International Journal of Cooperative Information Systems,

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, CANADA

ABSTRACT

We develop a 2-tier, plug-and-play mediation model for accessing a large number of heterogeneous data sources. This model defines a divide-and-conquer approach towards information integration. It is more suitable than existing models for applications such as electronic commerce. We also develop algebras that manipulate heterogeneous data, the *mediation enabling algebras*, that provide new techniques for efficient query processing in large-scale middleware. This paper presents the mediation model, architecture and techniques studied in the AURORA project.

Keywords: Mediator architecture, Mediation enabling algebras, Scalability

1. Introduction

Today, a vast amount of digital information is provided by sources such as database systems, WWW pages, file systems and spreadsheets. With the advent of the Internet, the way people use information is changing rapidly. Software that facilitates access to multiple heterogeneous information sources is needed. Such software is commonly known as *middleware*. Building middleware is a complicated process due to the heterogeneities among the sources at levels ranging from platform to semantics [12]; this complexity grows rapidly as the number of sources increases. Middleware provides one-stop data access by managing the complexity on the application's behalf; it performs *mediation* between applications and diverse data sources. The goal of the AURORA project is to develop a collection of mediators that can be used for constructing middleware which provides integrated (read-only) access to heterogeneous and autonomous data sources. This middleware must be:

1. *Scalable.* Adding or removing a data source from the access scope is easy.

2. *Efficient.* When processing a query, redundant data retrieval is minimized.

Research in AURORA has two main themes: a 2-tier mediation model and techniques for efficient query processing.

1.1. 2-tier Mediation Model

AURORA models mediation as a 2-step process: homogenization followed by integration, each performed by respective mediators. Homogenization removes idiosyncrasies of a source in structure and semantics; multiple sources can be homogenized independently and in parallel. After being homogenized, a data source can be "plugged" into an integration mediator and removed by "unplugging" it. Since all the sources are homogenized, the procedures of plugging and unplugging handle a small range of heterogeneities and require minimum expertise. Homogenization is more difficult and is assisted by AURORA tools.

AURORA's mediation model defines a divide-and-conquer approach towards information integration. It facilitates scalable data management in large scale applications such as electronic commerce. It also enables us to decompose the highly complex technical issues in large-scale middleware, such as query optimization, into more manageable problems. Although many previous mediation models exist, none provides the above facilities. Many technical issues in large-scale middleware are known to be difficult (ref. Sections 4.1.3, 4.1.4).

1.2. Efficient Query Processing

Much is known about identifying and resolving semantic heterogeneities [12,23], the impact of this process on query processing efficiency is seldom discussed. We intend to address this missing link. AURORA schema integration constructs are tightly tied with algebras suitable for query optimization. These algebras, called *Mediation Enabling Algebras* (MEAs), provide operations specifically designed for manipulating heterogeneous and autonomous data. With MEAs, we identify the impact of mediation and take it into account in query processing.

Mediator views as well as mediator queries are expressed in MEAs. The view expressions are used to modify a mediator view query into an expression that is then manipulated by an algebraic query optimizer in order to achieve optimal query performance. The following techniques are studied:

1. MEAs for AURORA mediators.

2. Algebraic query rewriting and optimization in each MEA.

3. Selective materialization of data to enable intelligent query decomposition.

The rest of this paper is organized as follows. Section 2 describes a motivating application for AURORA. Section 3 reviews existing approaches and techniques. Section 4 describes the AURORA approach. Section 5 describes the technique suite employed by AURORA-RH, the relational homogenization mediator. Section 6 describes AURORA integration mediators. Section 7 contains conclusions and future work.

2. Electronic Commerce: a Motivating Application

A virtual shopping mall is a typical electronic commerce (EC) application. A key component in this application is the catalog system. Companies organize their catalogs differently. This gives rise to a set of heterogeneous and autonomous catalogs. When the number of participating catalogs is large, it is difficult for a shopper to locate items of interest. One approach is to require all vendors to re-organize their catalogs in a common format and merge all the catalogs into a central catalog database which allows customers to perform sophisticated searching without dealing with individual catalogs. This requires re-engineering of existing catalogs. In general, vendors want to participate in the central catalog without making changes to their existing catalogs. We study a *virtual catalog* that has the look and feel of a central catalog but holds no physical data. Upon a customer request, this catalog retrieves relevant information from (multiple) individual catalogs and assembles an answer. Such a virtual catalog should satisfy the following requirements: (1) it is up-to-date but does not violate the autonomy of the participating catalogs; (2) its search performance does not degrade as the number of participating catalogs increases; (3) it allows easy inclusion of new catalogs and integrates with other EC applications; and (4) it is easy to construct; assisting tools should be provided.

