A Relational Algebra for Data/Metadata Integration in a Federated Database System

Catharine Wyss
Computer Science Department
Indiana University
crood@cs.indiana.edu

Dirk Van Gucht
Computer Science Department
Indiana University
vgucht@cs.indiana.edu

ABSTRACT

The need for ***interoperability*** among databases has increased dramatically with the proliferation of readily available DBMS and application software. Even within a single organization, data from disparate relational databases must be integrated. A framework for interoperability in a federated system of relational databases should be inherently ***relational,*** so that it can use existing techniques for query evaluation and optimization where possible and retain the key features of SQL, such as a modest complexity and ease of query formulation. Our contribution is a LOGSPACE relational algebra, the *Meta-Algebra* (MA), for data/metadata integration among relational databases containing semantically similar information in schematically disparate formats. The MA is a simple yet powerful extension of the classical relational algebra (RA). The MA has a natural declarative counterpart, the Meta-Query Language (MQL), which we briefly describe. We state a result showing MQL and the MA are computationally equivalent, which enables us to algebratize MQL queries in fundamentally the same way as ordinary SQL queries. This algebratization in turn enables us to use MA equivalences to facilitate the application of known query optimization techniques to MQL query evaluation.

Categories and Subject Descriptors

H.2 [Information Systems]: Database Management; H.2.3 **[Database Management]:** Languages-Query ***Languages;*** H.2.5 **[Database Management]:** Heterogeneous Databases; H.2.1 **[Database Management]:** Logical Design—*Data models, schema and subschema;* **H.2.4 [Database Management]:** Systems-Query ***processing, relational databases, distributed databases***

General Terms

Languages

Keywords

Database integration, database schema integration, federated database systems, Federated Information System (FIS), interoperability, metadata, metaquery, multidatabase, query languages, relational algebra, schema transparency

CIKM'01, November 5-10, 2001, Atlanta, Georgia, USA.

1. INTRODUCTION

The problem of integrating heterogeneous data sources has grown in importance within the last two decades. One reason for this is the rise in availability of web-based data sources, another is the proliferation of readily available DBMS and application software. Even within a single organization, data from disparate sources must be integrated. Fortunately, there are a wide variety of tools and methodologies available to address the need for integrating heterogeneous data, although many issues remain unresolved [10]. Solutions for integrating heterogeneous data often rely on multi-tiered conceptual models for integrating data. Examples include mediator systems, federated database systems, metadata repository systems and description-logic based systems [2, 7, 8, 11, 12, 14, 15,261.

In this paper, we consider a sub-problem of the data integration (and interoperability) problem: that of achieving schema transparency in a federated system of autonomous relational databases (RDBMSs). We assume that component databases contain semantically homogeneous information in structurally heterogeneous, relational formats. The goal within such a federation is to provide a sophisticated end user (one who knows SQL) with a uniform query language providing ***schema transparency.*** Each database in the underlying RDBMSs has a relational schema detailing the relations in the database, and their attributes; this information is stored in a centralized ***metadata dictionary.*** An end-user should not be expected to have detailed knowledge of the contents of the metadata dictionary, nor should the user be obliged to resolve any logical conflicts among the databases that arise from schematic discrepancies. This is the notion of *schema transparency* [3]. The user should be able to query constituent databases in the federation without prior knowledge of their exact schemas. Such a facility would be useful in schema browsing systems [16] or web-based interfaces.

Another important application of a schema transparent query language would be to provide sophisticated support to federation database administrators (DBAs), allowing the creation ofefficient metadatabase front ends, and assisting in the definition and implementation of wrappers, mediators, and integrating views. Ideally, such a language would be ***downward compatible*** with SQL, supporting the portability of legacy code. Also, the language should share some of the key features of SQL, such as ease of query formulation, sufficient expressiveness, a modest complexity (LOGSPACE), and support for aggregation [14].

The primary contributions of this paper are as follows.

1. Our main contribution (§3) is an extended relational algebra supporting schema transparency and data / metadata integra-

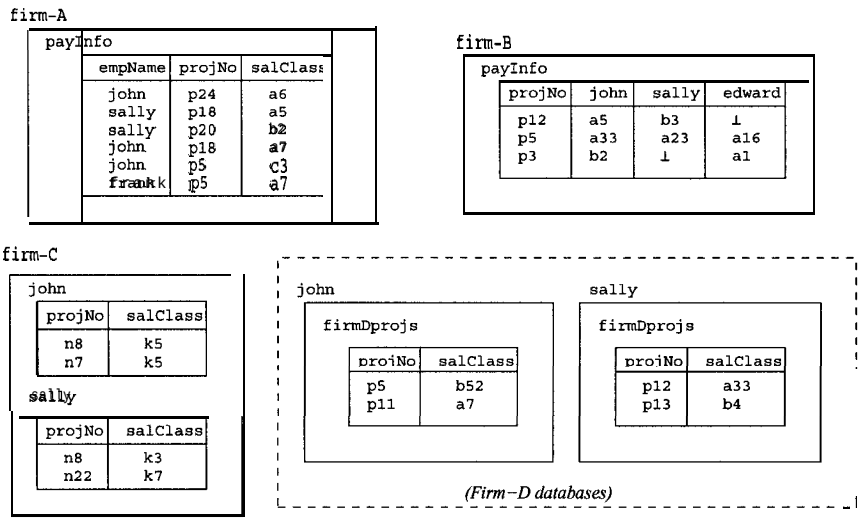

Figure 1: The `consultants` Federation.

tion, the *Meta-Algebra* (MA). The meta-algebra is based on the idea of adding *dereference* capabilities to RA operators.

