Flexible Mapping for Retrieval from Federated Schemas with Modelling Variations

Radhika Reddy

Data Intensive Systems Center
Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology
P. O. Box 91000, Portland, OR 97291-1000 USA

reddy@cse.ogi.edu

October 9, 1995

Abstract

Retrieving data from diverse information sources is a problem faced by many organizations. Many of these information sources are databases where different schemas may represent the same basic concept in different ways. The problem that leads to this semantic heterogeneity is termed *modelling variation*. Current approaches limit the retrieval of related information from multiple sources, because they do not support a wide range of modelling variations. We propose to work with an *abstract schema*, represented as an Entity-Relationship Diagram, to represent information of interest to a user or a group of users. The abstract schema may have been created in response to a set of related user requests or queries and is used for retrieval only. Database administrators of individual schemas provide mappings that relate constructs appearing in the abstract schema to constructs in the individual schemas. The abstract schema is populated, by issuing queries against the individual schemas. The contribution of this research is the definition of the mappings that handle a wider variety of modelling variations between the abstract schema and the individual schemas than previous approaches. These mappings can then be used to populate the abstract schema with valid data from underlying information sources.

1 Introduction

Due to advances in communication technology, the number of information sources accessible to a user has grown rapidly. To make effective use of the information, providing simultaneous access to all these diverse information sources is important. Many of these information sources are databases that typically have different schemas, use different data models, and may support different query languages. This heterogeneity makes it difficult for a user to successfully retrieve information. Even if all the databases use the same data model, users still need to deal with the semantic heterogeneity between the individual databases. Semantic heterogeneity includes the case where different schemas represent the same basic concept in different ways. We refer to this aspect of semantic heterogeneity as *modelling variation*. Retrieving data from multiple information sources, in spite of the modelling variations that exist between them, is the focus of this research.

Diverse information sources are often used in ways that differ from the underlying data contained in these sources. Users may request data in a form that may not reflect the way it is present in the source schemas. This difference is the modelling variation that exists between a user request and source data schemas. Modelling variations between individual source schemas may exist because the focus of each information source may be different, as different user groups with differing perspectives may have influenced the design of the different individual schemas. Also, most semantic data models offer a rich set of modelling constructs. This flexibility gives schema designers the opportunity to represent the same concept in different ways.

Current technology (which is described in the evaluation in Section 3) limits the successful retrieval of related information in the face of these modelling variations. Most approaches access multiple schemas by integrating two or more schemas and building a common schema, which is sometimes called an *integrated schema* [21], *federated schema* [26], or simply *global schema*. Some kind of correspondence assertions between the global schema and the individual schemas are maintained by the database administrator. When a user issues a query against the global schema using a common query language, the query is decomposed into queries against the individual databases using the correspondences. The results of these

queries are then integrated to produce the answer to the original request. These approaches do not accommodate a wide range of modelling variations. They are limited by the data model they use to represent the individual schemas and also by the fact that they allow the global schema to be updated. The goal of this research has been to extend the range of modelling variations handled by current integration methodologies when only retrieval of information is supported.

We propose the use of an *abstract schema* [24], that only represents data of interest to a user or a user group at a particular time. The abstract schema may have been created as a response to a user query or a set of queries. It may refer to data from a number of individual schemas. We use an Entity-Relationship Diagram (ERD) to represent the abstract schema as well as the individual schemas. Much like the approaches mentioned above, database administrators (DBAs) of each individual database must provide correspondences relating attributes and relationships in the abstract schema to constructs in the individual schemas. We propose to use the mappings provided by the DBAs to populate the abstract schema. The populated abstract schema is then used to answer user queries.

In later sections we prove that this approach handles all the modelling variations that are possible between the abstract schema and various information sources, represented using the ERD. We also prove that our approach is correct. By correct we mean that there is no inconsistency between the abstract schema and the individual source schemas.

We use the abstract schema for retrieval only. Schema mappings for updates must be complete, from the view of the underlying schemas. This usually means all the features of the underlying schemas must be represented in the integrated schema. Also, cardinality constraints and functional dependencies in the individual schemas need to be maintained in the integrated schema. Schema mappings have generally been limited by this requirement to provide a complete mapping [21, 26]. The theme of this research is the retrieval of semantically comparable constructs from arbitrarily different underlying structures. In other words, an abstract schema is a view over the multiple databases, and we are mapping retrieval-only views down to underlying databases. This means that we do not worry about extra information in the underlying schema (beyond the abstract schema) nor about

constraints. We have no need to express even keys or functional dependencies in the abstract schema. All of this contributes to the flexibility of our approach to accommodate modelling variations.

One of the motivations for this work is the growing interest in using various information sources to answer unanticipated questions. Such questions are often issued by decision makers and require very flexible retrieval from diverse underlying databases. The notion of an abstract schema was introduced to represent the information relevant to a given high-level request [24].

