Helping Data Suppliers and Consumers Negotiate the Details: 
A Database View Approach

Arnon Rosenthal, Edward Sciore

Abstract

Databases that interchange information will rarely have the same native form, so one must map between the supplier’s native interface and the consumer’s. Using a SQL view to define this map is convenient and powerful, because it provides not just an evaluation mechanism, but also query, and (to some degree) update and trigger capabilities. However, SQL views do not map the critical metadata (e.g., security, source attribution, and quality information) between supplier and consumer. This paper addresses the consequences of this lack, and provides some solutions.

We consider several negotiation scenarios between supplier (base table) and consumer (view), in order to identify requirements for passing metadata. We show that the number of translation rules shrinks greatly if we separate descriptions of metadata content from how the results are to be used. We then describe a theory that enables automated tools to translate many useful forms of metadata through views. In particular, the theory attempts to minimize the effort required from both tool implementers and data administrators.

Keywords: autonomous databases, views, metadata, warehouse, federation, negotiation.

Addresses: A. Rosenthal, The MITRE Corporation, Bedford MA, USA amie@mitre.org
E. Sciore, Boston College, Chestnut Hill, MA, USA and MITRE, sciore@cs.bc.edu

1 Introduction

This paper investigates the communication problems arising from autonomy and data heterogeneity in applications that support cooperating organizations.

Autonomy means that the shared information is not under central control. Typically, many users will have an interest in each piece of information, but only one user or group will have the right to change it. The collaborating parties need to communicate their changes, request changes from others, and negotiate differences.

Data heterogeneity means that the application does not provide the same interface to all collaborators. Instead, each party views information through its own interface, seeing only relevant data and operations, structured appropriately. Consequently, there must be a mechanism for translating messages from the sender’s interface to the recipient’s.
One such mechanism is to hand-code the translations into the implementation of the interfaces, as is done in typical 3GL applications. Another mechanism is to specify the interfaces declaratively, as is done in database-oriented applications. Here, interfaces are known as schemas, and SQL is used to specify the translation from source schemas to view schemas.

The SQL definition of a view only specifies how a view table is constructed from source tables. However, SQL’s declarative nature allows a database system to perform other translations with limited extra help. For example, the translation from a view table to source tables is known as view update [Kel86]. Database systems can perform this translation automatically for certain views; others require human choices; in the toughest cases, additional code must be provided. We call this process semi-automatic translation.

In this paper, we explore how a system can support the semi-automatic translation of metadata (such as security, constraint enforcement, or data quality) between sources and views. Completely automatic metadata translation is not always possible, because different kinds of metadata behave differently. For example, consider a view that is the join of two source tables. The access rights associated with the view table may be the intersection of the source access rights, whereas the trustworthiness of the view may be the minimum of the trust values of the sources. Other times, the translation may depend on how the source data was transformed to produce the view, and perhaps on organizational policies.

A substantial amount of metadata can be kept with each schema; it is quite reasonable for table columns to have around 8 properties each. The task of manually communicating metadata between schemas and maintaining correctness is daunting. To obtain a full flow of metadata, where all parties see a consistent picture, some amount of automation is essential.

Current DBMSs and metadata repositories have no automatic way of coordinating source and view metadata. Critical metadata communication (such as for access rights) is often performed manually, and the rest is ignored. As a result, most metadata is not shared. Automated metadata translation can create a virtuous circle. When metadata is exploited more widely, it has greater value to the organization, leading to incentives to capture and propagate more metadata. The result is that databases will be better described, and more usable by all parties.