2. In §4 we describe a declarative query language, *Meta-Query Language* (MQL) for restructuring relational data within a federation of RDBMSs. MQL allows data to be promoted to attribute, relation or database names, and metadata to be demoted to data. In §4.2, we illustrate the equivalence of the MA and MQL with an example. The equivalence is based on a transformation process whose core is fundamentally similar to that between SQL and the RA.

3. The MA facilitates known techniques for query evaluation and optimization. Query evaluation within the MQL/MA framework directly parallels SQL/RA query evaluation in a natural way, as indicated in §5.

Finally (§6), we indicate in more depth how our work improves on existing approaches and give directions for further research.[1]

2. FORMAL PRELIMINARIES

Let **dom** be a countably infinite set of alphanumeric strings, providing both data and metadata for our federation. Elements of **dom** begin with a lowercase letter and appear in typewriter font (for example, $\mathtt{x_1}$, `relName`).[2] We allow $\perp \in$ **dom** to appear as data but not as metadata. It is an artifact of our framework that we will now need *meta-metadata*, in order to distinguish metadata that is a formal component of our framework and metadata that comes from the constituent relational databases. For this, we use a set of distinguished constants, $\mathfrak{D} = \{a, \rho, \delta\}$ that will be used as column labels within the meta-algebra to distinguish the meta-level information used by the MA operators. We assume $\mathfrak{D} \cap$ **dom** $= \emptyset$.

Formally, a ***federation,*** $\mathcal{F}$, is a finite set of ***databases.*** Each database is a finite set of ***relations*** and each relation has a ***schema,*** $\mathcal{S}$ C **dom,** denoting attribute names. Each relation is a finite set of tuples, each tuple denoted by $\langle a_1 : x_1, \ldots . a_k : x_k \rangle$ where al, ... $a_k \in \mathcal{S}$ and $x_1, \ldots, x_k \in$ **dom.** Each metadata query presupposes a fixed federation that provides the context for the query. In this federation, each relation is assumed to have a uniquepathname, which is denoted by a tuple of the form $\langle \delta :$ `dbName`, p : `relName`$\rangle$, where $\delta, \rho \in \mathfrak{D}$. Pathnames are abbreviated as `dbName: :relName`, following the SchemaSQL convention [14, 16].

Consider the consultants federation (figure 1), illustrating varying methods of storing payroll information about consultants. John and Sally have each consulted for 4 different firms, and similar information is stored within these firms. Firm A stores salary information in a single relation, `firm-A: : payInfo`. Firm B uses a single relation as well (`firm-B : :payInfo`), however consultant names appear as attributes in this relation. Firm C uses several relations, each named after a consultant (for example, the relations `firm-C::john` and `firm-C::sally`). Firm D uses a separate database for each consultant; information within these databases is stored in the relation `f irmDproj s`.

Formally, a ***metaquery,*** $\mathcal{Q}$, is a second order transformation mapping a federation to a federation, $Q : \mathcal{F} \mapsto \mathcal{F}'$. "What is known about John in the `consultants` federation?" is a canonical **meta**-query. The result of this query is the subset of the consultants federation referring to the atom j `ohn`.

3. META-ALGEBRA

The Meta-Algebra (MA) extends the classical Relational Algebra (RA) with the power to query and interchange data and metadata. This ability requires extending the RA to query and utilize relational objects ***and*** their names. In effect, relation pathnames can be "dereferenced" to obtain the relation named.

3.1 Basic Terms

Basic terms in the **meta** algebra (MA) are (i) relation variables, denoted by identifiers in italics beginning with an upper-case letter (for example, $R_1, R_2, \ldots$) or (ii) **pathname** relations (for example $\{$`firm-A::payInfo`$\} = \{\langle \delta :$ `firm-A`, p : `payInfo`$\rangle\}$).

[1] For lack of space, all proofs are omitted.

[2] In general, **dom** may contain other elements (for example, numbers); we assume a uniform domain so as to simplify presentation.

δ(consultants) =

δ
firm-A
firm-B
firm-C
john
sally

$\rho(\delta$(consultants)) =

δ	ρ
firm-A	payInfo
firm-B	payInfo
firm-C	john
firm-C	sally
john	firmDprojs
sally	firmDprojs

α({firm-B: :payInfo}) =

δ	ρ	α
firm-B	payInfo	projNo
firm-B	payInfo	john
firm-B	payInfo	sally
firm-B	payInfo	edward

Figure 2: Metadata interface operators example.

3.2 Metadata Interface

The MA contains three operators used to extract metadata from the federation's metadata dictionary.[3] Recall δ, ρ, and α are special constants denoting relation columns containing meta-metadata (§2). We overload these symbols to also denote the operators δ, ρ and α, which return the database names, relation path names, and attribute names of their argument, respectively. Figure 2 illustrates the interface operators as applied to the federation of figure 1.

3.3 Extension/Intension Operators

The MA is essentially relational. However, since metadata is allowed as first class data, a basic expression can now represent either a pathname or the actual relation object. Only one of these is intended in a particular MA expression, thus our algebra contains two operators $\dot{\cdot}$ (extension) and $\bar{\cdot}$ (intension) to distinguish whether a relation object or pathname is intended, respectively.[4] The operator $\dot{\cdot}$ is also termed the *dereference* operator.

Given a basic expression, R, define $\bar{R}$ as the formal *pathname* of R and $\underline{R}$ as the actual relation object, R. For example, if dbl : : rell is the relation {(a : x, b : y), (a : u, b : v)}, then

$\overline{\text{dbl: :rell}} = \{\text{dbl: :rell}\} = \{\langle \delta : \text{dbl}, \rho : \text{rell}\rangle\}$, and

$\underline{\text{dbl: :rell}} = \{\langle \text{a} : \text{x}, \text{b}:\text{y}\rangle, \langle \text{a} : \text{u}, \text{b} : \text{v}\rangle\}$.

