

Accessing Heterogeneous Data Through Homogenization and Integration Mediators

Ling Ling Yan M. Tamer Özsu Ling Liu
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta, Edmonton, Alberta, T6G 2H1
{ling, ozsu, lingliu}@cs.ualberta.ca

Abstract

The AURORA mediator system employs a novel 2-tier, plug-and-play mediation model that is designed to facilitate access to a large number of heterogeneous data sources. This paper describes AURORA's mediation model and a suite of techniques used by a specific AURORA mediator, AURORA-RH. This suite includes a mediation methodology provided via an interactive mediator author's toolkit (MAT), a mediation enabling algebra, a query rewriting algorithm, and transformation rules that facilitate query optimization.

1. Introduction

The advent of the Internet gives rise to new types of applications such as electronic commerce and virtual enterprise that require integrated access to large number of heterogeneous data sources around the globe. Much is known about schema integration but the impact of this process on query processing efficiency is seldom discussed. In the Internet age, it is crucial that this impact be considered. The AURORA project builds integrated access to heterogeneous data sources based on a novel 2-tier, plug-and-play style mediation model shown in Figure 1. Data sources to be accessed via an application view V are first *homogenized* in regard to V and then *integrated* into V . Homogenization removes “deviations” of a source from V . Integration is a registration mechanism that relates a homogenized source into V . Homogenization and integration are performed by specialized mediators as in Figure 1. AURORA Mediator Author's Toolkits (MATs) provide homogenization and integration constructs and mandate *mediation methodologies*. Each mediator employs a *Mediation Enabling Algebra* (MEA) for manipulating heterogeneous data and for query optimization. With MEAs, AURORA identifies the

impact of mediation and takes it into account in query processing. This paper describes AURORA's mediation model and the suite of techniques used by a AURORA mediator, AURORA-RH, including a homogenization methodology, a MEA and query modification and optimization algorithms enabled by this MEA.

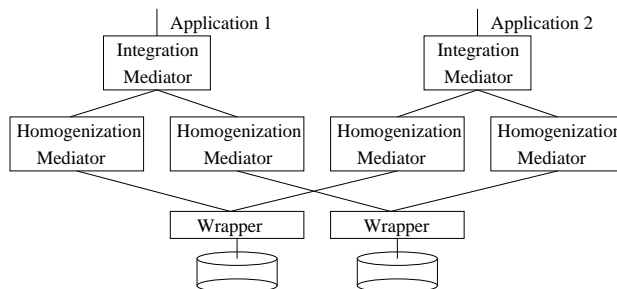


Figure 1. The AURORA Mediation Framework

Homogenization can be a complicated process when multiple types of mismatches exist simultaneously. A *homogenization methodology* ensures correct and complete homogenization. The MEA of AURORA-RH, MEA-RH, extends the relational algebra with operators specially designed for expressing *homogenizing views*. Queries posed against such views are mapped to one or more subqueries against the data source (via wrapper). This mapping should be optimized. In AURORA, such optimization is achieved via query modification in MEA-RH. We present a query rewriting algorithm and a set of transformation rules for query optimization in MEA-RH.

This paper is organized as follows. Section 2 introduces AURORA. Section 3 describes the structure of AURORA-RH. Section 4 describes MEA-RH. Section 5 describes MAT-RH. Section 6 describes query processing and optimization in AURORA-RH. Section 7 reviews related work. Section 8 contains conclusion and future research.

2. An Overview of AURORA

2.1. The AURORA Mediation Model

Assume that an application requires access to a set of data sources via an integrated view V . V is defined by application requirements, regardless of how the data are presented by the underlying data sources. In AURORA, each data source in the access scope of V must first be *homogenized* in regard to V and then *integrated* into V . Homogenization removes schematic discrepancies of individual data sources in regard to V ; these discrepancies arise when data sources model the same application domain differently. Once homogenized, a data source can be integrated into V through a simple “registration” mechanism. The exact definition of this mechanism is a research issue in AURORA. The goal is to make it (1) simple: it requires minimum expertise to manage; and (2) incremental: the sequence in which sources register is insignificant; and (3) scalable: its complexity is independent of the number of sources registered. AURORA mediation model is heavy in homogenization but light in integration; homogenization counts for most of the effort for including a new source. However, homogenization involves only one data source and can be managed with Mediator Author's Toolkits (MATs). Multiple sources can be homogenized in parallel.

2.2. Mediator Development Environment

AURORA provides a collection of workbenches, each consisting of a mediator skeleton and a Mediator Author's Toolkit (MAT).

AURORA Mediator Skeletons. A mediator consists of a mediator view and a query processor. Building a mediator means building both components. In AURORA, mediators are constructed from *mediator skeletons* that have the following built-in capabilities:

1. A mediation enabling algebra (MEA) for defining views and a repository to maintain them.
2. A query processor that entertains queries posed against views defined via the MEA.

Mediator skeletons are mediators with an empty view. Once a view is defined, a mediator skeleton becomes a custom-made mediator that is able to process queries posed against this view. Different types of mediators require skeletons that have different MEAs and query processors.

AURORA Mediator Author's Toolkits (MATs). In AURORA, a *mediator author* chooses a mediator skeleton, identifies heterogeneities among the sources, and defines views into the mediator skeleton to resolve the heterogeneities. As shown in Figure 2(a), AURORA provides interactive tools, the *Mediator Author's Toolkits* (MATs),

to assist the mediator authors in performing such tasks. A MAT has the following capabilities:

1. It mandates a *mediation methodology*; and
2. It provides mediation enabling operators (MEOs) for expressing the resolutions.

Construction of different types of mediators requires different methodologies and operators.

AURORA Mediator Development Environment. The AURORA mediator development environment consists of a collection of *workbenches*, environments where mediators of a particular type can be constructed by a mediator author. Figure 2(a) shows the general form of a workbench. Construction of different mediators requires different suites of MAT and skeleton. Rather than providing one workbench for all, AURORA provides a collection of workbenches classified along two dimensions: (1) the canonical data model and query language, relational or object-oriented, chosen according to application requirements. (2) the mediator type, homogenization or integration. Figure 2(b) shows this classification.