In the next subsection we summarize related work in other areas of multidatabase systems. In Section 2 we discuss our approach and give proofs for the correctness and completeness of the approach. In Section 3 we evaluate the approach with respect to other related approaches. Finally in Sections 4 and 5 we present conclusions and discuss future work possibilities.

1.1 Related Work

Multidatabase (MDB) systems have been an area of active research for many years. A variety of approaches to accessing MDB systems have been proposed. Federated approaches to accessing MDBs involve building a federation by controlled integration of component databases [17]. Integrated approaches, on the other hand, involve the construction of an integrated database that supports a single manipulation language [25, 6, 2]. Other approaches propose sharing semantic data values to allow multiple, independent databases to interoperate [16, 13]. Our interest is in federated approaches, because the abstract schema we work with provides a federated view over the individual schemas.

Federated database systems are a collection of independent databases. In their survey paper, Sheth and Larson [17] propose an architecture with schemas at five levels to describe federated databases. Each individual database has a *local schema* that is translated into a common data model. Each of these translated schemas is called a *component schema*. Depending on what the local system would like to make public, a part of the component schema called the *export schema* is made available for sharing. A *federated schema* is

an integration of multiple export schemas. Finally, the *external schema* is that part of the federated schema useful to a particular user or a group of users. For the rest of this subsection we use these terms when describing the various schemas.

Building the federated schema involves schema integration [3]. Schema integration is the process of taking two or more export schemas, resolving the differences between them and building a common (global) schema, that captures all the aspects of the individual schemas. A detailed survey by Batini et al. [3] discussed twelve methodologies for schema integration. Schema integration is not an easy process. It involves among other things, resolving naming conflicts, resolving structural conflicts, instance identification, query translation and source tagging.

Structural conflicts arise when a real-world object is represented using different modelling constructs in different local schemas. That is, structural conflict is another term for modelling variation. As mentioned in the introduction, resolving modelling variations is the focus of our research. The remainder of this subsection discusses previous work in other related areas of multidatabase systems. Currently these issues are outside the scope of our research.

Naming conflicts include both the synonym problem and the homonym problem. The synonym problem arises when two entities that represent the same real-world concept have different names in different schemas. The homonym problem arises when two entities have the same name in two schemas, but model two different real-world entities. Different approaches solve this problem in different ways but user input is needed to resolve the naming conflicts. Usually this input is provided as a set of correspondence assertions relating similar entities in different schemas even if they are named differently [17, 21].

Query processing [14, 9, 8] and query formulation [5, 18] in heterogeneous systems have been widely researched. Every federated system, whether it builds a federated schema (usually a view) or not, needs to support some mechanism for querying against the multiple schemas. When a global view is constructed, queries are issued against the global view. The query is then modified into equivalent queries against the local schemas [6]. Automated query translation has also been proposed [26]. In this approach the federated schema

captures the access path information for all attributes. So users do not have to specify how the attributes are to be derived from the component schemas. Each query against the federated schema is broken down into queries against individual schemas. The results are assembled and annotated with additional information and returned to the user. In our approach user queries are issued against an abstract schema, that contains data of interest from multiple schemas. The abstract schema can be populated with data from the component schemas based on the mappings provided by the DBAs. A populated abstract schema can then be used to answer user queries.

Instance identification is the process of determining the correspondence between object or entity instances from more than one database. It may happen that different databases model different information about the same object and there may be no way this information can be joined as no common key (information about the entities) exists among the different databases. Instance identification attempts to discover multiple occurrences of the same individual instance and resolve any inconsistencies that arise when redundant information is represented in several databases. Approaches proposed to solve this problem include using artificial intelligence techniques and inference rules [22] to derive data to identify the different instances. Another method of solving this problem is to request that a DBA provide semantic information about the instances [11].

Source tagging is the process of annotating a query (issued against a federated schema) result with additional information, like the source of the data and intermediate data sources used to arrive at the data [23]. This information may be needed to check the validity of the result.

Other related topics include database mapping [12], which discuss mapping between two databases expressed using different data models. We do not discuss heterogeneous data models here, as we assume that all our schemas are expressed using a single data model, the Entity-Relationship model. We plan on extending this work by considering heterogeneous and more complex data models.

2 Our Approach

We use an abstract schema to represent useful information that is to be extracted from multiple schemas. An abstract schema captures information that is of interest to a particular user or a group of users.

There are many advantages of using an abstract schema. An abstract schema can match a high-level user request. These high-level requests usually leave out the detail of the structure of the underlying information systems. One goal is to avoid the detail of the underlying schema in the abstract schema. For example, Figure 1 Schema2, which is an individual schema, has a *Contract* entity, that represents documents that relate *Employee* entities to the *Organization* they work for. This same entity and the functions *signs* and *set_by* are represented as only a relationship *employed_by* in Schema1, which is the abstract schema. In other words the details of the *Contract* entity need not be present in the abstract schema, if the user just needs to know which organization a particular person works for. An alternative to an abstract schema is to use an integrated schema, to represent all the underlying sources and let users issue queries against it, as is done in most federated approaches [21, 26]. Integrated schemas are especially difficult to define and maintain when the number of information sources goes up and diversity increases.