3.4 Extension Projection

One of the novel operators in the relational base of the MA is the *Extension Projection* operator. This operator is denoted $\underline{\Pi}$ and allows dereferencing the fields in a tuple as attribute names, obtaining a new column of data. This new column must be named explicitly within the extension projection.

Definition 1. (Extension Projection) Given an input relation, R, with schema $\mathcal{S} = \{A_1, \ldots, A_n\}$ and $x \in \mathbf{dom} - \mathcal{S}$, the *Extension Projection of R on column A_i producing column* x, denoted $\underline{\Pi}^{\text{'x'}}_{A_i}(R)$, is defined as follows. Whenever $\langle A_1 : t[A_1], \ldots, A_n : t[A_n]\rangle \in R$, then $\langle A_1 : t[A_1], \ldots, A_n : t[A_n], \text{x} : t[t[A_i]]\rangle \in \underline{\Pi}^{\text{'x'}}_{A_i}(R)$ where $t[t[A_i]] = \perp$ in case $t[A_i] \notin S$.

Example 1. Figure 3 shows the relation proj : : ex and the result of $\underline{\Pi}^{\text{'x'}}_{\text{a}}$(proj : :ex).

[3] These operators first appeared in [15].

[4] *Extension* and *intension* are terms from the philosophy of language. A crucial distinction made when using language to discuss language itself is between the real-world object a term refers to (its *extension)* and any internal or non-corporeal properties the term suggests (its *intension).* We recapitulate this distinction here.

a	b	c	d	e	f
a	b	c	a	b	c
b	a	a	x	b	b
f	a	c	g	e	i
x	c	c	c	c	c
e	x	y	x	y	x

a	b	c	d	e	f	x
a	b	c	a	b	c	a
b	a	a	x	b	b	a
f	a	c	g	e	i	i
x	c	c	c	c	c	$\perp$
e	x	y	x	y	x	y

Figure 3: Relations pro j : : ex **and** $\underline{\Pi}^{\text{'x'}}_{\text{a}}$(pro j : : ex).

3.5 Relational Selection

The MA allows ordinary relational selection, σ. However, note that extension projection allows *us* to select tuples *indirectly,* based on the result of a prior extension projection. This situation is quite common in the MA, and we introduce the macro notation $\sigma_{A_i = A_j}(R)$ to stand for the operation $\Pi_{A_1,\ldots,A_n}\left(\sigma_{x=A_j}(\underline{\Pi}^{\text{'x'}}_{A_i}(R))\right)$.

3.6 Renaming Columns and Relations

As in the relational algebra, we will sometimes need to rename the output of an MA expression. We introduce the renaming operator, $|^{\text{d::r}}_{\text{a}_1 \to \text{b}_1, \ldots, \text{a}_n \to \text{b}_n}$ which allows us to assign a new pathname to a relation and rename the columns $\mathbf{a_1}, \ldots, \mathbf{a_n}$, as $\mathbf{b_1}, \ldots, \mathbf{b_n}$, respectively, where $\mathbf{a_i, b_i \in dom\ (1 \leq i \leq n)}$.

3.7 Relational Projection

We retain ordinary relational projection, Π, in the MA. However, we allow negative projection conditions of the form $\neg A_i$. For example, $\Pi_{\neg \text{a}, \neg \text{d}}$(proj : : ex) removes columns a and d from proj : : ex.

It is also convenient to extend the projection notation so that we may rename columns directly within the projection. For example, $\Pi^{\text{'b}_1\text{'},\ldots,\text{'b}_n\text{'}}_{\text{a}_1,\ldots,\text{a}_n}(R)$ is shorthand for $\Pi_{\text{a}_1,\ldots,\text{a}_n}(R)|_{\text{a}_1 \to \text{b}_1, \ldots, \text{a}_n \to \text{b}_n}$.

Once we can rename directly within a projection, we can introduce a macro for bundling the extension projection into ordinary projections. We use the notation $\Pi^{\text{'b'}}_{\underline{\text{a}}}(R)$ as shorthand for $\underline{\Pi}^{\text{'b'}}_{\text{a}}(R)$. Thus, we may include extension projections in a list of ordinary projections (stipulating that extension projections are performed first). For example,

$$\Pi^{\text{'u'},\ldots,\text{'x'}}_{\text{a}, \underline{\text{b}}, \ldots, \underline{\text{d}}, \text{x}}(R) = \Pi^{\text{'u'},\text{'v'},\text{'w'},\text{'x'}}_{\text{a}, \text{v}, \text{c}, \text{x}}\left(\underline{\Pi}^{\text{'v'}}_{\text{b}}\left(\underline{\Pi}^{\text{'x'}}_{\text{d}}(R)\right)\right).$$

3.8 The Transpose Operator

The *Transpose Operator,* τ, allows a dynamic schema to be created where the new schema values are taken from a column of data. The contents of the new columns are obtained from a second column of data in the relation being transposed. The net effect is similar to matrix transposition, in that values that were "beside" one another in the two original columns are now "beneath" one another.

Dejnition 2. (*MA Transpose)* Given a relation R with schema $S = \{A_1, ..., A_i, A_j, ..., A_n,\}$, the *transpose of* A_i *on* A_j, $\tau_{A_i}^{A_j}(R)$, is given as follows: (i) the schema of $\tau_{A_i}^{A_j}(R)$ is $\{A_1, \ldots A_i\} \cup \{v : v \in R.A_j\}$ and (ii) whenever $\langle A_1 : \mathtt{a_1}, \ldots A_i : \mathtt{a_i}, A_j : \mathtt{a_j}, \ldots A_n : \mathtt{a_n} \rangle \in R$, then $\langle A_1 : \mathtt{a_1}, \ldots A_i : \mathtt{a_n}, \mathtt{a_j} : \mathtt{a_i} \rangle \in \tau_{A_i}^{A_j}(R)$.