This paper is organized as follows. Section 2 considers the role metadata plays in the communication between parties, and gives illustrative examples. Section 3 categorizes the communication needs shown in Section 2, and shows how they can be modeled with meta-attributes. Section 4 considers the problem of automating the translation of meta-attributes. As with view update, there are varying degrees of automation possible – some special cases require little additional specification, whereas the general case requires more. Section 5 concludes with research issues and open problems.
2 Collaboration Scenarios

Consider an insurance application used by three parties: the IS dept, the actuarial dept, and the marketing dept. The IS dept maintains two source tables:

POLICY(AutoId, ModelName, DriverName, DriverAge, Address, … )
ACCIDENT(AutoId, Year, DamageAmt, … )

The marketing dept sees a projection of the POLICY table containing customer information; the view is defined by the SQL query:

create view CUSTOMERS as
select DriverName, DriverAge, Address
from POLICY

The actuarial dept sees statistical information about accidents, which requires a join of the two source tables; the view is defined by the SQL query:

create view STATS as
select Year, DriverAge, sum(DamageAmt)
from POLICY p, ACCIDENT a
where p.AutoId = a.AutoId
group by Year, DriverAge

The two view tables also have different implementations. The marketing dept keeps CUSTOMERS virtual – operations to it are executed directly on the source tables. The actuarial dept extracts data into a separate data warehouse, which maintains a snapshot of STATS. The warehouse has its own security system.

This example is very simplified. In practice, the views are likely to include attribute renaming, conversions between category names and codes, and other changes that make each schema foreign to the others’ users. Our theory is designed to handle such cases.

Below we list some example scenarios, illustrating the wide variety of ways that metadata can be used in communication and negotiation. In each scenario, note that the system translates messages so that users need not manually map to the foreign schema.

1) Security Issues:
   a) The IS dept has established that only certain user groups (e.g., Auditors, ClaimsAdjusters) will have access to IS data, regardless of interface used, and informs the view parties of this. The marketing dept gets a message that describes the access permissions for marketing’s view tables implied by this policy. The actuarial dept gets a message telling what permissions are allowable for the warehouse to grant. These grants could be executed automatically by the warehouse, or the warehouse could treat them as an upper bound and make manual assignments.
   b) The marketing dept sends a message to the IS dept requesting additional write permissions on the CUSTOMERS view. This message is received by the IS dept as the implied request to grant more permissions on the POLICY table. The IS
dept may choose to honor the request, or it may send a counter-proposal to the marketing dept. This negotiation can continue until both parties are satisfied.

c) The marketing dept has been independently enforcing some access permissions on its views of the data (so as to provide marketers with faster feedback, in terms of their own view). It sends a message to the IS dept describing those permissions. The IS Department trusts the marketing dept, and decides that for requests coming through the marketing view, it will only check the access controls not checked on the views. The IS department knows exactly what to do because it understands the marketing dept actions in terms of the IS source tables.

d) The company wants to compare the actuary warehouse’s enforced permissions with the allowed permissions, to identify excessive grants at the warehouse or source permissions that seem not to be in use. The auditors translate the permissions so they are expressed against the same schema (probably the sources), and then compare them.

e) The IS dept is considering revoking some access permissions on its source tables, but wants to ask whether the view parties have strong objections. The view parties see the proposed changes (in terms of their view tables) and respond.

2) Data Quality Issues:
   a) Each source-table column has information about the data provider, accuracy, and precision estimates. This information should be associated with each corresponding column in the views.
   b) The organization has committed to knowledge management. It provides tools so that view users can estimate the importance and accuracy of view granules that they use. This information is periodically sent to the IS dept, in terms of source granules.
   c) The actuarial dept is unhappy with the accuracy of the data it receives – city names are too frequently misspelled. It sends feedback identifying the problem to the IS department.
   d) The IS dept gives guarantees for data in tables it provides (e.g., precision, accuracy). Marketing has specified how good its inputs ought to be. The company wants to determine granules where the provided data does not satisfy the specifications.

3) Constraints and Error Messages
   a) A user of the marketing view performs an update to CUSTOMERS that violates an integrity constraint on the source table POLICY. The resulting source error message is translated to refer to view tables and only then sent to the user.
   b) The marketing dept enforces some constraints on data entry. It communicates this to the IS department, so the latter can avoid making redundant constraint checks.

4) Electronic Commerce Issues
   a) Access to each source column requires a micropayment. The information policy committee wishes to know the payment required for each view.
3 Overview of the Proposed System

We now try to capture the requirements implied by the above scenarios, and to identify the major functions of a tool that might meet them. It would be fruitful (but beyond our scope) to see how well existing models of negotiations meet all these requirements. Visit [Ros98b] to see a simulation (html pages) of how such a tool might support metadata negotiations.