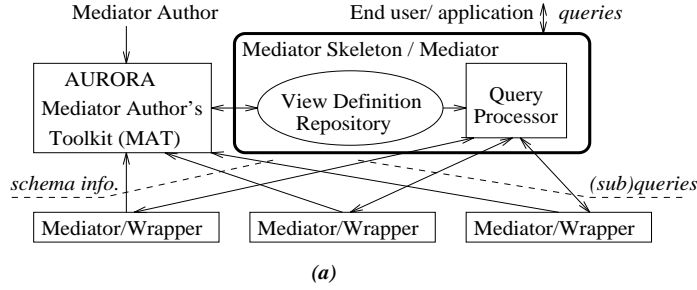
3. AURORA-RH

3.1. Homogenization and Query Processing

Let B be a relational database. Let H be a view consisting of relations M_1, \dots, M_n . To *homogenize* B into H is to specify procedures, $P_i(B)$ ($1 \leq i \leq n$), that construct relations M_i ($i = 1, n$) from the relations in B . B is the *source database*; relations in B are *source relations*; M_i ($i = 1, n$) are *target relations*. Queries posed against H are referred to as *mediator queries*. Assume procedures $P_i(B)$ ($1 \leq i \leq n$) have been specified and consider a mediator query Q . The task of processing Q is to 1) translate Q into queries over B ; and 2) send the queries to B and use the returned data to assemble the answer to Q .

Example Application. Figure 3 depicts a homogenization problem. Besides the differences in schema, we also assume: (1) In the source database, the sales and salary data is recorded in Canadian dollars, while in the target database, the same data is to be in US dollars; (2) In the target database, *Employee.salary* includes bonus as well as base salary; and (3) The target database perceives the domain of “jobs” differently from the source database. Rather than having job titles from {SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector}, the target database assumes the job titles are from {System Engineer, Development Engineer, Consultant, Research Scientist, Program Manager}.

Databases model *conceptual territories* by domains. Domains in different databases that model the same concept may differ, giving rise to *domain mismatches*. These domains can be converted to each other via *domain mappings*.



AURORA Workbenches

	Canonical Model	Relational	Object-Oriented
Mediator specialty			
Homogenization		AURORA-RH	AURORA-OH
Integration		AURORA-RI	AURORA-OI

Figure 2. AURORA Workbench: General Form and Classification

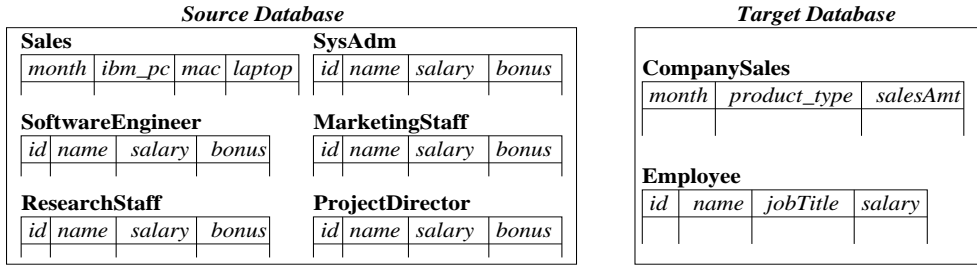


Figure 3. A Homogenization Problem

3.2. Architecture of AURORA-RH

Figure 4 shows the architecture. **MAT-RH** is a toolkit that assists a mediator author in constructing a homogenizing view, or a target database. It provides a MEA and mandates a homogenization methodology. Homogenization process is divided into 6 steps, each supported by a specialized tool (sections 5.3-5.8). Each tool accepts two types of information: (1) *transformations*, expressions in MEA-RH. (2) *domain mappings*, arbitrary functions. These are captured in the **View Definition Repository** and used for query processing. **AURORA-RH Primitives** are operators designed to facilitate homogenization; they extend the relational algebra to form MEA-RH. **AURORA-RH Query Processor (AQP)** translates a view query into a set of queries against the source and assembles the final answer using data retrieved from the source.

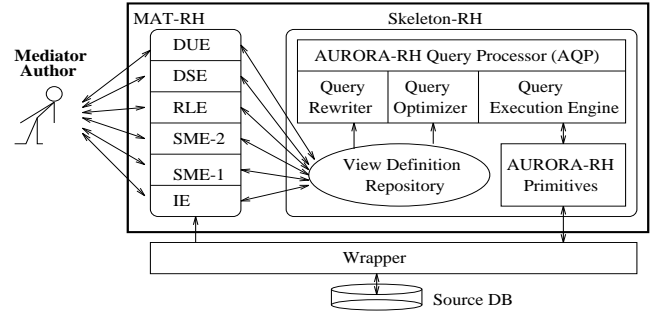


Figure 4. AURORA-RH Workbench

4. Primitives and Transformations

AURORA-RH primitives are MEOs designed to facilitate homogenization; they extend the relational algebra to form MEA-RH. All primitives take a relation as an argument and generate a relation; they compose with relational operators in a well-defined manner. For simplicity, two attributes are considered to be the “same” if they have the same name. We use $ATTR(R)$ to denote the set of attributes in relation R , $RELname(R)$ for the name of relation R , and $ATTRname(A)$ for the name of attribute

A. Let B be the source database to be homogenized. AURORA-RH provides the following primitives:

retrieve. Let Q be an algebraic expression over the source relations in database B ,

$$R' = retrieve(Q)$$

submits Q to database B and returns result in R' .

pad. Let R be a relation, A be an attribute, $A \notin ATTR(R)$, and c a constant,

$$R' = pad(R, A, c)$$

defines relation R' , $ATTR(R') = ATTR(R) \cup \{A\}$. The population of R' is defined by

$$R' = \{t' \mid t'[A] = c; t'[A'] = t[A'], t \in R, A' \in ATTR(R)\}$$

Let $R' = pad(retrieve(SysAdm), jobTitle, "SysAdm")$, where $SysAdm$ is given in Figure 3. R' has scheme $(id, name, salary, bonus, jobTitle)$ and includes all $SysAdm$ tuples tagged with “ $SysAdm$ ” as attribute $jobTitle$.

rename. Let R be a relation, $A \in ATTR(R)$, and n be an attribute name, $n \notin ATTR(R)$, then

$$R' = rename(R, A, n)$$

defines a relation R' with scheme identical to the scheme of R with attribute A renamed to n . The population of R' is defined by: $R' = \{t' | t'[n] = t[A], t'[A] = t[A], t \in R,$

$$A' \in ATTR(R) - \{A\}\}$$

deriveAttr. Let R be a relation. Let $L_i \subseteq ATTR(R)$ ($i = 1, k$) be a list of attributes in R . Let N_i ($i = 1, k$) be attributes. Let f_i be functions of appropriate signatures.