Example 2. The relation $R = \tau_{\mathtt{salClass}}^{\mathtt{empName}}(\mathtt{firm\text{-}A::payInfo})$ is shown in figure 4 (below). Note that the contents of `salClass` are identical with the column x in the extension projection $\underline{\Pi}_{\mathtt{empName}}^{\text{'x'}}(R)$. This holds generally, so that τ and $\underline{\Pi}$ are roughly inverses.

empName	projNo	salClass	john	sally	frank
john	p24	a6	a6	⊥	⊥
sally	p18	a5	⊥	a5	⊥
sally	p20	b2	⊥	b2	⊥
john	p5	c3	c3	⊥	⊥
frank	p5	al	⊥	⊥	a7

Figure 4: Relation $\tau_{\mathtt{salClass}}^{\mathtt{empName}}(\mathtt{firm\text{-}A::payInfo})$.

3.9 Cartesian Product

The Cartesian product operator takes two well-formed *basic relational terms* as arguments and returns the Cartesian product of the relations given by those arguments.

Definition 3. T is a *basic relational term* of the MA in case: (a) *T* is a relation variable, enclosed in an extension or intension operator, (b) *T* is a relation pathname, enclosed in an extension or intension operator, or (c) *T* is a well-formed 6, ρ, or α term.

If P is a finite product of basic relational terms and $R_1, \ldots, R_n$ is a list of distinct relation variables, then $P(R_1, \ldots, R_n)$ indicates that the relational variables of P are included among $R_1, \ldots, R_n$.

As an example of the Cartesian product operator,

$$\rho(\{\langle \delta : \mathtt{firm\text{-}C} \rangle\}) \times \underline{\mathtt{firm\text{-}C::john}} =$$

δ	ρ	projNo	salClass
firm-C	john	n8	k5
firm-C	john	n7	k5
firm-C	sally	n8	k5
firm-C	sally	n7	k5

3.10 The Map Operator

Thus far, our operators have only concerned relations. Yet an MQL query outputs a *federation;* two levels beyond our operators. To achieve this, we use a powerful Map operator to apply a base MA expression to a subset of relations in the federation, obtaining a federation as a result. The action of the Map operator is similar to the extended relational operations for multidatabases, given in [8]. First, we need a formal definition of the relational subset of the MA, the *base MA expressions.*

Dejnition 4. An MA expression, *E,* is a *base MA expression* if it is a finite sequence of transpose, projection, extension projection and selection operations applied to a finite product of basic relational terms, $P(R_1, \ldots, R_n)$.

The idea behind the Map operator is that we will apply a well-formed base MA expression to a set of relations picked out from the federation by the metadata extractors. Let $E(R_1, \ldots, R_n)$ be a well-formed base MA expression having n relation variables. Let $X_1, X_2, \ldots, X_n$ be *variable-free* base MA expressions yielding relations containing valid relation pathnames. The syntax of the Map operator is:

$$\mathfrak{Map}\ (E(R_1, ..., R_n);\ X_1, X_2, ..., X_n;\ N_r, N_d)$$

The idea is that we will apply the base expression E to the relations named by the X_i. The expressions N_r and N_d refer to columns of the Cartesian product within E (these columns may be projected out subsequently), and for each tuple t in this product, the columns referred to by N_r and N_d specify the pathname of the relation the result should go into. The result itself is computed tuple-by-tuple, based on *instantiations* that assign values to the variables $R_1, \ldots, R_n$ from the relations $X_1, \ldots, X_n$.

Example 3. We can now restructure the `firm-C` database into the format off `irm-A,` as follows:

$$\mathfrak{Map}\ (\Pi_{\rho,}^{\text{'empName', 'projNo', 'salClass'}}\ (\underline{R} \times \overline{R});\ \mathtt{p(firm\text{-}C)};\ \text{'payInfo','c2a'}).$$

The instantiation relation in this case is $\rho(\mathtt{f\ irm\text{-}C})$. Thus, the user does not have to know in advance how *many* relations there actually are in `firm-C,` or what they are named. For each of these relations, R, the expression $\underline{R}$ x $\overline{R}$ is computed: note how this pairs the tuples in each relation with the name of the relation. Thus, this relation name can be projected in the result, simultaneously with the information from the relation itself, into a single output relation `c2a : : payInfo`. Each $R \in$ `firm-c` is restructured in turn, and the results are placed in `c2a : : payInf` o.

3.11 Set Union and Difference

Given a finite number of map expressions, $\mathfrak{M}_1, \ldots, \mathfrak{M}_k$, we may take their set union or difference. For example, consider $\mathfrak{M}_1 \cup \mathfrak{M}_2$. Each relation in the output federations of $\mathfrak{M}_1$ and $\mathfrak{M}_2$, has a pathname. If pathnames match between $R_1 \in \mathfrak{M}_1$ and $R_2 \in \mathfrak{M}_2$, a union is formed between the two relations. The schema of $R_1 \cup R_2$ is $S_1 \cup S_2$ where $S_1 = \mathit{schema}(R_1)$ and $S_2 = \mathit{schema}(R_2)$. Existing tuples are then augmented with ⊥ where necessary. Set difference can be handled in a similar way. Note that if $S_1 = S_2$, ∪ and function as in the ordinary RA. The ability to "extend schemas" during ∪ allows one (for example) the capability to perform *roll-ups* naturally when combined with aggregation. MA ∪ and ╍ are properly termed *outer union* and *outer difference.*

3.12 Error Conditions

Error conditions may arise from applying MA operations to relations of improper schemas. In these cases, the operations will produce empty results (of the appropriate schemas). Thus, further operations are well-defined, albeit the final result may be the empty federation. As an example of such error conditions, the metadata interface operators ignore invalid arguments, in some cases returning the empty relation of appropriate schema. For example, there is no relation named foo in any of the databases in the `consultants` federation; hence, in the context of this federation the expression

$$\alpha(\delta(\mathtt{consultants}) \times \{(\mathtt{p:\ foo})\})$$

returns the empty relation of schema $\{\delta, \rho, \alpha\}$.