The proposed tool is based on three ideas: attaching metadata to granules of the database; translators that map metadata between schemas (upward or downward); and instructions that describe how the translated metadata should be employed at the recipient schema. We now consider each area in turn.

3.1 Attaching Meta-Attributes

We assume there is an annotation facility for attaching information to various granules of a database (e.g., a table, column, cell, or view). We believe that such a facility should be a fundamental capability in future DBMSs, but the full set of issues (e.g., administration and performance) are too big to tackle here. The notation below merely provides enough flexibility to express this paper’s ideas.

We model the meta-information needs of an application using meta-attributes. A meta-attribute is an attribute that can be associated with any database granule, in any schema. A meta-attribute instance is the association of a meta-attribute with a granule. The term T.M denotes the instance of meta-attribute M with table T, and T.C.N denotes the instance of meta-attribute N with column C of T. Values are assigned to meta-attribute instances.

As in [Sci94], we assume that meta-attributes are global to the application (i.e., can be attached to any schema), and are understood by all parties to have the same meaning. Wrappers or mediators can mask any heterogeneity. The organizational and technical issues of getting an agreed set of meta-attributes are important, but not addressed here.

Consider the scenarios of Section 2. Scenarios 1a, 1b, and 1e involve checking access predicates on tables. This can be supported by having a meta-attribute Rights associated with the tables (both base and virtual) in the system. That is, the value for POLICY.Rights is the access predicate corresponding to the POLICY table, etc. Scenarios 1c and 1d involve also knowing the access rights enforced by the warehouse. This requires another meta-attribute, say EnforcedByMktg.¹

Scenarios 2a, 2c, and 2d can be supported by associating meta-attributes Provider, Accuracy, and Precision with appropriate columns. Example meta-attribute instances are POLICY.AutoId.Provider, ACCIDENT.DamageAmt.Precision, and

¹ For the future, one might model this by a single meta-attribute, with a parameter to distinguish the preposition’s object, e.g., Mktg.
CUSTOMERS.Address.Accuracy. Scenario 2b would assign each column two meta-attributes for feedback information – importanceForMktg and importanceForActu – assigned. Finally, the constraint scenarios could associate a meta-attribute Constraint with each table or column.

3.2 Metadata Translation
An upward translator takes as input a view definition and the source’s input tables, annotated with metadata values. It produces a new set of meta-attribute values. If the mapping is ambiguous, a translator can produce a set of alternative results. A downward translator does the reverse mapping, taking view data and producing meta-attribute values for the sources. (We talk of translations dealing with one meta-attribute instance; the extension to multiple attributes and multiple tables is straightforward.)

The duty of a translator is to capture the notion of consistency between meta-attributes in sources and view. The administrator vouches for the correctness of the translator’s notion of consistency, based on an understanding of meta-attribute and query semantics.

Depending on the organizations’ political relationship, administrators may deviate from the translator’s suggestion. In particular, where there is an ordering relation (partial or total), organizational policy might declare one direction of deviation to be safe, and tell administrators to deviate only in that direction. For example, it would be considered safe if the warehouse grants fewer permissions than would be consistent with sources, and guarantees less accuracy than the sources claim.

The approach has a hidden strength – separation of concerns. When proposing a change, or deciding to deviate from consistency, administrators need look only at their own schema. Meanwhile, translation takes place within the relatively formal and often automatable environment provided by the theory of consistency.

3.3 Actions
An action consists of an invocation of a translator (upward or downward from some set of tables), a recipient, and instructions on how the resulting meta-attribute values should be used. The instruction information has two parts.

The first part describes how the translated data is applied to the system. An invoker doing a private “what if” analysis would specify that the resulting values would be displayed on his own desktop. If the invoker has sufficient certainty and authority, the instructions might run an automated installer on the target schema. If automatic installation is not feasible, the instructions might create an item in the recipient data administrator’s work queue. Other times the resulting values might go into a “proposals” queue in a negotiation system.

