics for merge is $T(\underline{K}, V, W) : -\phi(K, V), \lceil \psi(K, W) \rceil^W$.

In order to obtain the correct formulas from trees, encodeTree needs to be modified so that when traversing a non-total edge p_i that is not part of the skeleton, in the second-to-last line of the algorithm we must allow for the possibility of v_i not existing.

Our formal results still hold under the replication, the treatment of specialization hierarchy, and the encoding of the merging of non-total functional relationships into outer joins.

6 Experience

So far, we have developed the mapping inference algorithm by investigating the connections between the semantic constraints in both relational models and ontologies. The theoretical results show that our algorithm will report the "right" semantics for schemas designed following the widely accepted design methodology. Nonetheless, it is crucial to test the algorithm in real-world schemas and ontologies to see its overall performance. To do this, we have implemented the mapping inference algorithm in our prototype system MAPONTO, and have applied it on a set of schemas and ontologies. In this section, we provide some evidence for the effectiveness and usefulness of the prototype tool by discussing the set of experiments and our experience.

Our test data were obtained from various sources, and we have ensured that the databases and ontologies were developed independently. The test data are listed in Table 3. They include the following databases: the Department of Computer Science

Table 3. Characteristics of Schemas and ontologies for the Experiments

Database Schema	Number of Tables	Number of Columns	Ontology	Number of Nodes	Number of Links
UTCS Department	8	32	Academic Department	62	1913
VLDB Conference	9	38	Academic Conference	27	143
DBLP Bibliography	5	27	Bibliographic Data	75	1178
OBSERVER Project	8	115	Bibliographic Data	75	1178
Country	6	18	CIA factbook	52	125

database in University of Toronto; the VLDB conference database; the DBLP computer science bibliography database; the COUNTRY database appearing in one of reverse engineering papers; and the test schemas in OBSERVER [9] project. For the ontologies, our test data include: the academic department ontology in the DAML library; the academic conference ontology from the SchemaWeb ontology repository; the bibliography ontology in the library of the Stanford's Ontolingua server; and the CIA factbook ontology. Ontologies are described in OWL. For each ontology, the number of links indicates the number of edges in the multi-graph resulted from object properties. We have made all these schemas and ontologies available on our web page: www.cs.toronto.edu/~yuana/research /maponto/relational/testData.html.

To evaluate our tool, we sought to understand whether the tool could produce the intended mapping formula if the simple correspondences were given. We were concerned about the number of formulas presented by the tool for users to sift through. Further, we wanted to know whether the tool was still useful if the correct formula was not generated. In this case, we expected that a user could easily debug a generated formula to reach the correct one instead of creating it from scratch. A summary of the experimental results are listed in Table 4 which shows the average size of each relational table schema in each database, the average number of candidates generated, and the average time for generating the candidates. Notice that the number of candidates is the number of semantic trees obtained by the algorithm. Also, a single edge of an semantic tree may represent the multiple edges between two nodes, collapsed using our $p; q$ abbreviation. If there are m edges in a semantic tree and each edge has n_i $i = 1, .., m$ original edges collapsed, then there are $\prod_i^m n_i$ original semantic trees. We show below a formula generated from such a collapsed semantic tree:

TaAssignment(courseName, studentName) :-
Course(x_1), GraduateStudent(x_2), **[hasTAs;takenBy]**(x_1,x_2),
workTitle(x_1,courseName), personName(x_2,studentName).

where, in the semantic tree, the node *Course* and the node *GraduateStudent* are connected by a single edge with label **hasTAs;takenBy** which represents two separate edges, *hasTAs* and *takenBy*.

Table 4 shows that the tool only present a few mapping formulas for users to examine. This is due in part to our compact representation of parallel edges between two nodes shown above. To measure the overall performance, we manually created the mapping formulas for all the 28 tables and compared them to the formulas generated by the tool. We observed that the tool produced correct formulas for 24 tables. It demonstrated

Table 4. Performance Summary for Generating Mappings from Relational Tables to Ontologies

Database Schema	Avg. Number of Cols/per table	Avg. Number of Candidates generated	Avg. Execution time(ms)
UTCS Department	4	4	279
VLDB Conference	5	1	54
DBLP Bibliography	6	3	113
OBSERVER Project	15	2	183
Country	3	1	36

that the tool is able to understand the semantics of many practical relational tables in terms of an independently developed ontology.