4. A DECLARATIVE LANGUAGE FOR INTEROPERABILITY

4.1 Meta-Query Language

Meta-Query Language (MQL) is a declarative, SQL-like language for restructuring and querying relations in our federation. A key feature of the syntax of MQL is derived from SchemaSQL [14]: meta-range declarations in the FROM clause. MQL admits four types of range declarations, going beyond tuple variables to include *meta-variables* defined to range over database, relation or column names. Metavariable declarations are distinguished by the "-->" token!

In addition, MQL allows *dynamic output relations* to be defined in the SELECT clause. Following the MA convention, all column projections must be explicitly named; we use the standard AS keyword in the SELECT clause for naming output columns in the case of static output. In the case of dynamic output, where the schema of the output relations is determined at run-time from the input relations, we use the keyword ON instead of AS. Below, we illustrate the power of MQL with sample queries that restructure between the A and B databases of our consultants federation (figures 5 and 6). Note the use of the keywords INTO and WITHIN in the SELECT clause, which give the output relation and database names, respectively. Thus, an MQL query maps a federation to a federation, as does the MA.

A full EBNF for MQL appears in [27].

```
SELECT A.projNo AS 'projNo', A.salClass ON A.empName
          INTO 'payInfo' WITHIN 'a2b'
FROM   firm-A::payInfo A
```

Figure 5: firm-A::payInfo ↦ a2b::payInfo.

```
SELECT B AS 'empName', T.projNo AS 'projNo',
       T.B AS 'salClass'
          INTO 'payInfo' WITHIN 'b2a'
FROM   firm-B::payInfo -> B, firm-B::payInfo T
WHERE B != 'projNo'
```

Figure6: Q2: firm-B::payInfo ↦ b2a: :payInfo

4.2 An Algebratization of MQL

MQL is a declarative language that a federation DBA would use to define the translations among relations in the constituent databases of the federation. MQL can be given a semantics based on this notion, independently of the MA. However, it is also useful to think of the MA as providing the semantics of MQL, a point of view which is valid in light of the following theorem.

THEOREM 1. *(1) For any MQL query, Q, there exists an MA expression $\hat{Q}$ such that for any input federation $\mathcal{F}$, $Q(\mathcal{F}) = \hat{Q}(\mathcal{F})$.*

(2) For any MA expression, M, there exists an MQL query $\hat{M}$ such that for any input federation $\mathcal{F}$, $M(\mathcal{F}) = \hat{M}(\mathcal{F})$.

[5] Although the syntax of MQL range declarations is similar to that of SchemaSQL [14, 16], the semantics differs in that range declarations appearing in the same MQL query are *independent* of one another, as in SQL [27].

Due to space constraints, a proof of the equivalence is omitted here. We illustrate part (1) of the theorem with an example showing the translation from an MQL query to a corresponding MA expression.

```
(SELECT A.projNo AS 'projNo', A.salClass AS 'salClass'
        INTO A.empName WITHIN 'ab2c'
FROM   firm-A::payInfo A)
UNION
(SELECT B.projNo AS 'projNo', B.EmpName AS 'salClass'
        INTO EmpName WITHIN 'ab2c'
FROM   firm-B::payInfo -> EmpName, firm-B::payInfo B
WHERE EmpName <> 'projNo')
```

Figure7: firm-A+finn-B-t firm-C

Example 4. Consider an MQL query that restructures the information in the firm-A and firm-B databases into the format of firm-c (figure 7). This query is composed of two sub-queries, Q_{a2c} and Q_{b2c}. Each sub-query is translated separately using a four-step process.

1. First, relations named in the FROM clause are gathered into the instantiation relations. In our case, we have: X_{a2c} := {firm-A::payInfo} and X_{b2c} :={firm-B::payInfo}.
2. Next, the naming terms for the output relations are created. We have that N_r^{a2c} := empName, N_d^{a2c} := 'ab2c', and N_r^{b2c} := α, N_d^{b2c} := 'ab2c'.
3. Next, the base MA expression that will be mapped to the relations in X_{a2c} and X_{b2c} is created. In this case, we have

$$E_{a2c} := \Pi^{\text{'projNo','salClass'}}_{\text{projNo, salClass}}(\underline{R}) \text{ and}$$

$$E_{b2c} := \Pi^{\text{'projNo','salClass'}}_{\text{projNo, } \underline{\alpha}}(\underline{R} \times \alpha(\overline{R}))$$

   Here, we have only needed projections; an ON expression in the SELECT clause would necessitate transposition.