Database administrators at various sites provide mappings that relate abstract schema constructs to constructs in the individual schemas at their site. The mappings provide semantic information about how data from different schemas relates to the abstract schema. They also provide a way of representing the modelling variations that exist between the different schemas.

Based on the mappings, which relate abstract schema constructs to corresponding constructs in the individual sources, the abstract schema can be populated and then queried. Queries would then be issued against the individual data schemas to extract the information that a particular abstract schema construct maps to. It is at this point that any representational conversions between values are performed. Representational conversions are required, for example, when in the abstract schema an amount is needed in dollars, but the amount in

one schema is represented as shillings, and in still another schema as rupees. Rupees and shillings must be converted to dollars as the abstract schema is populated.

The problem of instance identification also needs to be solved at this step. If an abstract schema entity maps to two different entities from two different schemas, additional semantic information about how to interpret the data is needed. For example, if there exists a salary attribute in the abstract schema and salary is recorded in two local schemas. We need to know if we have to choose the salary from only one of the schemas (the current salary), or if salaries from the two schemas need to be added (as the two schemas model the fact that a person has two incomes).

We do not discuss these issues any further. We are currently focusing on the mappings provided by the DBA, and how these mappings resolve all possible modelling variations, when the mappings are from an abstract schema, represented as an ERD, to an underlying source schema, also represented as an ERD.

2.1 Model used

We use ERDs for schemas at both levels because they are very widely used to represent database schemas and they have already been mapped to many traditional data models. And except for concepts such as generalization, the ER model corresponds closely to the ODMG Object Definition Language [1]. Our use of ERDs means that we accommodate any data model that can be represented using entities, relationships, and attributes.

In this research, each of the ERDs are converted into a data model that is based on the functional data model [19]. This conversion is straightforward and is only done to make the formalization of the mapping more compact.

The features of the functional data model used in this research can be outlined as follows.

- The basic constructs of the data model are elements and functions. Elements can be entities or value-sets. Entities are intended to model real-world objects that exist and they correspond to entities in an ERD. Value-Sets are analogous to domains in the original relational model [4]. We define a value-set as a domain of permitted values

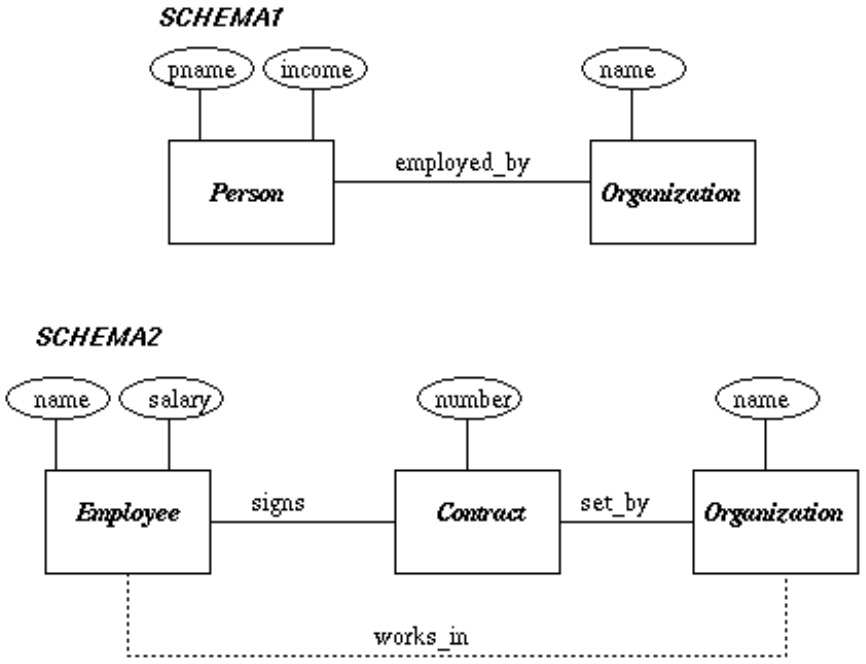

Figure 1: An Entity maps to a Relationship

that may be assigned to an attribute of a particular entity. Note that technically, elements and entities (as well as relationships) are actually element types and entity types (and relationship types). We blur the terminology as is commonly done with the ER model.

- A function can be a relationship, an attribute or a representation conversion (rep-conversion).

  $Relationship : Entity \rightarrow Entity$
  $Attribute : Entity \rightarrow Value\text{-}Set$
  $Rep\text{-}conversion : Value\text{-}Set \rightarrow Value\text{-}Set,$
  (as long as the rep-conversion is invertible.)

  Note that many rep-conversions are one-to-one functions, and thus are invertible.

  We assume the existence of inverse relationships, though they are not explicitly shown in the ERD.