However, we wanted to know the usefulness of the tool. To evaluate this, we examined the generated formulas which were not the intended ones. For each such formula, we compared it to the manually created and correct one, and we used a very coarse measurement to record how much effort we had to spend to debug the generated formula in order to make it correct. Such a measurement only recorded the changes of predicate names in a formula. For example, the tool generated the following formula for the table *Student(name, office, position, email, phone, supervisor)*:

$$\textit{Student}(X_1),\ \textit{emailAddress}(X_1,\textit{email}),\ \textit{personName}(X_1,\textit{name}),\ \textit{Professor}(X_2),\ \textit{Institute}(X_3),\ \textit{head}(X_3,X_2),\ \textit{affiliatedOf}(X_3,X_1),\ \textit{personName}(X_2,\ \textit{supervisor})\ldots \quad (1)$$

If the intended semantics for the above table columns is:

$$\textit{Student}(X_1),\ \textit{emailAddress}(X_1,\textit{email}),\ \textit{personName}(X_1,\textit{name}),\ \textit{Professor}(X_2),\ \textit{GraduateStudent}(X_3),\ \textit{hasAdvisor}(X_3,X_2),\ \textit{isA}(X_3,X_1),\ \textit{personName}(X_2,\ \textit{supervisor})\ldots \quad (2)$$

then, one can change the three predicates $\textit{Institute}(X_3),\ \textit{head}(X_3,X_2),\ \textit{affiliatedOf}(X_3,X_1)$ in formula (1) to $\textit{GraduateStudent}(X_3),\ \textit{hasAdvisor}(X_3,X_2),\ \textit{isA}(X_3,X_1)$ instead of writing the entire formula (2) from scratch. Our experience working with the tool has shown that significant effort have been saved when building semantic mappings from tables to ontologies, because in most cases one only needed to change a relatively small number of predicates in an existing formula.

Tables 4 indicate that execution times were not significant, since, as predicted, the search for subtrees and paths took place in a relatively small neighborhood.

7 Conclusion and Future Work

Semantic mappings between relational database schemas and ontologies in the form of logic formulas play a critical role in realizing the semantic web as well as in many data sharing problems. We have proposed a solution to infer the LAV mapping formulas from simple correspondences, relying on information from the database schema (key and foreign key structure) and the ontology (cardinality restrictions, **is-a** hierarchies). Theoretically, our algorithm infers all and only the semantics implied by the ER-to-relational design if a table's schema follows ER design principles. In practice, our ex-

perience working with independently developed schemas and ontologies has shown that significant effort has been saved in specifying the LAV mapping formulas.

We are working towards disambiguation between multiple possible semantics by exploiting the following sources of information: first, a richer modeling language, supporting at least disjointness/coverage in **is-a** hierarchies, but also more complex axioms as in OWL ontologies; second, the use of the *data* stored in the relational tables whose semantics we are investigating. For example, queries may be used to check whether complex integrity constraints implied by the semantics of a concept/relationship fail to hold, thereby eliminating some candidate semantics.

Acknowledgments. We are most grateful to Renée Miller and Yannis Velegrakis for their clarifications concerning Clio, comments on our results, and encouragement. Remaining errors are, of course, our own.

References

1. D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data Integration in Data Warehousing. *J. of Coop. Info. Sys.*, 10(3):237–271, 2001.
2. M. Dahchour and A. Pirotte. The Semantics of Reifying n-ary Relationships as Classes. In *ICEIS'02*, pages 580–586, 2002.
3. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *SIGMOD'04*, pages 383–394, 2004.
4. A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening Ontologies with DOLCE. In *EKAW'02*, pages 166–181, 2002.
5. J.-L. Hainaut. *Database Reverse Engineering*. http:// citeseer.ist.psu.edu/ article/ hainaut98database.html, 1998.
6. S. Handschuh, S. Staab, and R. Volz. On Deep Annotation. In *Proc. WWW'03*, 2003.
7. A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *J. of Intelligent Info. Sys.*, 5(2):121–143, Dec 1996.
8. V. M. Markowitz and J. A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE TSE*, 16(8):777–790, August 1990.
9. E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation Across Preexisting Ontologies. In *CoopIS'96*, pages 14–25, 1996.
10. R. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB'00*, pages 77–88, 2000.
11. L. Popa, Y. Velegrakis, R. J. Miller, M. Hernandes, and R. Fagin. Translating Web Data. In *VLDB'02*, pages 598–609, 2002.
12. M. R. Quillian. Semantic Memory. In *Semantic Information Processing*. Marvin Minsky (editor). 227-270. The MIT Press. 1968.
13. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10:334–350, 2001.
14. R. Ramakrishnan and M. Gehrke. *Database Management Systems (3rd ed.)*. McGraw Hill, 2002.
15. H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hubner. Ontology-Based Integration of Information - A Survey of Existing Approaches. In *IJCAI'01 Workshop. on Ontologies and Information Sharing*, 2001.