4. Finally, we assemble the components above into $\mathfrak{Map}$ expressions. We have

$$\widehat{Q_{a2c}} := \mathfrak{Map}(E_{a2c}; X_{a2c}; N_r^{a2c}, N_d^{a2c})$$
$$= \mathfrak{Map}\left(\Pi^{\text{'projNo', 'salClass'}}_{\text{projNo, salClass}}(\underline{R}); \{\text{firm-A::payInfo}\}; \text{empName}, \text{'ab2c'}\right)$$

$$\widehat{Q_{b2c}} := \mathfrak{Map}(E_{b2c}; X_{b2c}; N_r^{b2c}, N_d^{b2c})$$
$$= \mathfrak{Map}\left(\Pi^{\text{'projNo','salClass'}}_{\text{projNo, } \underline{\alpha}}(\underline{R} \times a(R)); \{\text{firm-B::payInfo}\}; \alpha, \text{'ab2c'}\right)$$

The final output query is the MA expression $\widehat{Q_{a2c}} \cup \widehat{Q_{b2c}}$.

The heart of this translation is very similar to that between SQL and the RA, in that the SELECT clause translates to Π terms, the FROM clause range declarations appear inside the Cartesian product, and the WHERE clause corresponds to selection conditions. Thus the MQL/MA framework is useful for interoperability within a federation of RDBMS, since much of what is known about optimizing and implementing SQL carries over to an MQL-based system. In 55, we give specific MA equivalences and data structures.

5. METADATA-QUERY PROCESSING

In this section, we discuss the expressiveness, evaluation and optimization of metaqueries within the MA framework. We begin by stating a result concerning the expressiveness of the Meta-Algebra as a query language.

PROPOSITION 1. *MA* $\subseteq$ LOGSPACE.

The heart of the proof of proposition 1 relies on the tuple-by-tuple evaluation of MA expressions that was indicated throughout §3. The only workspace that is necessary are indices into the input federation: the output is generated a tuple at a time, from the input.

Next, we sketch the evaluation and optimization of MQL queries within a canonical query processing system. An interoperability engine would reside over a federation of relational databases. Since the MA is inherently relational, query evaluation and optimization follow the canonical treatment of RA queries (figure 8).

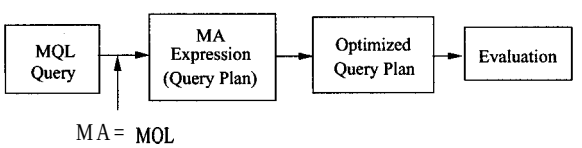

Figure 8: Metadata Query Processing Lifecycle

5.1 MA Evaluation

The evaluation of MA expressions is based on the notion of an *indirect index.* An indirect index is an ordinary relational index (for example, a hash or B+ index) where the search key is indicated *indirectly* within each tuple indexed. As an example, consider an index on $\underline{a}$ of the table proj : : ex (figure 3). Each tuple $t \in$ proj : : ex is indexed on the value of the t [a] field (as opposed to the value of the a field in a normal index). The key values are thus {a, d, y, x}.

Indirect indices allow the usual algorithms for the evaluation of Π, σ, and $\bowtie$ to apply in the case of MA expressions, with straightforward modifications. This means that we can view a base MA expression as an evaluation plan (having a canonical tree form) and optimize accordingly. As with the generation of ordinary indices, query optimizers can create indirect indices only on columns that frequently appear in extension projections or selections, to minimize the additional space required for these indices.

The final stage in evaluating an MQL query is to use the optimized base expression within a $\mathfrak{Map}$ expression: we want to re-arrange the query tree so that not only are intermediate relations small, but sub-trees that must be issued as queries to the constituent databases are grouped so as to minimize network utilization.

5.2 MA Optimization

Many of the RA algebraic equivalences used in query optimization carry over to the MA. However, due to the dynamically generated output schemas that some MA expressions require, some equivalences have well-definedness conditions that can only be evaluated at run-time.

Indirect selection conditions do not interfere with the usual RA equivalences. This is because the MA σ operator does not produce a dynamic output schema (proposition 2). Since the output schema is static, as in SQL, the proofs of certain equivalences carry over directly to the MA (corollary 1).

PROPOSITION 2. *For relation instance r, and selection condition C, schema*$(\sigma_C(r))$ *= schema(r).*

COROLLARY 1. *For relation instances* $\mathbf{r_1}$ *and* $\mathbf{r_2}$,
(1) $\sigma_{c_1 \wedge c_2 \cdots \wedge c_n}(\mathbf{r_1}) \equiv \sigma_{c_1}(\sigma_{c_2}(\cdots(\sigma_{c_n}(\mathbf{r_2}))\cdots))$,
(2) $\sigma_{c_1}(\sigma_{c_2}(\mathbf{r_1})) \equiv \sigma_{c_2}(\sigma_{c_1}(\mathbf{r_2}))$, *and*
(3) $\mathbf{r_1} \bowtie_C \mathbf{r_2} \equiv \mathbf{r_2} \bowtie_C \mathbf{r_1}$

Note that it is not necessarily the case that $\sigma_C(\mathbf{r_1}) \times \mathbf{r_2} \equiv \sigma_C(\mathbf{r_1} \times \mathbf{r_2})$, even if C only mentions fields in $\mathbf{r_1}$. Suppose C mentions field a *indirectly,* for example C := $\underline{a} = b$ where $a, b \in$ *schema*$(\mathbf{r_1})$. If column a of $\mathbf{r_1}$ includes $x \notin$ *schema*$(\mathbf{r_1})$ such that $x \in$ *schema(q),* we may rule out new tuples when we enclose $\mathbf{r_2}$ in the selection. Thus we can distribute selections only in the following case:

LEMMA 1. *Suppose schema*$(\mathbf{r_1}) \cap$ *schema*$(\mathbf{r_2}) = 0$ *and a, b* $\in$ *schema*$(\mathbf{r_1})$. *Then,* $\sigma_{\underline{a}=b}(\mathbf{r_1} \times \mathbf{r_2}) \equiv \sigma_{\underline{a}=b}(\mathbf{r_1}) \times \mathbf{r_2}$ *in case* $\{v : v \in \mathbf{r_1}.a\} \cap$ *schema*$(\mathbf{r_2}) = 0$.