3. Existing Approaches and Techniques

A few paradigms exist for facilitating integrated access to heterogeneous and autonomous data sources, notably the federated database systems (FDBs) [33] and the mediator systems [37]. More recent approaches handle sources with diverse availability and query processing capability.

Federated databases (FDBs) [4,15,35,1,2,5,12,3] support variations of a five-level extended schema architecture as shown in Figure 1. There are two types of FDBs: the *tightly-coupled* and the *loosely-coupled*. Tightly-coupled systems require the construction of a *global schema* via which queries can be posed. Semantic heterogeneities among participating databases are resolved at the global scheme level. Loosely-coupled systems maintain a collection of possibly inconsistent schemas that are represented uniformly in a canonical data model. The end-user queries these schemas using a *multidatabase query language* such as MSQL [19]. Semantic heterogeneities are resolved at query time.

A mediator is a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications [37]. A mediator system has the form illustrated in Figure 3. A *wrapper* is a special mediator that handles idiosyncrasies of individual data sources. Today, many mediator systems are being built [28,29,26,25,34,30,31,8,9,32,10]. Unlike federated databases that often aim at creating a single database illusion, mediator systems offer specific information services. A mediator system is not monolithic but is often a network of mediators, each accessing multiple heterogeneous data sources and/or other mediators; it often has knowledge-based capabilities. Domain specific mediators may

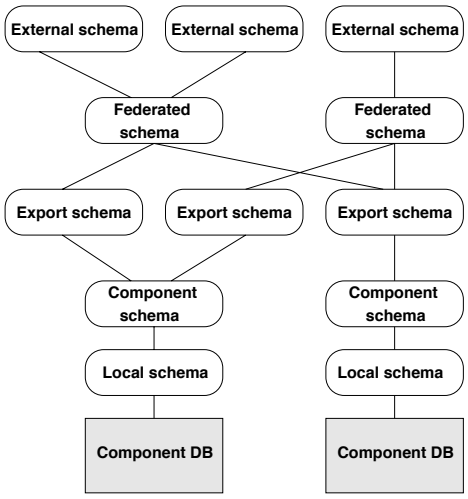

Figure 1: FDB Schema Architecture

exploit domain knowledge to enhance usability and performance.

With the Internet and WWW, more data sources are open with diverse availability and query processing capabilities. A few approaches deal with this situation. DISCO [36] extends the ODMG ODL to allow a bag of extents for a single interface type. Adding a new source means adding an extent into this bag. DIOM [20] intends to identify relevant data sources and bind to them at runtime. The Information Manifold project [18,16,17] considers large number of data sources with varying query capabilities. It assumes the existence of a *worldview*, a pre-defined common application view. A source participating in the worldview is regarded as a *materialized view* of the worldview, with *capability records* attached describing the types of queries it can handle. Adding a new source only means adding a new materialized view. The problem of answering a query against the worldview is transformed to that of answering a query using existing materialized views with additional constraints, this problem is solved [18]. Handling sources with limited query capabilities is especially useful for accessing sources on the Web.

Intelligent mediation techniques [30,10] 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, these techniques can replace mediator authors. AURORA investigates a host of issues in mediator view expression and query processing that are essential even when heterogeneities are detected and resolved automatically.

Kent [11] identifies domain mappings for resolving domain and schema mismatches. Resolutions for individual mismatches are demonstrated using an object-oriented database programming language. However, this work does not provide a mediation methodology, nor does it explore query optimization techniques in presence of the new language constructs. Kim et al. [13] provides a comprehensive classification of