The second part describes the mode in which the system should interpret the meta-attribute value. For example, a constraint predicate can be interpreted as describing the current state (descriptive), giving a condition that all updates must satisfy (prescriptive),
or stating how the data ought really to be (normative). Each of these share much (if not all) of their translation logic. Such modes allow us to create a family of meta-attributes that share a core meaning and a translation algorithm, but differ in other respects.

The scenarios in Section 2 illustrated communication such as commands, requests, and proposals between schemas. All of these can be modeled as actions that invoke a translator on meta-attributes. For example, the IS dept controls the values of the meta-attribute Rights on its source tables, and assigns values to POLICY.Rights and ACCIDENT.Rights directly using grant commands. These values are then translated to values for CUSTOMERS.Rights and STATS.Rights. The Marketing department prefers automated administration, and allows invocation of an installer script on its CUSTOMERS table. However, the warehouse administrators, who wish to reduce system load, restrict the rights proposed by IS and accept only users who are also in the groups Actuary or Auditor.

4 Computing Translated Meta-Attribute Values

We have seen that sending a message from one party to another may require translating meta-attributes from the sender’s schema to the recipient’s schema. This section considers the problem of how to generate those translations, i.e., how to compute the value of a meta-attribute in the recipient’s schema, using values for the meta-attribute in the sender’s schema.

One possibility is to have a technologist provide translation logic, possibly as part of the view. But it is infeasible to write and maintain large numbers of such programs (one per meta-attribute instance), and the investment might not be justified. Instead, we opt for a simple theory that lets us automate many important cases.

Section 3 introduced the concepts of bottom-up translation (when the sender is a source) and top-down translation (when the sender is a view). Bottom-up translation is more fundamental, and is covered in this section. Top-down translation involves ambiguities reminiscent of view update [Kel86], and remains an open problem.

4.1 Query-Insensitive Meta-Attributes

Given source meta-attribute values and an SQL view definition, the problem is to compute meta-attribute values for the view. For some meta-attributes, this computation can be complex, and can depend on the operations used in the SQL definition, the order in which they are used, and perhaps statistical assumptions or organizational policy.

For example, consider Precision from Section 2. Suppose that all DamageAmt values are within $100 of the actual value, i.e., ACCIDENT.DamageAmt.Precision = 100. What then should be the translated value of STATS.SumDamageAmt.Precision? To handle the worst case, the value should be $100*N, where N is the number of records in the aggregated group. If one assumes statistical independence, N should be replaced by squareRoot(N).
The remainder of this paper deals with a subset of meta-attributes that have a much simpler behavior—those meta-attributes whose translated value is insensitive to details of the view query. That is, the computation of these meta-attributes is determined solely by the meta-attribute values on the data used in the view computation.\(^2\)

It turns out that many types of metadata fall into this category. For example:

- **Security permissions.** One has the right to execute a SQL query if one has the right to access each source table, regardless of the query details. If each permission is represented by the access predicate \(p_i\), then the computed value at the view is \(\text{AND}(p_i)\).

- **Timestamps.** Suppose each source table has a timestamp meta-attribute that means “this data is guaranteed up to date, as of this timestamp”. Then the computed view meta-attribute should receive the worst (i.e., the minimum) timestamp of any of its inputs.

- **Quality.** Data may be kept at several quality levels, representing successive levels of care (e.g., raw, reviewed, confirmed, warranted…). The computed value at a view is the worst level of any of its source values.

- **Configuration Management.** A software development environment might provide a partially-ordered set of release levels. The release level of a view table is the greatest lower bound of the release levels at the source.

- **Source attribution.** Each source table has a meta-attribute listing the origins of its data. The view table’s meta-attribute would be the union of the source lists.

- **Pricing.** Suppose each source table has a meta-attribute giving the cost of accessing it. Then the cost associated with the view is the sum of these costs.

The common factor in the above examples is that each defines an operation (denoted \(\land\) in section 4.1.3) on table \(^3\) meta-attributes \(\{T_i.M\}\) such that the following translation formula is justified: the result metadata for \(Q(T_1, \ldots T_n) = T_1.M \land T_2.M \land \ldots \land T_n.M\).