$$R' = deriveAttr(R, L_1, N_1, f_1, \dots, L_k, N_k, f_k)$$

defines a relation R' , $ATTR(R') = ATTR(R) \cup \{N_1, \dots, N_k\}$. The population of R' is defined by:

$$R' = \{t' | t'[N_i] = f_i(t[L_i]), 1 \leq i \leq k; t'[A] = t[A], A \in ATTR(R) - \{N_1, \dots, N_k\}, t \in R\}$$

For each tuple $t \in R$, *deriveAttr* generates $t' \in R'$ by adding fields N_i to t ($i=1,k$) and sets its value to be $f_i(t[L_i])$. *deriveAttr* is used in map values with arbitrary functions, as in sections 5.7 and 5.8.

A **transformation expression**, T_E , is a well-formed expression in MEA-RH. T_E defines a relation, $R = T_E$. If T_E is *retrieve(Q)*, R is a *direct relation*, otherwise, R is a *derived relation*. A direct relation is the result of a query over the source.

5. Mediator Author's Toolkit: MAT-RH

5.1. Domain and Schema Mismatches

Let B be a source database and M be a target relation. MAT-RH identifies these types of mismatches:

Cross-over schema mismatches. A **type 1** cross-over mismatch happens when a concept is represented as data in M but as relations in B . A **type 2** cross-over mismatch happens when a concept is represented as data in M but as attributes in B .

Domain structural mismatches. This mismatch happens when a domain in M corresponds to a domain with a different data type or several data domains in B .

Domain unit mismatches. This mismatch happens when a domain in M assumes different unit of measurement from the corresponding domain(s) in B .

Domain population mismatches. This mismatch happens when a domain in M assumes different population from the corresponding domain(s) in B .

Example-1. The example shown in Figure 3 demonstrates all of the above mentioned mismatches:

- (cross-over schema mismatch, type 1.) In the target database, *jobs* is represented as data domain "*jobTitle*" in relation *Employee*, but is represented as relations in the source database.

- (cross-over schema mismatch, type 2.) In the target database, *product types* is represented as data domain *product_type*, but is represented as attributes in the source database.

- (domain structural mismatch.) In the target database, *salary* means the total income. The same concept is represented by two data domains, *salary* and *bonus*, in the source database.

- (domain unit mismatch.) In the target database, all money amounts use US dollar as unit, while in the source database, all money amounts reported are in Canadian dollars.

- (domain population mismatch.) The two database schemas assume different domain populations of the concept *jobs*.

5.2. Homogenization Methodology

The MAT-RH mandates a 6-step methodology for homogenization:

1. Construct an import schema;
2. Resolve type 1 schema mismatches;
3. Resolve type 2 schema mismatches;
4. Link relations;
5. Resolve domain structural mismatches; and,
6. Resolve domain unit/population mismatches.

This methodology is demonstrated by examples in the rest of this section. In each step, derived relations and/or domain mappings are specified. MAT-RH supports each step by a specialized tool (environment) that accepts certain transformations and domain mappings. Some environments provide special transformations for resolving specific types of mismatches. In the following sections, each environment is described along 4 dimensions:

- 1) Input relations.
- 2) Transformations allowed.
- 3) Output semantic information: domain mappings and other semantic information allowed.
- 4) Output relations.

5.3. Import Environment (IE)

The input to IE (Figure 6) includes all the source relations exported by source B . Output of IE is a set of direct relations of the form $R = retrieve(Q)$, where Q is a relational algebraic expression over database B . No semantic mapping information is produced.

Example-2: Importing source database. The IE allows the mediator authors to choose relations and data of interest from the source database. In our example, all relations are of interest. The importing step produces a set of direct relations $R = retrieve(R)$, where $R \in \{Sales, SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector\}$.

5.4. SME-1 for Type 1 Cross-over Mismatches

Figure 6 depicts Schema Mismatch Environment 1 (SME-1).

Input relations. All relations produced by IE.

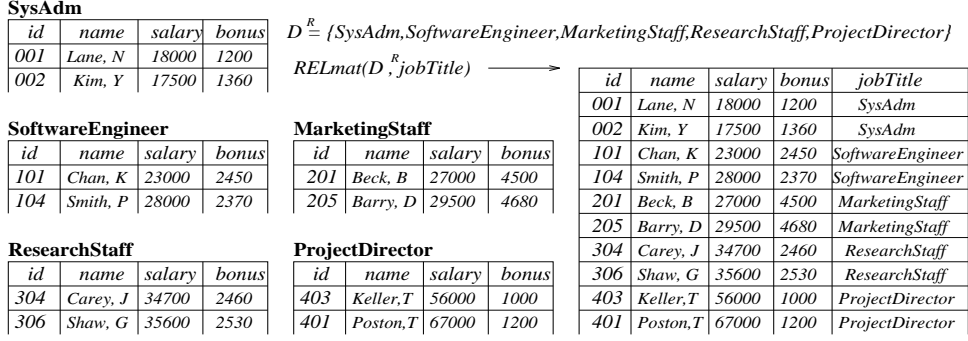


Figure 5. The $RELmat$ operator

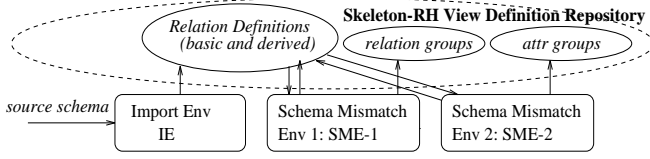


Figure 6. IE, SME-1 and SME-2

Transformation operators. $RELmat$, π , σ , \bowtie , $rename$, pad , $deriveAttr$.

$RELmat$ (relation materialize) is a special transformation operator. Given $D^R = \{R_1, \dots, R_n\}$, a group of relations with identical schemes, let A be an attribute, $A \notin ATTR(R_1)$, then

$$RELmat(D^R, A) = \bigcup_{i=1}^n pad(R_i, A, RELname(R_i))$$

The result relation contains tuples from all the relations in D^R , each tagged with a new field A that contains the name of the relation it came from, as illustrated in Figure 5. The application of $RELmat$ is illustrated in Example-3.