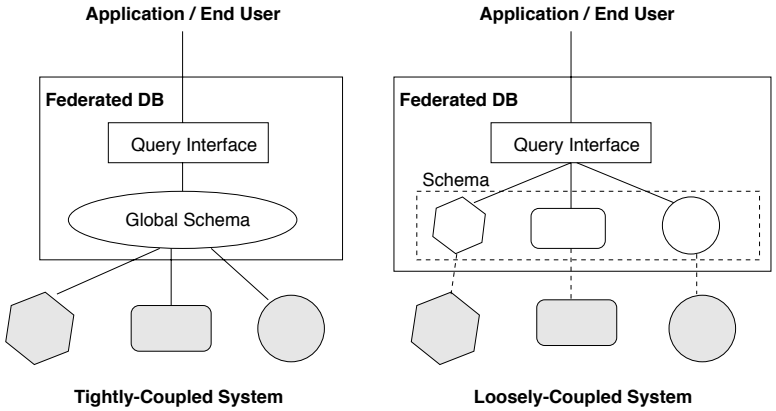

Figure 2: The Federated Database Systems

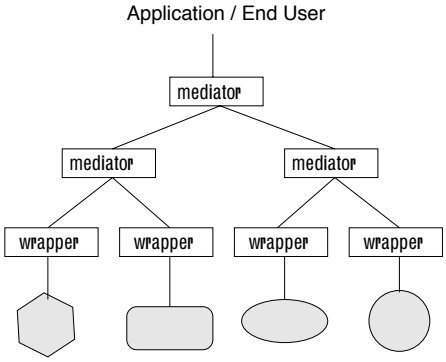

Figure 3: A Mediator System

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. Goh et al. [10] uses ontology to detect and resolve mismatches due to different units of measure. It is not clear how other types of schematic mismatches can be handled.

4. The AURORA Approach

4.1. AURORA Mediation Model

A *mediation model* describes how heterogeneities among data sources are perceived and handled. The AURORA mediation model, shown in Figure 4, is a unique 2-tier model. It models mediation as a 2-step process: *homogenization* followed by *integration*, performed by respective, specialized mediators.

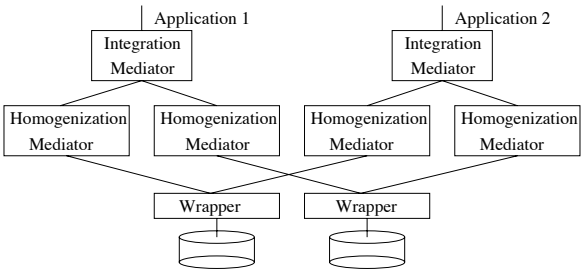

Figure 4: The AURORA Mediation Model

4.1.1. A 2-tier Mediation Model

We distinguish between two categories of heterogeneities among data sources: *schematic mismatches* that arise when the same application domain is modeled differently; and *instance level conflicts* that arise when inconsistent values on the same real world object are recorded. Schematic mismatches must be resolved before instance level conflicts are tackled. The process of resolving schematic mismatches is referred to as *homogenization*. In AURORA, specialized mediators, the *homogenization mediators*, support this process. The result of homogenizing a source is a *homogenizing view* that hides the deviations of this source from the target application view in both structure and semantics. The *integration mediator* "glues" a large number of data sources together by combining all the homogenizing views into an integrated view. Since all the sources are homogenized, the integration process is greatly simplified; it only considers instance level conflicts; schematic heterogeneities have been largely removed by homogenization.

4.1.2. 2-tier Model and Scalability

To include a new data source into the access scope, one must resolve two issues:

1. *Communication.* It must be possible to "talk" to the data source. This is achieved by a wrapper that removes idiosyncrasies of the data source in communication protocol, data model, and query language.

2. *Semantic integration.* It must be possible to construct an integrated view that protect the end-users from the scale and diversity of the access scope.

Scalability requires that both steps be performed rapidly. For 1, this requires rapid, if not automatic, wrapper generation. Various enabling techniques have already been developed [29]. Much is known about *how* 2 can be done, the scalability aspect of it has not been well addressed (ref. Section 4.1.3).

The scalability issue in AURORA is reduced to the issues of rapid homogenization of individual sources and rapid integration of multiple sources. Homogenization can be performed in parallel among sources. It concerns single data sources and has good potential to be fast. AURORA provides tools to assist in this process.

Integration handles multiple but homogeneous (although still autonomous) sources. To achieve scalability of this step, AURORA integration mediator assumes that the integrated view is pre-defined by application requirements. Let this view be V_W. To integrate objects in each data source (it now appears as a homogenization mediator) into V_W, AURORA requires these objects to be *registered* as *fragments* of objects defined in V_W. Removing a source from the access scope only requires the relevant registrations to be cancelled. This way the integration is performed in a plug-and-play fashion.