  $Relationship^{-1} : Entity \rightarrow Entity$
  $Attribute^{-1} : Value\text{-}Set \rightarrow Entity$

$Rep\text{-}Conversion^{-1}$: $Value\text{-}Set \rightarrow Value\text{-}Set$

Note that each function (and inverse function) $f: X \rightarrow Y$ (and $f^{-1}: Y \rightarrow X$) may map one instance of $X(Y)$ to zero, one or more instances of $Y(X)$.

foreach $x \in X$, $f(x) \subseteq Y$ and
foreach $y \in Y$, $f^{-1}(y) \subseteq X$

We require that

$y \in f(x) \Leftrightarrow x \in f^{-1}(y)$

For this research, we assume that attributes do not take null values.
When $Attribute : X \rightarrow Y$, attribute cannot map an instance of X to zero instances of Y.

Figure 1 shows two ERDs. The conversion of the ERD of Schema1 to a schema in the functional data model is given below:

1. *Person* and *Organization* are entities.
2. *pname*, *income*, and *name* are attributes formally listed as:

   $pname(Person) = STRING$
   $income(Person) = INTEGER$
   $name(Organization) = STRING$

   where STRING and INTEGER are value-sets. Note that value-sets do not appear in the ERD notation; although they are usually specified as part of the schema that corresponds to the ERD.
3. *employed_by* is a relationship formally represented as :

   $employed\_by(Person) = Organization$
4. $employed\_by^{-1}$ is the inverse relationship, formally represented as:

   $employed\_by^{-1}(Organization) = Person$

   Similarly inverse attributes exist.

- Rep-conversions are functions that convert one value to another. For example in Figure 3, *maps* is the representation conversion that converts *cid* attribute to *cno*.

cno as STRING = maps(cid as STRING)

A rep-conversion might also convert from one encoding to another. For example, from M or F to Male or Female for gender. A rep-conversion may also be the identity function so that we can navigate through the underlying source schema using the equi-join operator on attribute values.

- Properties of an object may be derived from other properties. That is, functions can be composed. As an example, the *works_in* property in Figure 1 Schema2 intuitively relates an *Employee* to the *Organization* in which he or she works. It is the composition of *signs* and *set_by* properties.

  works_in(Employee) = Organization
  where *works_in = signs.set_by*
  so *set_by(signs(Employee)) = Organization*

- A composition of functions can be formally defined as:

  If f_1, f_2, ..., f_n, $n > 1$, are functions then the composition $f_1.f_2 \ldots f_n$ is defined when $domain(f_i) = range(f_{i-1})$ for $2 \leq i \leq n$ and is defined as:

  - if f_1: $X \rightarrow Y$ *and* $f_2 : Y \rightarrow Z$
    $f_1.f_2(x) = f_2(f_1(x)) = \{z \mid z \in Z \wedge \exists\ y,\ y \in f_1(x) \wedge z \in f_2(y)\}$
  - if $f_1 : X \rightarrow Y$, $f_2 : Y \rightarrow Z$, $\ldots$ $f_n : V \rightarrow W$
    then $f_1.f_2 \ldots f_n(x) = f_n(f_1.f_2 \ldots f^{n-1}(x))$
    In other words, any finite set of functions that meet the constraints above can be composed.

A formal description of the grammar for this functional model DDL is given in the appendix. The grammar borrows some of its constructs from the ODMG Object Definition Language [1].

2.2 Mappings

As mentioned in the previous section, database administrators at every site provide mappings that relate the abstract schema and the individual schema they manage. The mapping from the abstract schema to the individual schema is represented as a function **p**. For the following definition of p, the constructs from abstract schema and the individual schemas

are represented with the first letter in upper-case, but the individual schema constructs are primed ($\prime$).

- $p(Fn) = fn\_list$ *where*
  $Fn = Attribute \mid Relationship$
  $fn\_list = Fn' \mid fncomp$
  $Fn' = Attribute' \mid Relationship' \mid$ Rep-$Conversion'$
  $fncomp = Fn' \mid fncomp(Fn')$

In Figure 1, if Schema1 is the abstract schema and Schema2 is the individual schema, the mappings provided by the DBA would be as follows:

$p(pname) = name$
$p(income) = convert\_to\_annual(salary)$,
$p(employed\_by) = signs.set\_by$.

The *convert_to_annual* function is the representation conversion from a value in the abstract schema to a value in the individual schema, and is used when populating the abstract schema.

2.3 Proofs of Correctness and Completeness

Notation The abstract entities are represented in upper-case, the functions and instances in lower-case. The same notation is used for the individual schema, except that they are primed ($\prime$).

Definition 1 *Information present in a database, as represented in an ERD, consists of elements, represented as entities or value-sets, and functions among these elements represented as attributes, relationships, and representational conversions.*

Definition 2 *Define "connected_to" relation between two elements iff there exists a function relating the two elements.*
$connected\_to(X, Y) \Leftrightarrow f : X \rightarrow Y$ *where*
f can be an attribute, relationship, rep-conversion, inverse attribute, inverse relationship, or inverse rep-conversion; or a finite composition of these functions and inverse functions.