Note that the condition can only be checked at *run-time,* in which case we call it a *dynamic condition.* Dynamic conditions characterize the difference between many MA equivalences and their RA counterparts, in particular those involving MA projection.

PROPOSITION 3. *For a relation instance,* $\mathbf{r_1}$,

(1) Let $a, b, c_1, \ldots, c_n \in$ *schema*$(\mathbf{r_1})$. *Then*

$$\tau^b_{\underline{a}}(\mathbf{r_1}) \equiv \tau^b_{\underline{a}}(\Pi^{c_1,\ldots,c_n}_{c_1,\ldots,c_n}(\mathbf{r_1}))$$

in case $(\{v : v \in \mathbf{r_1}.a\} \cup \{b\}) \subseteq \{c_1, \ldots, c_n\}$. *Here,* $\tau^b_{\underline{a}}(R)$ *is shorthandfir* $\tau^b_{\mathbf{x}}(\underline{\Pi}^{'x'}_{a}(R))$.

(2) Let $a, b, c \in$ *schema*$(\mathbf{r_1})$. *Then*

$$\tau^b_a(\sigma_{c='x'}(\mathbf{r_1})) \equiv \sigma_{c='x'}(\tau^b_a(\mathbf{r_1}))$$

in case $c \notin \{v : v \in \mathbf{r_1}.b\}$.

The dynamic conditions make it more difficult to restructure the tree form of an MA query; however, many optimizations can be saved. For example, we can push selection conditions down the tree, with a goal of both minimizing intermediate relation sizes, and also minimizing the number of tuples obtained from the constituent databases across the network.

6. DISCUSSION AND CONCLUSIONS

6.1 Related Work

To put our work in context, we briefly discuss other approaches that allow data/metadata integration by introducing more powerful query languages.

One of the first such higher-order languages is HiLog [4]. HiLog is a complex object based logic programming language with function terms. There is a single namespace over which terms are defined;

these terms can appear in predicate and attribute positions, supporting various types of metadata querying. One of the major limitations of HiLog is that terms have fixed arity, hence limiting the flexibility needed for true schema independent querying. Dynamic output schemas cannot be generated using a HiLog query.

Soon after HiLog's introduction, the Ross Algebra and SchemaLog were introduced. In [21], Ross introduced an algebra and calculus wherein it is possible to demote relation names to ordinary domain values. A safe fragment of his calculus is proceduralized through equivalence to his algebra. However, it is impossible to promote data to metadata.

SchemaLog [13, 15] is a logic programming language whose syntax is similar to the Interoperable Database Language developed in [12]. Like HiLog, SchemaLog has a second-order syntax but first-order semantics. SchemaLog has been implemented over a single database using a top-down processing technique and novel restructuring operators [1]. Although SchemaLog allows data/metadata integration at all levels, it uses tuple identifiers and supports full recursion, a significant drawback for efficient query processing. Furthermore, logic programming is foreign to many users, who are familiar with SQL-type declarative query languages.

SchemaSQL evolved from SchemaLog [14] to meet the need for an SQL-like metadata query language. However, SchemaSQL did not retain the completeness of SchemaLog for data / metadata integration: only one column of values can become metadata in a restructuring view statement. Furthermore, SchemaSQL performs restructuring exclusively by means of materialized views [14, 16], which do not compose well with other types of restructuring that SchemaSQL is capable of [6, 20].

Neither SchemaSQL nor SchemaLog has an equivalent, compositional, relational algebra; thus, the equivalence between MQL and the MA is a significant step forward.[6] Furthermore, the MA is a principled extension of the classical relational algebra (RA) and does not introduce any unfamiliar restructuring operations. Such operations have been a problem for equivalence proofs, which require parallel syntactic and semantic compositionality in the language and algebra. Meta-query processing in the MQL/MA framework parallels relational query processing in the SQL/RA framework. On the other hand, current implementations of SchemaSQL and SchemaLog require novel physical operators [1, 17] and optimization techniques [1, 17,221.

Frameworks for interoperability within federated information systems contain potential for data / metadata integration. However, in most such frameworks, component schemas have to be known in advance of specifying restructuring mappings (for example, this is the case in the procedural mapping language BRIITY [10]).

Several distinctions are commonly made with reference to federated information systems (FIS) architectures, such as monolithic versus hierarchical [18], centralized versus distributed [18], and coupled (loosely or tightly) versus autonomous [23]. Furthermore, a distinction between multidatabases (decentralized, autonomous relational sources) and federated (schema integrated) databases is often made [19, 24, 25]. We do not view the MQL/MA framework as restricted to a single architecture; rather our platform could be useful for administrators and sophisticated users of any FIS architecture where some degree of global data interoperation is sought among relational databases. Many existing tools for FIS architectures are based on object models. We feel the success of the relational platform calls for an approach based more closely on Codd's relational framework; early precedents for such an approach include [5].

[6] The algebra of [16] covers only a subset of SchemaLog and SchemaSQL. The recast tabular algebra introduced in [17] is PSPACE complete [20], whereas SchemaSQL is LOGSPACE.

6.2 Future Directions

One area of future research would be to investigate *aggregation* in MQL. SchemaSQL supports many types of aggregation, above and beyond that of ordinary SQL [14]. More recently, nD-SQL supports data/metadata integration beyond SchemaSQL and further aggregation capabilities [6]. The aggregation capabilities of SchemaSQL carry over to MQL in a relatively straightforward way. Furthermore, since MQL supports the additional data/metadata integration of nD-SQL, future work exploring just how far aggregation can go in MQL would be informative.