With many mediation models and architectures already developed, *does the world need another one?* In the next two sections, we answer this question.

4.1.3. Related Mediation Models

With respect to scalability, we distinguish between two types of mediation models: *derivation-based* and *plug-and-play*.

In derivation-based models, a mediator view is *derived* from a set of source descriptions, usually via a monolithic view specification. When sources participate or withdraw from the scope of the mediator, this specification must be modified. This modification may affect many parts of the specification, in the worst case, the specification has to be reconstructed. When the number of participating sources are large, this specification becomes large and complex, difficult to maintain. Most FDBs and mediator systems uses this model.

In plug-and-play models, a common application view is pre-defined. Data sources contribute to and withdraw from this view without affecting other participating sources or the applications that access this view. To our knowledge, two systems use this model: DISCO [36] and Information Manifold (IM) [18].

Apparently, derivation-based models do not favor scalability since including or excluding a source is difficult when the number of sources involved is large. The plug-and-play model avoids this problem. AURORA's mediation model is indeed a plug-and-play model; like those used by DISCO and IM. However, the AURORA model is 2-tier, using two types of mediators, each handling a specific range of heterogeneities, while both DISCO and IM models are 1-tier, using a single mediator to handle the whole range of heterogeneities considered. IM assumes that wrappers handle some mismatches but it is not clear what these mismatches are; IM concentrates on query plan generation. Both DISCO and IM consider a limited range of mismatches; none of them considers instance level conflicts.

4.1.4. Why 2-tier?

A 2-tier model defines a divide-and-conquer approach to information integration. Such an approach facilitates applications such as electronic commerce. It also allows us to better manage the technical complexity in large-scale middleware.

2-tier mediation in EC
Typically, to include a supplier catalog into a virtual catalog, the supplier is first required to map his or her catalog into a format required by the virtual catalog. Essentially, the supplier catalog must be *homogenized* before participating in the virtual catalog. Homogenization is performed by suppliers independently referencing the common catalog format. Individual suppliers are not concerned with inter-catalog conflicts which are resolved at the central catalog level. Often, suppliers are provided with a *workbench* to perform homogenization. This workbench is indeed a homogenization mediator. The central catalog is an integration mediator. A supplier can participate in multiple virtual catalogs requiring varying catalog formats. In this case, the supplier must use multiple homogenization mediators.

Obviously, AURORA's 2-tier model models the activity of constructing virtual catalogs closely. For DISCO or IM to be used in the above scenario, some refinement to their mediation models must be done to clearly define which mismatches are to be resolved by the suppliers independently and which are to be handled at the central catalog level. In general, the 1-tier mediation models do not support application scenarios like the one described above.

Managing technical complexities in large-scale middleware
There are two aspects to this complexity: integration and query processing.

Complexity in integration. When there are 100 sources involving many types of mismatches and conflicts, which one do we resolve first? Can multiple people work on a single integration task? None of the previous models address these issues.

Complexity in query processing. When a large number of highly heterogeneous sources are involved in a query, we face a complex optimization problem that is unknown to traditional data management systems. Query optimization in middleware systems is known as a difficult problem even without considering the scale of the system [21]. In large scale middleware systems such as virtual catalogs in EC, this problem is even more difficult [38].

AURORA's 2-tier model manages both complexities. To manage the complexity of integration, AURORA divides the integration task into two well-defined, smaller tasks, homogenization and integration, that are to be performed by multiple people assisted by specialized tools mandating specific mediation methodologies. AURORA's 2-tier mediation model enables us to decompose the query processing issue into two smaller problems: that in homogenization mediators and that in integration mediators. It is our intension to take advantage of this simplification to develop new techniques in mediator query processing. As shown later, each type of mediator uses a specialized Mediation Enabling Algebra (MEA) to facilitate efficient query processing.

4.2. AURORA Mediators as Distributed Components

AURORA does not restrict the canonical data model to a specific one that is deemed to be most "suitable". Rather, AURORA provides mediators and wrappers

in two popular data models, the relational data model and the ODMG object data model. Necessary facilities are provided to allow the two data models to coexist seamlessly. AURORA mediators are classified along two dimensions: the canonical data model and the mediator type, homogenization or integration. Figure 5 shows this classification.