Assumption 1 *Every construct in the abstract schema is mapped to some construct (or constructs) in the individual schemas.*

We introduce the following constraints to define a well-formed mapping, p.

Constraint 1 *This constraints states that if a function between two elements X and A in the abstract schema is mapped to a function between X′ and A′ in the individual schema, then X and A intuitively map to X′ and A′ respectively.*

> *if f:* $X \rightarrow A$ *and* $f' : X' \rightarrow A'$
> *then* $p(f) = f' \Rightarrow p(X) = X'$ *and* $p(A) = A'$
> *Further if,* $a \in f(x)$ *then* $p(x) = x'$,
> *and* $p(a) = a' \Rightarrow a' \in f'(x')$.

Constraint 2 *If the attributes of an entity in the abstract schema map to attributes of more than one entity in the individual schema, there must exist a function relating these entities in the individual schema, and this function has to be specified by the DBA.*

> *if* $f_1 : E \rightarrow A_1, f_2 : E \rightarrow A_2$, $f_1' : E_1' \rightarrow A_1'$, $f_2' : E_2' \rightarrow A_2'$
> *then* $p(f_1) = f_1'$, $p(f_2) = f_2' \Rightarrow$ *connected_to*(E_1', E_2')
> *More generally* $(\forall E_i')$ $(\forall E_j')$ *where* $E_i' \neq E_j'$,
> $f_i : E \rightarrow A_i$, $f_j : E \rightarrow A_j$, $f_i' : E_i' \rightarrow A_i'$, $f_j' : E_j' \rightarrow A_j'$ *then*
> $p(f_i) = f_i'$, $p(f_j) = f_j' \Rightarrow$ *connected_to*(E_i', E_j')

Statement 1 (Correctness) *If two elements are related through a function in the abstract schema, then the corresponding elements in the individual schema are also connected through the function that corresponds through p to the abstract schema function.*

> *Formally, connected_to*(X, Y) *based on* $f \Rightarrow$ *connected_to*$(p(X), p(Y))$ *based on p(f), where X and Y are elements in the abstract schema and p(X) and p(Y) are elements in the individual schema.*
> *Further if,* $y \in f(x)$, $p(x) = x'$, *and* $p(y) = y'$, *and* $p(f) = f'$ *then* $y' \in f'(x')$

Proof:

This follows from Constraint 1 and Constraint 2.

Statement 2 (Completeness) *Even in the face of all modelling variations (without disjunctive mappings), possible from the abstract schema using ERDs, to underlying schemas*

using ERDs, all constructs in the abstract schema will be successfully mapped to the underlying schema.

Proof:

The modelling variations possible between any two schemas represented as an ERD can be exhaustively listed. This list is similar to those presented previously using the relational model to represent the schemas [7, 10]. If these variations can be expressed using our mappings, then we can say that all modelling variations are successfully mapped.

Function maps to Function or Function maps to Composition-of-Functions This is the trivial case, as our definition of mappings maps a function to either a function or a list of functions.

Function maps to List-of-Functions The information captured by one function in the abstract schema, is split between a list of functions in the individual schema. This is a case of disjunctive mapping, and is outside the scope of the completeness statement.

Element maps to Element Elements mapping to elements follows intuitively from the definition of mappings. Constraint 1 and Constraint 2 formally illustrate this.

Element maps to List-of-Elements This is supported in our mapping when the information captured in one entity in the abstract schema is split between a list of entities in the individual schema, and Constraint 1 and Constraint 2 are satisfied.

An example of this is shown in Figure 2. The entity *Bank_scheme* in the abstract schema corresponds to entities *Banker, Branch,* and *Customer* in the individual schema. We discuss this example in detail in Section 3.

Any other mapping is a disjunctive mapping, where an element in the abstract schema maps to two or more disconnected elements in the individual schema.

Element maps to Function An element mapping to a function violates Assumption 1, that every construct needs to be mapped. Currently we are not handling these variations, but will address them in our future work.

As an example, if Figure 1 Schema2 is the abstract schema and Schema1 the individual schema. The mappings as specified by the DBA would then be:

p(name) = pname
p(salary) = convert_to_salary(income)
p(name) = name

There is no function in the individual schema that corresponds to the function *number* in the abstract schema. Hence, even though *Contract* intuitively maps to the relationship *employed_by*, we cannot map its attribute, *number*. The value of *number* is needed, because the two functions *signs* and *set_by* are composed on the value of *number*.

Function maps to Element A function mapping to an element, in effect means that the function maps to a composition of two distinct functions connected by the element. More formally:

if $f(E) = A$, $f_1'(E') = A'$ and $f_2'(A') = B'$
then $p(f) = A' \Rightarrow p(f) = f_1'.f_2'$
where $p(E) = E'$ and $p(A) = B'$.