Output semantic information. SME-1 allows specification of relation groups (D^R). In the target database, the names of the relations in a relation group form an enumerated data domain which represents a concept that is represented as relations in the source database.

Output relations. Derived relations.

Example-3: solving type 1 cross-over schema mismatch. The source database models $jobs$ as relations. The target database models $jobs$ as a data domain $Employee.jobTitle$. The following is the resolution to this mismatch:

$$D^R = \{SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector\}$$

$$S_{Employee} = RELmat(D^R, jobTitle)$$

Figure 5 further illustrates this resolution.

5.5. SME-2 for Type 2 Cross-over Mismatches

Schema Mismatch Environment 2 (SME-2) is depicted in Figure 6.

Input relations. All relations defined in previous steps.

Transformation operators. $ATTRmat$, π , σ , \bowtie , $rename$, pad , $deriveAttr$.

$ATTRmat$ (attribute materialize) is a special transformation operator. Given $D^A = \{A_1, \dots, A_n\}$, a group of attributes in a relation S that have identical data types, let N_A and N_V be attributes, $N_A, N_V \notin ATTR(S)$, then:

$$ATTRmat(S, D^A, N_A, N_V) = \bigcup_{i=1}^n pad(rename(\pi_{ATTR(S) - D^A \cup \{A_i\}}(S), A_i, ATTRname(N_V)), N_A, ATTRname(A_i))$$

The effect of this operator is illustrated in Figure 7. Application of $ATTRmat$ is demonstrated in Example-4.

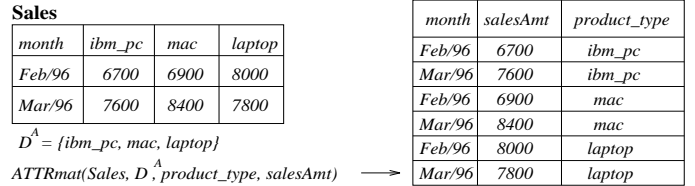


Figure 7. The $ATTRmat$ operator

Output semantic information. SME-2 allows specification of attribute groups (D^A). In the target database, the names of the attributes in an attribute group form an enumerated data domain which represents a concept that is represented as attributes in the source database.

Output relations. Derived relations.

Example-4: Solving type 2 schema mismatch. The source database models $product_types$ as attributes ibm_pc , mac , $laptop$ in $Sales$, while the target database models it as a domain $CompanySales.product_type$. The following is the resolution:

$$D^A = \{ibm_pc, mac, laptop\}, S_{CompanySales} = ATTRmat(Sales, D^A, product_type, salesAmt)$$

This resolution is further illustrated in Figure 7.

5.6. RLE: Environment for Relation Linking

Relation Linking Environment, RLE, is depicted in Figure 8. The input includes relations previously defined. Derived relations can be defined using $rename$, π , σ , and

∅. RLE mandates the derivation of a distinguished relation S_M , a “prototype” of M modulo data domain mismatches. S_M is the only relation referenced in future steps.

Example-5: relation linking. In Example-3 and 4, relations $S_{Employee}$ and $S_{CompanySales}$ are defined. Nothing is to be done in RLE in the example application.

5.7. DSE: Solving Domain Structural Mismatches

The Domain Structure Environment, DSE (Figure 8), supports resolution of structural mismatches (section 5.1).

Input relations. The distinguished relation, S_M .

Transformation operators. None.

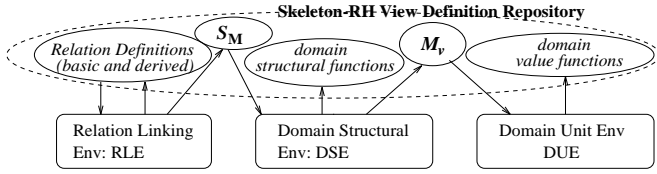


Figure 8. RLE, DSE and DUE

Output semantic information. DSE captures domain structural functions. Consider $A \in ATTR(M)$. The corresponding domain(s) of A in S_M might be an attribute of the same data type, or an attribute of a different data type, or several attributes. Let $ATTR(M) = \{A_1, \dots, A_m\}$. To derive each of these attributes from attributes of S_M , the DSE requires the following to be specified for each A_i ($i = 1, m$):

1. $L_i = \{A_{i,1}^S, \dots, A_{i,k}^S\}$, attributes in S_M that correspond to A_i . By default $L_i = \{S_M.A_i\}$. If $A_i \notin ATTR(S_M)$, L_i must be given explicitly.

2. *domain structural function (DSF)*, f_i^s , that maps L_i to A_i . f_i^s is an identity function by default.

Inverses of DSFs, if they exist, must be given.

Output relations. No relation is explicitly derived. However, by defining DSFs for all attributes in M , the following relation is implicitly defined:

$$M_v = \pi_{A_1, \dots, A_m}(\text{deriveAttr}(S_M, L_1, A_1, f_1^s, \dots, L_m, A_m, f_m^s))$$

M_v is identical to M modulo domain value mismatches.

Example-6: solving domain structural mismatch. In relation $S_{Employee}$ defined in Example-3, there is (base)*salary* and *bonus*. In target relation $Employee$, we expect *salary* to include the total income of an employee. The following is a resolution to this mismatch:

$$L_{Employee.salary} = \{salary, bonus\},$$

$$f_{Employee.salary}^s(s, b) = s + b$$

5.8. DUE: Solving Domain Value Mismatches

The Domain Unit Environment, DUE (Figure 8), supports resolution of domain unit/population mismatches(section 5.1).

Input relations. Relation M_v constructed by DSE.

Transformation operators. None.

Output semantic information. The DUE captures domain value functions. For an attribute $A \in ATTR(M)$, the values of $M_v.A$ may differ from that of $M.A$ due to (1) difference in unit of measure; and/or (2) difference in domain population. DUE requires that for each attribute $A \in ATTR(M)$, a *domain value mapping* be specified to convert values in domain $M_v.A$ to that in $M.A$. This mapping is by default an identity function but can be an arbitrary function or a stored mapping table. In this paper, we only consider *domain value function (DVF)*, that maps each $M_v.A$ value to a unique $M.A$ value. Inverses of DVFs, if they exist, must also be specified.