Mediator Type \ Canonical Data Model	Relational	Object-Oriented
Homogenization	AURORA-RH	AURORA-OH
Integration	AURORA-RI	AURORA-OI

Figure 5: AURORA Mediator Classification

We design and develop three types of software components: the wrappers, the homogenization mediators and the integration mediators. These are provided as distributed components that cooperate via an Object Request Broker (ORB), as shown in Figure 6. AURORA wrappers and mediators support pre-defined interfaces shown in Figure 7.

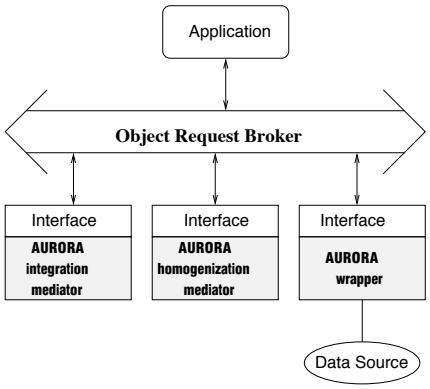

Figure 6: AURORA Components and ORB

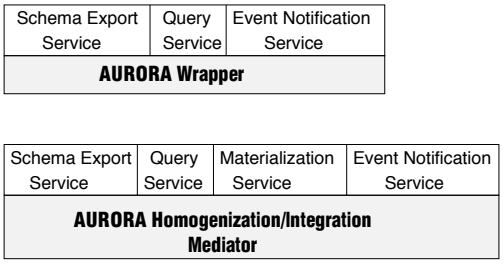

Figure 7: The AURORA Mediator Interfaces

A middleware that facilitates integrated access to multiple heterogeneous data sources can be constructed by using a network of mediators that cooperate with one another to provide an integrated data service. With AURORA, one can choose between relational and object-oriented components. The use of AURORA mediators in building middleware is best illustrated by Figures 8 and 9. AURORA-O mediators have the built-in capability of accessing AURORA-R components, but not vice versa.

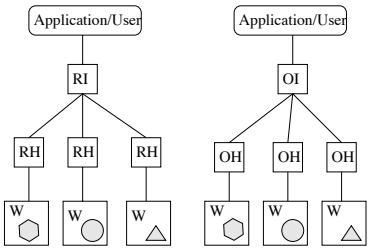

Figure 8: AURORA Application: uniform

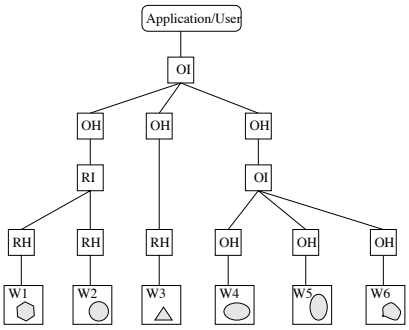

Figure 9: AURORA Application: mixed

4.3. AURORA Mediator Development Workbench

Each AURORA mediator is a *mediator development workbench* consisting of a mediator skeleton and a toolkit named MAT.

Mediator Skeletons. The most important components in a mediator are an integrated view over (multiple) data sources and a query processor that answers queries posed against this view. Building a mediator means building the view, the query processor, and software modules that support other standard services. In AURORA, mediators are constructed from *mediator skeletons* that have all the necessary components of a mediator except for the view.

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. AURORA

MATs assist the mediator authors in performing such tasks. This scenario is shown in Figure 10. A MAT has two main functionalities: it mandates a *mediation methodology* and it provides *Mediation Enabling Operator*s (MEOs).