The MA/MQL framework has deep affinities with complex object models; pursuing these would be a profitable avenue of future research. Furthermore, the relationship between the MA and the tabular algebra is of interest, in particular because of the completeness result that states the tabular algebra can perform all possible restructurings on tabular data [9].

Finally, the MA extension operator encapsulates the dereferencing operation used in hypertext and XML links on the world wide web. Since XML is a burgeoning standard for information storage on the web, it would be interesting to explore this similarity with a view toward assimilating XML and relational data in a single, relational framework based on a suitable extension to the MA.

7. ACKNOWLEDGMENTS

Several individuals contributed to earlier versions, including Jan Paredaens, Mehmet Dalkilic and Dennis Groth. This version was immeasurably improved during conversations with Edward Robertson, Chris Giannella and the IU Database Lab. We also thank the anonymous reviewers for helpful suggestions and comments.

8. REFERENCES

[1] A. J. Andrews, L. V. Lakshmanan, N. Shiri, and I. N. Subramanian. On implementing *SchemaLog* – a database programming language. In *Proceedings of the 5th International Conference on Information and Knowledge Management (CIKM '96)*, 1996.

[2] P. A. Bemsten, A. Y. Levy, and R. A. Pottinger. A vision for management of complex models. Technical Report MSR-TR-2000-53, Microsoft Research, 2000.

[3] S. Busse, R.-D. Kutsche, U. Leser, and H. Weber. Federated information systems: Concepts, terminology and architectures. Technical Report 99-9, Technische Universität Berlin, 1999.

[4] W. Chen, M. Kifer, and D. S. Warren. HiLog: A foundation for higher-order logic programming. Technical report, Computer Science Department, SUNY at Stony Brook, 1990.

[5] E. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[6] F. Gingras and L. V. Lakshmanan. nD-SQL: A multi-dimensional language for interoperability and OLAP. In *Proceedings of the 24th VLDB Conference*, 1998.

[7] F. Gingras, L. V. Lakshmanan, I. N. Subramanian, and D. Papoulis. Languages for multi-database interoperability. In *SIGMOD Tools Demo*, 1997.

[8] J. Grant, W. Litwin, N. Roussopoulos, and T. Sellis. Query languages for relational multidatabases. *VLDB Journal*, 2:153–71, 1993.

[9] M. Gyssens, L. V. Lakshmanan, and I. N. Subramanian. Tables as a paradigm for querying and restructuring. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS '96)*, 1996.

[10] T. Harder, G. Sauter, and J. Thomas. The intrinsic problems of structural heterogeneity and an approach to their solution. *VLDB Journal*, 8(1):25–43, 1999.

[11] R. Jakobovits. Integrating autonomous heterogeneous information sources. Technical report, Computer Science Department, University of Washington, 1997.

[12] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *Proceedings of ACM SIGMOD*, 199 1.

[13] L. V. Lakshmanan, F. Sadri, and I. N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *DOOD '93*, 1993.

[14] L. V. Lakshmanan, F. Sadri, and I. N. Subramanian. SchemaSQL – a language for interoperability in relational multi-database systems. In *Proceedings of the 22nd VLDB Conference*, 1996.

[15] L. V. Lakshmanan, F. Sadri, and I. N. Subramanian. Logic and algebraic languages for interoperability in multidatabase systems. *Journal of Logic Programming*, 32(2):101–149, 1997.

[16] L. V. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL: An extension to SQL for multi-database interoperability. http://www.uncg.edu/~sadrif

[17] L. V. Lakshmanan, F. Sadri, and S. N. Subramanian. On efficiently implementing SchemaSQL on a SQL database system. In *Proceedings of the 25th VLDB Conference*, 1999.

[18] S. Mehrotra, H. F. Korth, and A. Silberschatz. An architecture for large multidatabase systems. Technical Report CS-TR-92-50, Department of Computer Science, University of Texas at Austin, 1993.

[19] S. B. Navathe and M. J. Donahoo. Towards intelligent integration of heterogeneous information sources. In *Proceedings of the 6th International Hong Kong Computer Society Database Workshop*, 1995.

[20] C. M. Rood, D. Van Gucht, and F. I. Wyss. MD-SQL: A language for meta-data queries over relational databases. Technical Report TR-528, Computer Science Department, Indiana University, 1999.

[21] K. A. Ross. Relations with relation names as arguments: Algebra and calculus. In *Proceedings of the I I th ACM Conference on Principles of Database Systems (PODS '92)*, 1992.

[22] F. Sadri and S. B. Wilson. Implementation of SchemaSQL – a language for relational multi-database systems. Technical report, Department of Mathematical Sciences, University of North Carolina at Greensboro, 1997.

[23] A. P. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In *Proceedings of the IFIP DS-5 Conference on Semantics of Interoperable Database Systems*, pages 283-3 12, 1992.

[24] M. Tresch and M. H. Scholl. A classification of multi-database languages. Technical Report 94-07, University of Ulm Faculty of Computer Science, 1994.

[25] M. W. W. Vermeer and P. M. G. Apers. On the applicability of schema integration techniques to database interoperation. In *International Conference on Conceptual Modeling/the Entity Relationship Approach*, pages 179-l 94, 1996.

[26] J. Widom. Research problems in data warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM '95)*, 1995.

[27] C. Wyss, F. Wyss, and D. Van Gucht. Augmenting SQL with dynamic typing to support interoperability in a relational federation. In *Proceedings of the 4th International Conference on Engineering Federated Information Systems (EFIS 2001)*, 200 1.