Output relations. No relation is explicitly derived. However, by specifying DVFs for each attribute in M , the following relation is implicitly derived:

$$M = \text{deriveAttr}(M_v, A_1, A_1, f_1^v, \dots, A_k, A_k, f_m^v)$$

where $ATTR(M) = \{A_1, \dots, A_m\}$ and f_i^v ($i = 1, m$) is the DVF for attribute A_i .

Example-7: Solving domain unit mismatch. $Employee_v$ derived in Example-6 has attribute *salary*, but its values are in Canadian dollars. In the target database, we expect to see US dollars only. Assume 1 Canadian dollar worths r US dollars, we define DVF for $Employee.salary$ as:

$$f_{Employee.salary}^v(s) = \text{CNDtoUSD}(s) = s \times r$$

Similarly, we define DVF for $CompanySales.salesAmt$:

$$f_{CompanySales.salesAmt}^v(s) = \text{CNDtoUSD}(s) = s \times r$$

$\text{CNDtoUSD}()$ has an inverse, USDtoCND .

Example-8: Solving domain population mismatch.

$Employee_v$ derived in Example-6 has attribute *jobTitle* but its values are from $\{\text{SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector}\}$. $Employee.jobTitle$ consists of $\{\text{System Engineer, Development Engineer, Consultant, Research Scientist, Program Manager}\}$. To resolve this mismatch, a DVF must be specified for $Employee.jobTitle$:

$$f_{Employee.jobTitle}^v(j) = \text{jobMap}(j)$$

jobMap is a mapping table given in Table 1, it has an inverse.

source database	target database
SysAdm	System Engineer
SoftwareEngineer	Development Engineer
MarketingStaff	Consultant
ResearchStaff	Research Scientist
ProjectDirector	Program Manager

Table 1. jobMap for $Employee.jobTitle$

6. AURORA-RH Query Processor (AQP)

As shown in Figure 4, AQP consists of a query execution engine, a query rewriter and a query optimizer. Query Exe-

cution Plans(QEPs) are expressions that involve only source relations; it can be depicted as an operation tree whose nodes are annotated with an operator name and an argument list. A non-leaf node of the tree is either an AURORA-RH primitive, *rename*, *pad* or *deriveAttr*, or a relational operator. The leaf nodes of the tree are *retrieve* primitives. Figures 9, 10, and 11 are QEP trees. The AQP query execution engine evaluates QEP trees bottom up.

6.1. AQP Query Rewriter

We consider mediator queries in the form of $\pi_L \sigma_p(M)$, where L is a list of attributes in M and p is a predicate. The rewriting algorithm given below can be adapted for join queries. Via MAT-RH, the derivation of M is captured as transformations and domain mappings in the View Definition Repository. These are used to rewrite Q into a QEP.

Algorithm. AQPrewriteQuery

Input: $Q = \pi_L \sigma_p(M)$ **Output:** A QEP for Q

1. Replace M in Q with S_M . Replace *RELmat* and *ATTRmat* with its definition.
 - while** (Q involves a *direct* or *derived* relation R)
 - Replace R with its derivation.
 - Replace *RELmat* and *ATTRmat* with its definition.
2. Let $Q = \pi_L(Q')$, let $\{A_1, \dots, A_m\}$ be all the attributes in L whose domain value functions, f_1^v, \dots, f_m^v , are *not* identity functions, rewrite Q as:

$$Q = \text{deriveAttr}(\pi_L(Q'), A_1, A_1, f_1^v, \dots, A_m, A_m, f_m^v)$$
3. For each attribute A involved in p , if its DVF, f^v , is not identity, replace it with $f^v(A)$.
4. Let $L = \{A_1, \dots, A_n\}$. Let f_1^s, \dots, f_n^s be the DSFs of A_1, \dots, A_n . Let $L_i (i = 1, n)$ be the list of attributes which are the arguments of f_i^s . Then do the following:
 - (a) Replace Q' by

$$\text{deriveAttr}(Q', L_1, A_1, f_1^s, \dots, L_n, A_n, f_n^s)$$
 - (b) For $i = 1$ to n do:
 - If f_i^s is an identity function, first, remove L_i, A_i, f_i^s from the argument list of *deriveAttr* function constructed above; second, if the argument attribute in L_i, A_i^l , has a different name from A_i , replace the first argument of *deriveAttr*, E' with *rename*($E', A_i^l, \text{ATTRname}(A_i)$).
5. For each attribute A involved in p , if its DSF, f^s , is not identity, replace A with $f^s(A'_1, \dots, A'_k)$, where A'_1, \dots, A'_k is the argument list of f^s .
6. Repeat until no modification can be made:

For each subexpression in p that is in the form of $f(E_1) \theta f(E_2)$ or $f(E_3) \theta c$, where E_1, E_2 and E_3 are expressions, c is a constant, $\theta \in \{=, >, <\}$, and f is a function which has an inverse f^{-1} , if f is strictly monotonic or θ is “=”, replace this subexpression with $E_1 \theta E_2$ or $E_3 \theta c'$, respectively, where $c' = f^{-1}(c)$. ■

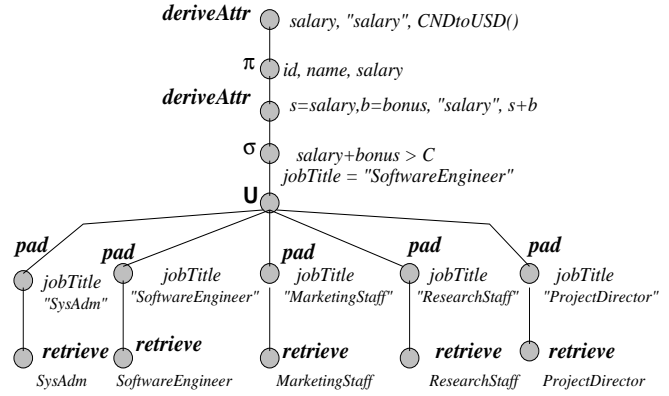


Figure 9. An QEP for Example-9.