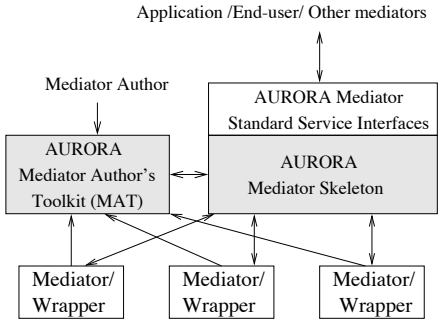

Figure 10: An AURORA Workbench

4.4. Constructing Virtual Catalogs with AURORA

Now we go back to our example of virtual catalogs described in Section 2 and see how the AURORA approach could be used. Consider a number of vendors each having an autonomous catalog database. A virtual catalog effort can be initiated by a third-party broker who seeks to offer value-added services. The broker first designs a common catalog structure, its data model and query language. To include a vendor into the virtual catalog, the broker first homogenizes the vendor's catalog using an AURORA homogenization mediator. This process maps the vendor catalog structure and semantics into those in the common catalog. After homogenization, it should be straightforward to "plug" a catalog into an AURORA integration mediator that supports the common catalog. While homogenization is a more complex process, the broker can hire a few people to homogenize individual vendor catalogs in parallel. An integration mediator is where large number of virtual catalogs merge but the integration is a simple mechanism. Overall, construction of the virtual catalog is scalable.

4.5. Enabling Techniques in AURORA

Each AURORA component requires a suite of enabling techniques. The basis of such a suite is a *Mediation Enabling Algebra*, a MEA, that provides *Mediation Enabling Operators*, MEOs, that are specially designed for manipulation of heterogeneous and autonomous data. Different mediators require different MEAs. Once a MEA is designed, development of the rest of the suite involves the following:

- Design of a MAT. This involves the design of a mediation methodology and the supporting tools. Each tool offers a subset of the MEOs in the MEA.
- Design of a skeleton. This involves (1) developing a query rewriting algorithm

in the MEA; (2) developing query transformation rules in the MEA that potentially allow query optimization; (3) design of a query optimization strategy; and (4) developing techniques for evaluating expensive MEOs efficiently.

The technique suite for AURORA-RH mediator has been developed and is presented in detail in Section 5. This suite sets a paradigm; other suites can be developed in a similar fashion. Work in integration mediator is in early stage and is described in Section 6.

5. Techniques in AURORA-RH

5.1. Homogenization and Query Processing

Let B be a relational database. Let H be a view consisting of relations $M_1, ..., M_n$. 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 11 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*.

5.2. Architecture of AURORA-RH

Figure 12 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 5.4.3-5.4.8). 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**

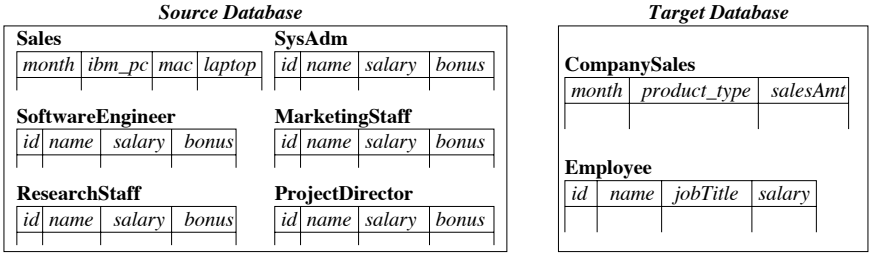

Figure 11: A Homogenization Problem

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.

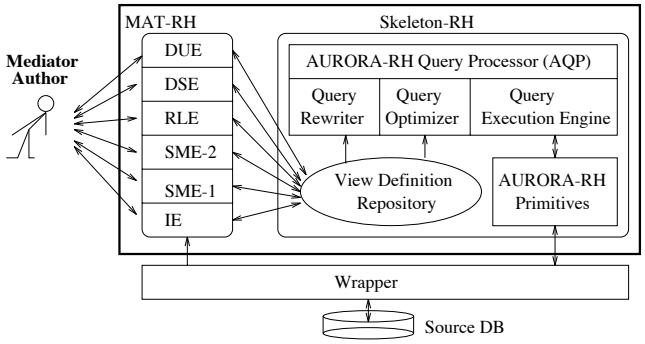

Figure 12: AURORA-RH Workbench

5.3. 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 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 $R' = pad(retrieve(SysAdm), jobTitle, \text{"}SysAdm\text{"})$, where $SysAdm$ is given in Figure 11. 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.

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

defines a relation R', $ATTR(R') = ATTR(R) \cup \{N_1, ..., N_k\}$, with population:
$R' = \{t' \mid t'[N_i] = f_i(t[L_i]), 1 \leq i \leq k;\ t'[A] = t[A], A \in ATTR(R) - \{N_1, ..., N_k\}, t \in R\}$
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 5.4.7 and 5.4.8.