An example of this mapping is shown in Figure 1. The information specified in *employed_by* relationship is captured by the *Contract* entity. But the mappings are specified as a composition of *signs* and *set_by* functions, that are connected through the *Contract* entity.

There are no other cases. Note that disjunctive mappings are currently not supported in other related approaches [21] because they prevent update of the integrated schema. Thus, excluding disjunctive mappings from our work at the present time, does not invalidate our claim that we support more modelling variations than existing approaches.

3 Evaluation

We review current approaches that deal with federated database systems. We compare our approach with existing approaches.

Parent et al. [15, 21] propose a database integration methodology. They use the ERC+ [20] data model to express the individual schemas. The correspondences between two schemas are expressed as assertions. The integrator receives as input, the two schemas to be integrated and the interschema assertions. A set of integration rules are provided. Each assertion is considered and an appropriate integration rule is applied and an integrated schema is built. The integrated schema can be updated.

The abstract schema in our approach is only used for retrieval. This gives us greater flexibility in our mappings. With their current integration rules, Parent et al. do not handle cases when an entity in a component schema corresponds to more than one entity in another component schema. We handle this case because we allow the attributes of an abstract schema entity to be split between two or more entities. An example of how we handle this is shown in Figure 2.

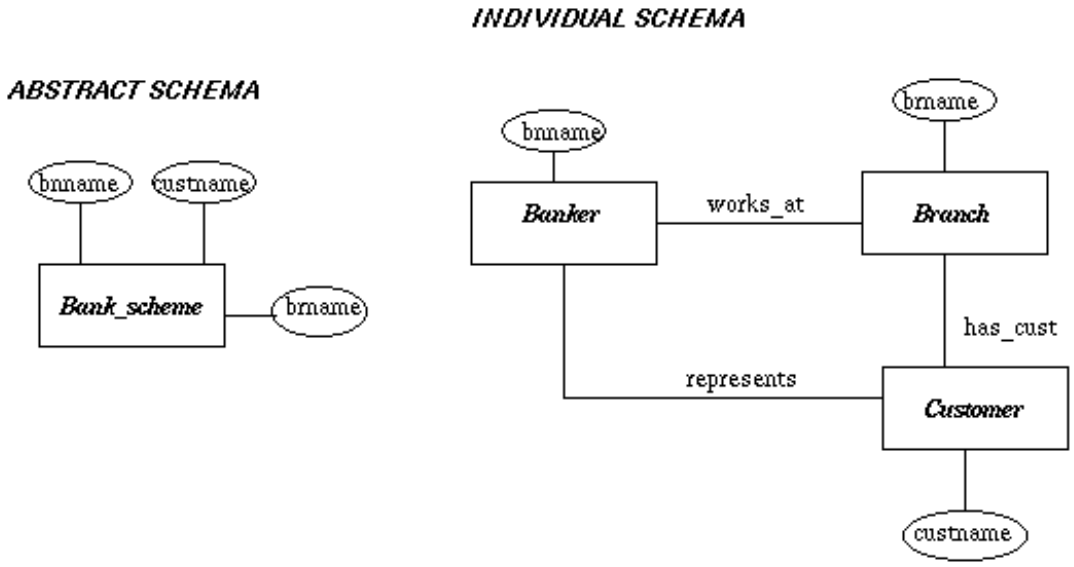

Figure 2: Entity maps to Entity list

The two schemas in Figure 2 capture information about a banking domain. The individual schema has three entities *Banker, Branch,* and *Customer*. The three entities are related via three different relationships. The same information is captured in the abstract schema in the *Bank_scheme* entity that captures information about a customer, a branch, and a

banker.

The mappings as provided by the DBA:

$p(bnname) = bnname$
$p(custname) = custname$
$p(brname) = brname$

According to Constraint 2, the relationships between the three entities must be specified by the DBA. For this example, the following three relationships are specified as the proper way to connect the three entities in the individual schema, according to the intended semantics of the abstract schema.

$works\_at(Banker) = Branch$
$has\_cust(Branch) = Customer$
$represents(Customer) = Banker$

In the approach by Parent et al. when an existing schema is modified the correspondence assertions need to be modified, and the integrated schema has to be rebuilt. But in our approach, when a schema at a site changes, the DBA at that site has to only provide new mappings between the changed schema and the abstract schema. This means less work needs to be done by a DBA, if individual schemas change frequently.

Zhao et al. [26] proposed a federated meta-model approach where a federated schema containing information about the existing schemas is built. The difference between the federated schema and abstract schema is that the federated schema has to contain information about all the attributes that are present in the individual schemas. One of the advantages of using an abstract schema is we can mask the details of the individual schemas. Our aim is to be able to populate the abstract schema when it contains as little information as is necessary to suit the user.