Example-9. Consider query:

$Q = \pi_{id, name, salary} \sigma_{salary > 50000} \sigma_{jobTitle = \text{“DevelopmentEngineer”}} (Employee)$ that retrieves the *id*, *name* and *salary* of development engineers who earn more than 50000. Rewrite Q using the above algorithm:

step 1. From the definitions of $S_{Employee}$ and *RELmat* in Example-3 and section 5.4, we get:

$$Q = \pi_{id, name, salary} \sigma_{salary > 50000} \sigma_{jobTitle = \text{“DevelopmentEngineer”}} (\text{pad}(\text{retrieve}(\text{SysAdm}), \text{“jobTitle”}, \text{“SysAdm”}) \cup \text{pad}(\text{retrieve}(\text{SoftwareEngineer}), \text{“jobTitle”}, \text{“SoftwareEngineer”}) \cup \text{pad}(\text{retrieve}(\text{MarketingStaff}), \text{“jobTitle”}, \text{“MarketingStaff”}) \cup \text{pad}(\text{retrieve}(\text{ResearchStaff}), \text{“jobTitle”}, \text{“ResearchStaff”}) \cup \text{pad}(\text{retrieve}(\text{ProjectDirector}), \text{“jobTitle”}, \text{“ProjectDirector”}))$$

steps 2 and 3. DVFs are defined for *salary* and *jobTitle*. The *salary* is in projection list. Performing steps 2 and 3 we get:

$$Q = \text{deriveAttr}(\pi_{id, name, salary} \sigma_{CNDtoUSD(salary) > 50000} \sigma_{jobMap(jobTitle) = \text{“DevelopmentEngineer”}} (\bigcup(\text{pad}(\dots))), salary, \text{“salary”}, CNDtoUSD())$$

steps 4 and 5. *salary* has a non-trivial DSF. It is in the projection list and the predicate. Performing steps 4 and 5, we get:

$$Q = \text{deriveAttr}(\pi_{id, name, salary} (\text{deriveAttr}(\sigma_{CNDtoUSD(salary+bonus) > 50000} \sigma_{jobMap(jobTitle) = \text{“DevelopmentEngineer”}} (\bigcup(\text{pad}(\dots))), \{salary, bonus\}, \text{“salary”}, f_{Employee.salary}^s)), salary, \text{“salary”}, CNDtoUSD())$$

step 6. *CNDtoUSD* has an inverse *USDtoCND*, so does *jobMap*(Table 1). Let $C = USDtoCND(50000)$. Performing step 6, we get the final QEP, also shown in Figure 9:

$$Q = \text{deriveAttr}(\pi_{id, name, salary} (\text{deriveAttr}(\sigma_{salary+bonus > C} \sigma_{jobTitle = \text{“SoftwareEngineer”}} (\bigcup(\text{pad}(\dots))), \{salary, bonus\}, \text{“salary”}, f_{Employee.salary}^s)), salary, \text{“salary”}, CNDtoUSD())$$

6.2. AQP Query Optimization

The AQP query optimizer maximizes the number of relational operations performed by the source database so as to leverage the query optimization capability of the source and reduce the amount of data fetched. A QEP generated by the rewriter is transformed to *enlarge* the (sub)queries submitted to the source database. As *retrieve* is the only operator that submits queries, the optimizer pushes as many as possible relational operators into *retrieve*. Consider a QEP tree such as Figure 9. the query optimizer 1) pushes relational operators “across” *pad*, *rename*, and *deriveAttr* so that they move towards the leaves; and 2) pushes relational operators across *retrieve*, so that they become part of the argument (annotation) of the *retrieve* leaf. In this section, we discuss transformation rules and control strategies in AQP query optimizer.

Table 2 gives transformation rules for exchanging relational operators with *pad*, *rename* and *deriveAttr*. For simplicity, the rules for *deriveAttr* are given only for cases where there is one derived attribute. Proof of rules for *deriveAttr* is given in [18]. $p^{N \leftarrow X}$ denotes the predicate obtained from p by substituting all appearances of N with X . If p does not involve N , $p^{N \leftarrow X} = p$. $L_{N \leftarrow A}$ denotes the list of attributes obtained from L by replacing attribute N with A . If L does not involve N , $L_{N \leftarrow A} = L$.

A relational operator can be pushed into *retrieve* if it is acceptable to the source query facility. As most relational query languages do not allow user-defined functions, selections whose predicates involve functions that are not built-in in the source query facility do not exchange with *retrieve*. This potentially increases the amount of data fetched from the source. In Algorithm **AQPrewriteQuery** step 6, inverses of domain mapping functions are used to eliminate such selection predicates.

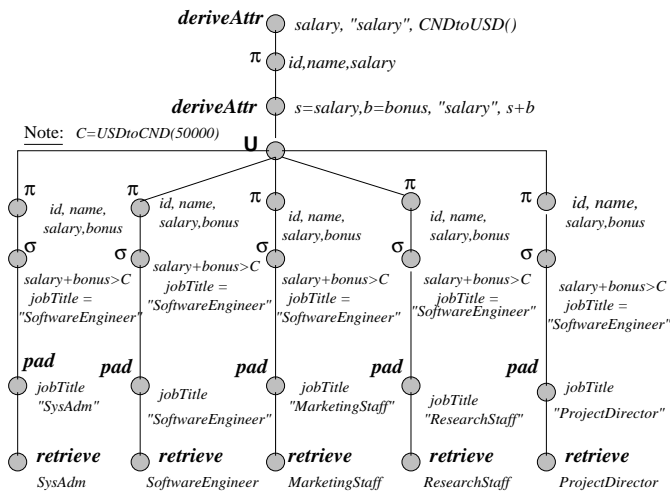


Figure 10. Transformed QEP

Example-10. Use rule $T_{deriveAttr}[2]$ to exchange π with the *deriveAttr* under it, the π argument list is now $id, name, salary, bonus$. Exchange \cup with this π and σ , we get Figure 10. Push σ across *pad* operators using rule $T_{pad}[3]$, many of the *pad* subtrees become ϕ , e.g.

$$\begin{aligned} & \sigma_{jobTitle="SoftwareEngineer"}(pad(retrieve(SysAdm), \\ & \quad jobTitle, "SysAdm")) \\ &= \sigma_{"SysAdm"="SoftwareEngineer"}(pad(retrieve(SysAdm), \\ & \quad jobTitle, "SysAdm")) = \phi \end{aligned}$$

Trim the empty branches from Figure 10 to get Figure 11 (a). Use rule $T_{pad}[1]$ to push π across *pad* to obtain Figure 11 (b). Finally, push the relational operators across *retrieve*. Since the selection predicate involves a function $+$, known to the source database, both π and σ exchange with *retrieve* to form Figure 11 (c).