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.

5.4. Mediator Author's Toolkit: MAT-RH

5.4.1. 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 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. Figure 11 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 *Employee*, but as relations in the source.
- (cross-over schema mismatch, type 2.) In the target database, *product types* is represented as data domain *product_type* but as attributes in the source.
- (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.
- (domain unit mismatch.) In the target database, all money amounts use US dollar as unit, while in the source, all money amounts are in Canadian dollars.
- (domain population mismatch.) The two database schemas assume different domain populations of the concept *jobs*.

5.4.2. 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.

5.4.3. Import Environment (IE)

The input to IE (Figure 14) 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. IE allows the mediator authors to choose relations and data of interest from the source. 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* $\}$.

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

Figure 14 depicts Schema Mismatch Environment 1 (SME-1).

Input relations. All relations produced by IE.

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

RELmat(relation materialize) is a special transformation operator. Given $D^R = \{R_1, ..., R_n\}$, 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 13. 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 the group form an enumerated data domain which represents a concept that is represented as relations in the source.

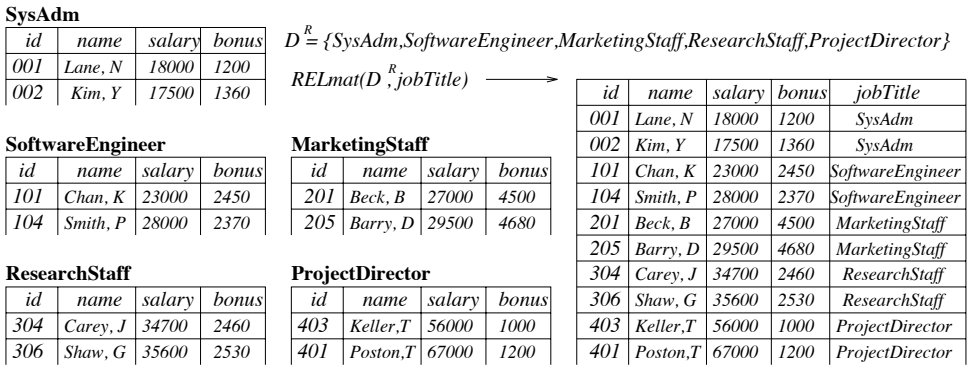

Figure 13: The *RELmat* operator

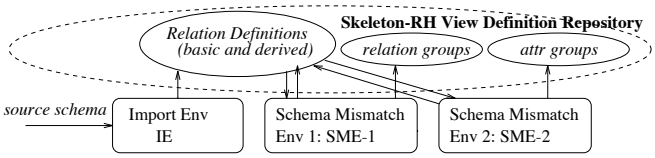

Figure 14: IE, SME-1 and SME-2

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 13 further illustrates this resolution.

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

Schema Mismatch Environment 2 (SME-2) is depicted in Figure 14.

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 $D^A = \{A_1, ..., A_n\}$, 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 15. Application of $ATTRmat$ is demonstrated in Example-4.

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.

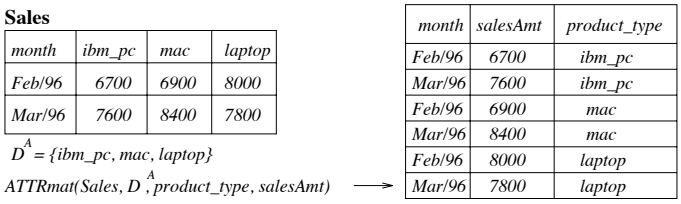

Figure 15: The *ATTRmat* operator

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 15.

5.4.6. RLE: Environment for Relation Linking

Relation Linking Environment, RLE, is depicted in Figure 16. The input includes relations previously defined. Derived relations can be defined using $rename$, π, σ, and $\bowtie$. 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.

5.4.7. DSE: Solving Domain Structural Mismatches

The Domain Structure Environment, DSE (Figure 16), supports resolution of structural mismatches (Section 5.4.1).

Input relations. The distinguished relation, S_M.

Transformation operators. None.