### 4.2 The Opportunity to Exploit Query Equivalence

In each example above, we suggested a translation algorithm that knows nothing about the view definition, other than the granules used to compute the view. Suppose we assume that the metadata environment also knows how to rewrite SQL queries to equivalent queries (either on its own or by interacting with the query processor). Each such rewrite might yield a different translation. We would like to pick the one that yields the most favorable meta-attribute values.

Use of such rewrite-generated metadata is justifiable in each of the above examples. For example, consider security, where the desired effect is that one should have Read permission on \(V\) iff one has enough Read accesses to compute \(V\) from the base tables.

\(^2\) We are seeking a first, simple case that would be profitable to implement. Afterward, a system that automated translation for, say, Precision and a few other important meta-attributes through a few common query operators could be quite useful, especially if it were extensible to accommodate new knowledge.

\(^3\) The same approach seems to work for columns and other granules. [Ros98a] contains some partial results.
Suppose Q' uses fewer tables than Q (e.g., by exploiting integrity constraints to eliminate irrelevant data). Then since V was computable by Q', it can be read by anyone who satisfies the permissions implied by Q'. In general, if Q has equivalent queries \{Q_1, \ldots, Q_n\}, then the permissions at the view should be the OR of the permissions implied by each Qi.

The other meta-attributes above behave similarly. For timestamps, we would use whichever equivalent query gave a later timestamp. For quality level, the resulting level is the maximum of the computed levels of the equivalent queries. For source attribution (e.g., for paying for rights) one could provide an expression describing the alternative sources (an OR of the various attributions), so the system could determine at run-time which has minimum cost.

In each case, there is an operation (lattice-join, denoted ∨ in 4.1.3) that can be applied to the metadata from the various alternatives to compute the combined result.

One form of query rewrite that is particularly useful is to use (local) views. In particular, if a source has more favorable meta-attribute values for a portion of a table, then it could create a view and assign those meta-attribute values to it.

For example, consider the view table CUSTOMERS from Section 2. This is frequently-used information, and the IS department has guaranteed that it has accuracy = High. Now a direct mail application written over POLICY, which could be rewritten using CUSTOMERS, can infer that its data will be High quality.

In general, many databases have subsets that have special properties, but are not entire tables or columns. If one defines a view that captures this subset, one can attach metadata to describe the properties. Using query rewrite, one may be able to infer additional properties of a user’s view [Mot96].

### 4.3 Lattices for Meta-Attribute Computation

We present a model that abstracts the examples above. To simplify notation, we assume the underlying database is relational, and metadata vectors are attached relational tables. We are convinced that the approach can be adapted to other granularities (e.g., column or cells) and to expressions over arbitrary data objects.

We require metadata values to form a lattice [Dav91]. A lattice is a partial order such that the least upper bound and greatest lower bound of any two values always exist. If x and y are lattice values, then \(x \land y\) (“x meet y”) returns the greatest lower bound in the partial order, and \(x \lor y\) (“x join y”) returns the least upper bound.

In all of our examples, the following distributive axiom is also satisfied:

\[
x \land (y \lor z) = (x \land z) \lor (y \land z)
\]

for all x, y, z.
A lattice satisfying this axiom is called a distributive lattice. All lattices in this paper are distributive. The distributive law enables a sort of dynamic programming (as in query optimizers); one can combine rewrites of subqueries before enumerating rewrites for larger queries; see Section 4.1.4.

In the previous subsection, each example meta-attribute required two operations: an operation that computed a view value from source values; and an operation that combined values from alternative equivalent queries. These operations correspond to lattice meet and join, respectively.

Thus when a meta-attribute is defined, its definer must say whether its translation semantics can be described by a lattice. If so, she must specify appropriate meet and join operations. For example, if meta-attribute Rights is implemented using access predicates, its meet is AND and its join is OR; if it were implemented using user lists, then its meet would be INTERSECT and its join would be UNION. For the meta-attribute Timestamp, the meet is MIN and the join is MAX.