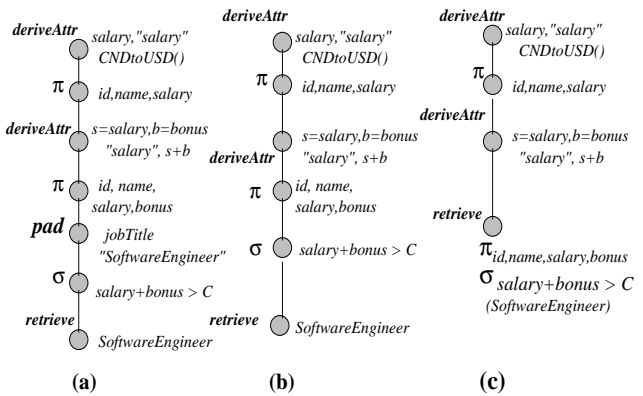


Figure 11. Transformed QEPs

7. Related Work

There are many mediation frameworks in the literature, Multidatabase [2], Superviews [13], TSIMMIS [14], Garlic [1], HERMES [16], DIOM [9], to cite a few. The AURORA project is different from all these frameworks due to its novel plug-and-play mediation model. Usually, an integrated view is constructed using a *mediation language*, such as the Mediator Specification Language of TSIMMIS [14], or the definition language of UniSQL/M [6], that allows specification of the derivation of this view from data provided by the underlying sources. Adding or removing a data source requires modifying this specification. When a large number of data sources are involved, this specification grows large and complex, difficult to create or modify. AURORA's mediation model intends to remove this problem. AURORA integration mediators bear some similarity to the Information Manifold (IM) project [8] in that both systems assume the existence of a pre-defined application view (called the “worldview” in IM). However, AURORA differs from IM by including a homogenization step. Technically, IM's query processing is logic-based while that

Transformation rules for <i>pad</i>	
$T_{pad}[1].$	$\pi_L(pad(R, N, s)) \equiv \pi_L(R), L \subseteq ATTR(R), N \notin ATTR(R).$
$T_{pad}[2].$	$\pi_L(pad(R, N, s)) \equiv pad(\pi_{L-\{N\}}(R), N, s), L \subseteq \{N\} \cup ATTR(R), N \in L.$
$T_{pad}[3].$	$\sigma_p(pad(R, N, s)) \equiv pad(\sigma_{pN \leftarrow s}(R), N, s).$
$T_{pad}[4].$	$R \bowtie_p pad(R_1, N_1, s_1) \equiv pad(R \bowtie_{pN_1 \leftarrow s_1} R_1, N_1, s_1).$
$T_{pad}[5].$	$pad(R_1, N_1, s_1) \bowtie_p pad(R_2, N_2, s_2) \equiv pad(pad(R_1 \bowtie_{pN_1 \leftarrow s_1, N_2 \leftarrow s_2} R_2, N_1, s_1), N_2, s_2).$
Transformation rules for <i>rename</i>	
$T_{rename}[1].$	$\pi_L(rename(R, A, N)) \equiv \pi_L(R), L \subseteq ATTR(R), N \notin ATTR(R).$
$T_{rename}[2].$	$\pi_L(rename(R, A, N)) \equiv rename(\pi_{LN \leftarrow A}(R), A, N),$ $L \subseteq \{N\} \cup ATTR(R) - \{A\}.$
$T_{rename}[3].$	$\sigma_p(rename(R, A, N)) \equiv rename(\sigma_{pN \leftarrow A}(R), A, N).$
$T_{rename}[4].$	$R \bowtie_p rename(R_1, A_1, N_1) \equiv rename(R \bowtie_{pN_1 \leftarrow A_1} R_1, A_1, N_1).$
$T_{rename}[5].$	$rename(R_1, A_1, N_1) \bowtie_p rename(R_2, A_2, N_2)$ $\equiv rename(rename(R_1 \bowtie_{pN_1 \leftarrow A_1, N_2 \leftarrow A_2} R_2, A_1, N_1), A_2, N_2).$
Transformation rules for <i>deriveAttr</i>	
$T_{deriveAttr}[1].$	$\pi_L(deriveAttr(R, L_1, N, f)) \equiv \pi_L(R), L \subseteq ATTR(R), N \notin ATTR(R).$
$T_{deriveAttr}[2].$	$\pi_L(deriveAttr(R, L_1, N, f)) \equiv \pi_L(deriveAttr(\pi_{L-\{N\} \cup L_1}(R), L_1, N, f)),$ $L \subseteq \{N\} \cup ATTR(R), N \in L.$
$T_{deriveAttr}[3].$	$\sigma_p(deriveAttr(R, L, N, f)) \equiv deriveAttr(\sigma_{pN \leftarrow f(L)}(R), L, N, f).$
$T_{deriveAttr}[4].$	$R \bowtie_p deriveAttr(R_1, L_1, N_1, f_1) \equiv deriveAttr(R \bowtie_{pN_1 \leftarrow f_1(L_1)} R_1, L_1, N_1, f_1),$ $ATTR(R) \cap ATTR(R_1) = \phi, N_1 \notin ATTR(R).$
$T_{deriveAttr}[5].$	$deriveAttr(R_1, L_1, N_1, f_1) \bowtie_p deriveAttr(R_2, L_2, N_2, f_2)$ $\equiv deriveAttr(deriveAttr(R_1 \bowtie_{pN_1 \leftarrow f_1(L_1), N_2 \leftarrow f_2(L_2)} R_2, L_1, N_1, f_1), L_2, N_2, f_2),$ $ATTR(R_2) = \phi, N_1 \notin ATTR(R_2), N_2 \notin ATTR(R_1), N_1 \neq N_2, N_2 \notin L_2.$

Table 2. Transformation Rules for *pad*, *rename* and *deriveAttr*

in AURORA is based on MEAs which provide operators specially designed for handling data from heterogeneous sources.

[15] and [4] present intelligent mediation techniques that detect and resolve semantic heterogeneities automatically by reasoning about semantics in a knowledge base or ontology. In AURORA, such tasks are performed by a mediator author using MATs. Once established, intelligent mediation techniques can replace mediator authors. The AURORA approach is a practical alternative. AURORA investigates a host of issues in mediator view expression and query processing that are essential even when heterogeneities are detected and resolved automatically.