We also handle value-value functions. Both the above mentioned approaches have relationships (foreign-keys) relating two entities (relations), but there is no explicit function relating two values. We handle a wider range of modelling variation due to this function. In Figure 3, the functions (relationships, attributes, and rep-conversions) can be listed as:

$name(Student) = STRING$
$takes(Student) = Course$

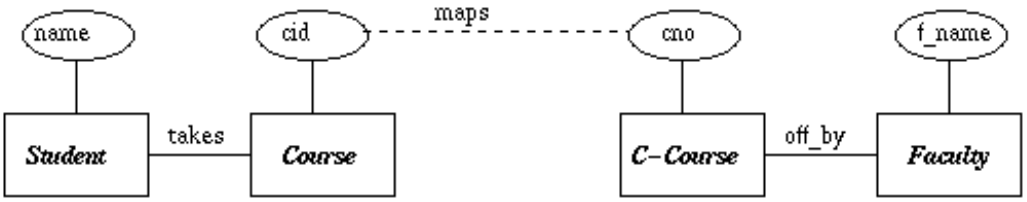

Figure 3: An Example of Value-Value function

cid(Course) = STRING
maps(cid as STRING) = cno as STRING
cno(C-Course) = STRING
off_by(C-Course) = Faculty
f_name(Faculty) = STRING

There is no explicit relationship between attributes *cid* and *cno*, but the two are related by a *maps* rep-conversion function. For example, courses offered at OGI have a *cid*, which is the course number. Some of these courses are offered through OCATE, and hence non-OGI faculty can teach them. The *cno*, specifies the number the course is assigned at the university, whose faculty member teaches it. Even in the case where a representation conversion is not needed, the *Value-Set* $\rightarrow$ *Value-Set* function is the identity function that corresponds to an ordinary, value-based equi-join. For example, in the above schema, if both *cid* and *cno* used the same Value-set, then *maps* would be an identity function and this would be an example of an equi-join. This is one of the main reasons why we can handle the full range of modelling variations: we allow any element to be an entity or a value and mix navigational traversal through relationships with equi-join through values.

4 Contribution & Conclusions

In this paper we have shown how we can successfully map to extract data, represented as an abstract schema, from multiple data schemas in response to a user query. The data extraction works in face of modelling variations between the individual schemas. Users are not expected to know how the data from the individual data sources is related. The main emphasis of this research has been resolving modelling variations between individual

schemas. Given that the abstract and individual schemas are expressed using the ERD, we can populate the abstract schema in face of all the possible modelling variations that can exist between the individual schemas and the abstract schema. We have formally proved the completeness of our approach.

The main differences (and hence contribution) between our work and previous work in this area can be listed as follows:

Modelling Variations We handle a wider range of modelling variations. There are two reasons for this. Since we allow a new function *(Rep-conversion)* relating two values, we can handle more variations. An example of this is discussed in Figure 3. Also, since we do not allow the abstract schema to be updated, we can handle more variations. An example of this is shown in Figure 2.

Easier for Users The users specify queries in terms of what they want, and not in terms of what data is present. Due to this, they need not know the actual detail of the individual schemas.

Schema Integration on Demand This means we do not integrate all the individual schemas to provide a global schema. Instead we create an abstract schema, that contains data of interest to a group of users. This abstract schema contains (integrates) only that part of data from every individual source, that answers a users query or is of interest to the user.

Less correspondence assertions Previous approaches list correspondence assertions relating every construct in one schema to a corresponding construct in another schema. Thus the DBA specifies a long list of assertions. In our approach, the DBA only specifies mappings relating functions. All other correspondences for entities and value-sets are implied by this mapping.

5 Future Work

In this paper we discuss a methodology for resolving modelling variations between an abstract schema and an individual schema. This is a first step to accessing data from multiple schemas. For future work, we plan to work on the following issues which we haven't worked on in this paper.

- extending the ERD by considering generalization and aggregation hierarchies.
- considering heterogeneous data models. We can already handle modelling variations when the schemas are expressed using the relational data model, because of the well-known mapping from an ERD to relational model using foreign keys to represent relationships.
- including null attributes and relationships with attributes in the ERDs used to represent the schemas.
- allowing the case when an abstract schema construct does not map to any construct in the individual schema. In some cases, this can be allowed by generating an identifier for the unmapped construct.
- allowing disjunctive mappings. This case arises when an abstract schema construct maps to two or more distinct unrelated individual schema constructs. The complexity of disjunctive mappings, arises from the scope of the disjunctive mapping of individual constructs in the context of the complete abstract schema

Appendix

<entity> ::= **Entity** <entity-name>
| **keys[s]** <key-list>

<value-set> ::= **Value-set** <valset-name>
<entity-name> ::= <string>
<valset-name> ::= <string>
<key-list> ::= <key-name> | <key-name>,<key-list>
<key-name> ::= <attr-name> | (<attr-list>)
<attr-list> ::= <attr-name> | <attr-name>,<attr-list>
<attr-name> ::= <string>
<function> ::= <fn> | <fncomp>
<fncomp> ::= <fn> | <fncomp>"("<fn>")"
<fn> ::= <attribute> | <relationship> | <rep-conversion>