We can now define the bottom-up translation function for a given meta-attribute value. Observe that our example meta-attributes generally provide a bound on some aspect of their data granule. The bound on that aspect of the results is computed from the inputs’ bounds (lattice meet). It need not be tight. Second, if one has alternative bounds that all hold, one can choose the most favorable (or otherwise combine them -- lattice join).

Thus we require the meta-attributes to satisfy two axioms:

• Every query result must satisfy the assertion corresponding to the “meet” of all input meta-attribute values.

  That is, let Q be an SQL query mentioning tables \{T1, \ldots, Tn\}, and let M be a meta-attribute defined for \Ti. Define \( Q.M = T1.M \land T2.M \land \ldots \land Tn.M \). Then \( Q(T1, \ldots, Tn) \) must satisfy \( Q.M \).

• Use of an equivalent query also gives a correct assertion. Let V be a view table defined by the SQL query Q, and let M be a meta-attribute defined on the source tables of Q. Then the value of V.M is defined by \( V.M = \lor (Qi.M) \) for each query Qi equivalent to Q.

One can treat the above formula as a way of generating a translator. For whatever set of equivalent queries one finds, one gets a useful assertion. As one improves the query rewriter, one may make the bound tougher.

### 4.4 Performance Issues

The computation of a meta-attribute is defined by considering all possible queries equivalent to the view definition. Assuming that the lattice is distributive, one can use dynamic programming to enumerate all the rewritten forms, implicitly. In particular, if you have two ways to compute a subresult, you can take their lattice join (i.e. \( \lor \)) at that time, and use it in all expressions that employ the subresult.
One could achieve this effect by a form of query rewrite, which generates the alternative computations and then employs a “choose one” operator to compute the result. (Such operators are used in query optimization, to allow optimization contingent on run-time arguments.) For metadata computations, choice maps to the lattice’s join operation.

We observe that in practice, a system need not feel obligated to compute all equivalent queries. If only some (or no) equivalent queries are found, then the computed meta-attribute value will be a low approximation to the defined value. That is, if $c$ is the computed value and $d$ is the defined value, then $c \leq d$.

Although our use of query equivalence is quite different from that of a query processor, they have a common need – to enumerate equivalent queries. Ideally, we want this commonality to be shared. For example, the optimizer could show its rewrites to a metadata tool (though not to end users). It then could extend the transforms as needed. Research results for query rewrite should certainly be able to be used for both purposes, i.e., we don’t need a separate research stream.

### 4.5 Related Work

The work that seems closest to ours is [Mot96]. It prototyped a clever metadata calculus and calculation algorithm, which could answer the question (expressed in our vocabulary) "could the query result be computed solely from tables having exactly the given meta-attribute values"? We have given additional examples, can compute new meta-attribute values, and are not restricted to Select/Project/Join queries.

Our reliance on query rewrite is a mixed blessing. It lets us exploit a fast-advancing field, and might allow exploitation of rewrite rules provided for new query operators. We need not implement a separate metadata calculus. On the other hand, it introduces a dependency on external services not present in [Mot96].

### 5 Summary and Open Problems

This paper describes the challenges of administering multi-tier data. Using scenarios, we gained an overview on the functionality one needs in an administration environment. The basic goal is to allow each administrator to work within their own schema, without tracing through derivation logic. Today’s tools give almost no help in coordinating metadata across schemas, so a tool that automated part of the task would seem a useful advance. A concept demonstration is available at [Ros98a].

We explored ways to minimize the number of translators to be written, and identified six kinds of metadata where translators could be generated automatically. We used a lattice to abstract these cases. Rather than create a new metadata calculus, we built over the mature theory of query equivalence.

Many problems have been left open. First, the theory needs to be extended to top-down translation, and to derivation-sensitive meta-attributes. Preliminary research in this
direction appears in [Ros98b]. Second, we need to verify our conviction that the approach can be adapted to other granularities (e.g., column or cells) and to expressions over arbitrary data objects. We also need to investigate how to take advantage of the rewrite rules that are already captured in query processors. Most important, but beyond our resources, there is need to create a real tool, and to test it with real users.

6 References