[5] identifies domain mappings for resolving domain and schema mismatches. Resolutions for individual mismatches are demonstrated using an object-oriented database programming language. [5] does not provide a mediation methodology, nor does it explore query optimization techniques in presence of the new language constructs. [6] provides a comprehensive classification of mismatches and conflicts. Resolutions for individual conflicts are given. New language constructs are proposed but query rewriting and optimization methods for these constructs are not given. [4] uses ontology to detect and resolve mismatches due to different units of measure. It is not clear how [4] handles other types of schematic mismatches.

Disco [17] extends ODMG ODL for mediation and in-

tends to use Volcano for query optimization. It introduces a logical operator *submit* and gives rules for exchanging relational operators with it. The cost model used is unclear. [3, 10] describe approaches that collect/establish statistics to build mediator query cost models. AURORA-RH concentrates on single-source query modification techniques to leverage the source query optimization capability; a mediator query cost model is not necessary. However, mediator query cost model is an interesting research topic.

8. Conclusion and Future Work

We have described AURORA, a project that develops techniques for building efficient and scalable mediation. Our contributions are as follows. First, we have proposed a novel plug-and-play mediation model (Figure 1) which is a divide-and-conquer approach to building integrated access to heterogeneous sources. This model enables us to reduce the general mediator query processing issue into two smaller problems: that in homogenization mediators and that in integration mediators. AURORA develops specialized mediation enabling algebras (MEAs) for each sub-problem. Second, we have described a complete suite of techniques used by a specific AURORA mediator, AURORA-RH, including (1) a homogenization methodology supported by MAT-RH; (2) a MEA, MEA-RH; and (3) query modification and optimization algorithms based on

MEA-RH.

Currently we are implementing AURORA-RH. The major implementation issue is the design of an interactive user interface of MAT-RH. To use MAT-RH, the mediator author needs to access knowledge about the source and target schemas, the mismatches, and the resolutions. A friendly user interface should clearly present this information and guide the mediator author through a correct and complete homogenization. In particular, this interface should help the user to: (1) follow the homogenization methodology (Section 5.2); (2) correctly use the AURORA-RH transformations, such as *RELMat* and *ATTRmat*; (3) define domain structural/value functions with appropriate signatures and valid implementations; and (4) browse the source and target schema, and current transformations and mappings captured in the view definition repository.

Our ultimate goal is to build a collection of mediators, AURORA-RI, AURORA-OH, and AURORA-OI (Figure 2 (b)). These mediators will be of similar forms as AURORA-RH but require different mediation methodologies and MEAs. Different query rewriting algorithm and transformation rules must be developed in similar fashion as in AURORA-RH. We are also designing an infrastructure where all AURORA mediators communicate and cooperate with one another via ORBs.

References

- [1] M. J. Carey et al. Towards Heterogeneous Multimedia Information Systems: the Garlic Approach. In *Fifth Int. Workshop on Research Issues in Data Engineering – Distributed Object Management (RIDE-DOM'95)*, pages 124–131, Tai Pei, Taiwan, Mar. 1995.
- [2] U. Dayal and H.-Y. Hwang. View definition and generalization for database integration in a multidatabase system. *IEEE Trans. on Software Engineering*, SE-10(6):628–645, Nov. 1984.
- [3] W. Du, R. Krishnamurthy, and M. Shan. Query Optimization in a Heterogeneous DBMS. In *Proc. 18th Int'l Conf. on Very Large Data Bases*, pages 277–291, 1992.
- [4] C. H. Goh, M. E. Madnick, and M. D. Siegel. Ontologies, Context, and Mediation: Representing and Reasoning about Semantic Conflicts in Heterogeneous and Autonomous Systems. Working Paper 3848, MIT Sloan School of Management, 1995.
- [5] W. Kent. Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language. In *Proc. 17th Int'l Conf. on Very Large Data bases*, pages 147–160, 1991.
- [6] W. Kim et al. On Resolving Schematic Heterogeneity in Multidatabase Systems. *Distributed and Parallel Databases*, 1(3):251–279, 1993.
- [7] R. Krishnamurthy, W. Litwin, and W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, pages 40–49, 1991.
- [8] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB 96*, Bombay, India, Sept. 1996.
- [9] L. Liu, C. Pu, and Y. Lee. An Adaptive Approach to Query Mediation Across Heterogeneous Information Sources. In *Int. Conf. on Cooperative Information Systems (CoopIS)*, pages 144–156, June 1996.
- [10] H. J. Lu, B. C. Ooi, and C. H. Goh. On Global Multidatabase Query Optimization. *ACM SIGMOD Record*, 21(4):6–11, Dec. 1992.
- [11] W. Meng et al. Construction of Relational Front-end for Object-Oriented Database Systems. In *Proc. 9th Int'l. Conf. on Data Engineering*, pages 476–483, 1993.
- [12] P. Missier and M. Rusinkiewicz. Extending a multidatabase manipulation language to resolve schema and data conflicts. In *IFIP TC-2 Working Conference on Data Semantics (DS-6)*, Stone Mountain, Georgia, May 1995.
- [13] A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Trans. on Software Engineering*, SE-13(7):785–798, July 1987.
- [14] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. In *Proc. 12th Int'l. Conf. on Data Engineering*, 1996.
- [15] X. Qian and T. F. Lunt. Semantic Interoperation: A Query Mediation Approach. Technical Report SRI-CSL-94-02, Computer Science Laboratory, SRI International, Apr. 1994.
- [16] V. S. Subrahmanian et al. HERMES: Heterogeneous Reasoning and Mediator System. Unpublished document, University of Maryland.
- [17] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Heterogeneous Databases and the Design of Disco. In *Proceedings of the International Conference on Distributed Computer Systems*, 1996.
- [18] L. L. Yan, T. Ozsu, and L. Liu. Towards a Mediator Development Environment: The AURORA Approach. Technical Report TR-96-21, Department of Computing Science, University of Alberta, Aug. 1996.
- [19] C. Yu et al. Translation of Object-Oriented Queries to Relational Queries. In *Proc. 11th Int'l. Conf. on Data Engineering*, pages 90–97, 1995.