<attribute> ::= **Attribute** <attr-name>"("<domain-entity>")"
value-set <valset>
inv <inv-attrname>"("<valset>")"

<relationship> ::= **Relationship** <rel-name>"("<domain-entity>")"
range <range-entity>
inv <inv-relname>"("<range-entity>")"

<rep-conversion> ::= **Rep-conversion** <rep-name> "("<valset>")"
value-set <valset>
inv <inv-repname>"("<valset>)

<rel-name> ::= <string>
<rep-name> ::= <string>
<inv-relname> ::= <string>
<inv-attrname> ::= <string>
<inv-repname> ::= <string>
<domain-entity> ::= <entity>
<range-entity> ::= <entity>
<valset> ::= <value-set>

References

[1] T. Atwood, J. Duhl, G. Ferran, M. Loomis, and D. Wade. *Object Database Standard: ODMG - 93*. Morgan Kaufmann, 1994.

[2] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, SE-10(6):650–664, November 1984.

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. In *ACM Computing Surveys*, volume 18(4), pages 323–364, December 1986.

[4] E. F. Codd. A relational model of data for large shared data banks. *Communcations of the ACM*, 13(6):377–387, June 1970.

[5] B. Czejdo, D. W. Embley, and M. Rusinkiewicz. An approach to schema integration and query formulation in federated database systems. In *Proceedings of the IEEE Data Engineering Conference*, pages 477–484, December 1987.

[6] U. Dayal and H. Y. Hwang. View definition and generalization for database integration in a multidatabase system. *IEEE Transactions on Software Engineering*, SE-10(6):628–645, Nov 1984.

[7] W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, 24(12):12–18, December 1991.

[8] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In J. Clifford and R. King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on the Management of Data*, volume 20, pages 40–49, May 1991.

[9] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Lecture Notes in Computer Science: Deductive and Object-Oriented Databases*, pages 81–100, December 1993.

[10] C. Lee, C. Chen, and H. Lu. An aspect of query optimization in multidatabase systems. *ACM SIGMOD Record*, 24(3):28–33, September 1995.

[11] E. P. Lim, S. Prabhakar, J. Srivastava, and J. Richardson. Entity identification in database integration. Technical Report TR 92-62, University of Minnesota, Minneapolis, MN 55455, December 1992.

[12] L. Mark, N. Roussopoulos, T. Newsome, and P. Laohapipattana. Incrementally maintained network→relational database mapping. *Software-Practice and Experience*, 22(12):1099–1131, December 1992.

[13] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, March 1995. IEEE Computer Society Press.

[14] M. P. Reddy, B. E. Prasad, and P. G. Reddy. Query processing in heterogeneous distributed database management systems. In A. Gupta, editor, *Integration of Information Systems: Bridging Heterogeneous Databases*, Piscataway, NJ, 1989.

[15] C. Parent S. Spaccapietra and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. LBD research report, EPFL, Lausanne, February 1991.

[16] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous database systems. In *Transactions on Database Systems*, June 1994.

[17] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. In *ACM computing surveys*, volume 22, pages 183–237. Association for Computing Machinery, Inc., September 1990.

[18] A. P. Sheth, J. A. Larson, A. Cornelio, and S. B. Navathe. A tool for integrating conceptual schemas and user views. In *Proceedings of the Fourth International Conference on Data Engineering*, pages 176–183. IEEE Computer Society Press, 1988.

[19] D. W. Shipman. The functional data model and the data language daplex. In *ACM Transactions on Database Systems.*, volume 6 of *ACM Series on Computing Methodologies.*, pages 140–173. Association for Computing Machinery, March 1981.

[20] S. Spaccapietra and C. Parent. ERC+: an object based entity relationship approach. In P. Loucopoulos and R. Zicari, editors, *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development.* John Wiley, 1992.

[21] S. Spaccapietra and C. Parent. View integration: a step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 1993.

[22] Y. R. Wang and S. E. Madnick. The inter-database instance identification problem in integrating autonomous systems. In *Proceedings of the fifth international conference on Data Engineering*, pages 46–55. IEEE Computer Society Press, Feb 1989.

[23] Y. R. Wang and S. E. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In *Proceedings of the 16th VLDB Conference*, pages 519–538, Brisbane, Australia, 1990.

[24] G. Washburn. *Vertical Information Management: A Framework to Support High-Level Information Requests in the Context of Autonomous Information Systems.* PhD thesis, Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA, May 1995.

[25] W. K. Whang, S. Chakravarthy, and S. B. Navathe. Heterogeneous databases: Inferring relationships for merging component schemas, and a query language. Technical Report UF-CIS-TR-92-048, University of Florida, December 1992.

[26] J. L. Zhao, A. Segev, and A. Chatterjee. A universal relation approach to federated database management. In *Proceedings of the eleventh international conference on Data Engineering*, pages 261–270, Taipei, Taiwan, March 1995. IEEE Computer Society Press.