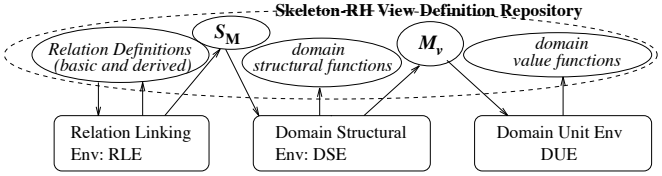

Figure 16: RLE, DSE and DUE

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 $ATTR(M)=\{A_1, ..., A_m\}$. 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. $L_i = \{A^S_{i,1}, ..., A^S_{i,k}\}$, 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:

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

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 be the total income of an employee. The following is a resolution:

$L_{Employee.salary} = \{salary, bonus\}$, $\quad f^s_{Employee.salary}(s, b) = s + b$

5.4.8. DUE: Solving Domain Value Mismatches

The Domain Unit Environment, DUE (Figure 16), supports resolution of domain unit/population mismatches(Section 5.4.1).

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:

$$M = deriveAttr(M_v, A_1, A_1, f_1^v, ..., A_k, A_k, f_m^v)$$

where $ATTR(M) = \{A_1, ..., A_m\}$ 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:

$$f^v_{Employee.salary}(s) = CNDtoUSD(s) = s \times r$$

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

$$f^v_{CompanySales.salesAmt}(s) = CNDtoUSD(s) = s \times r$$

$CNDtoUSD()$ has an inverse, $USDtoCND$.

Example-8: Solving domain population mismatch. $Employee_v$ derived in Example-6 has attribute *jobTitle* but its values are from {*SysAdm, SoftwareEngineer, MarketingStaff, ResearchStaff, ProjectDirector*}. *Employee.jobTitle* consists of {*System Engineer, Development Engineer, Consultant, Research Scientist, Program Manager*}. To resolve this mismatch, a DVF must be specified for *Employee.jobTitle*:

$$f^v_{Employee.jobTitle}(j) = jobMap(j)$$

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

5.5. AURORA-RH Query Processor (AQP)

As shown in Figure 12, AQP consists of a query execution engine, a query rewriter and a query optimizer. Query Execution Plans(QEPs) are expressions that involve only

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

Table 1: *jobMap* for *Employee.jobTitle*

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 17, 18, and 19 are QEP trees. The AQP query execution engine evaluates QEP trees bottom up.

5.5.1. 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 $\{A_1, .., A_m\}$ be all the attributes in L whose domain value functions, $f_1^v, ..., f_m^v$, are *not* identity functions, rewrite Q as:
   $Q = deriveAttr(\pi_L(Q'), A_1, A_1, f_1^v, ..., A_m, A_m, f_m^v)$
3. For each attribute A involved in p, if its DVF, f^v, is not identity, replace it with $f^v(A)$.
4. Let $L = \{A_1, ..., A_n\}$. Let $f_1^s, ..., f_n^s$ be the DSFs of $A_1, ..., A_n$. 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 $deriveAttr(Q', L_1, A_1, f_1^s, ..., L_n, A_n, f_n^v)$
   (b) For $i = 1$ to n do:
   If f_i^s is an identity function, first, remove L_i, A_i, f_i^v from the argument list of *deriveAttr* function constructed above; second, if the argument attribute in L_i, A_i', has a different name from A_i, replace the first argument of *deriveAttr*, E' with $rename(E', A_i', ATTRname(A_i))$.
5. For each attribute A involved in p, if its DSF, f^s, is not identity, replace A with $f^s(A_1', ..., A_k')$, where $A_1', ..., A_k'$ 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 $c' = f^{-1}(c)$. ■

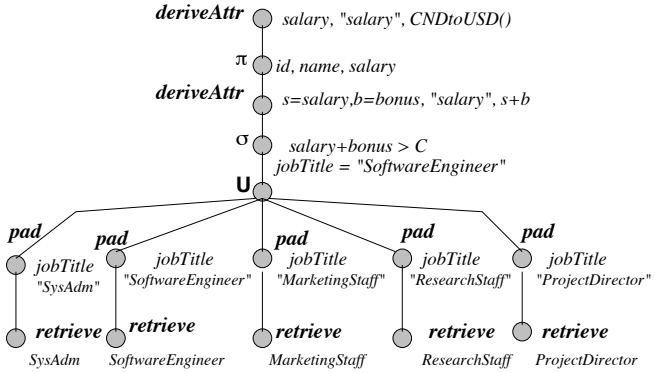

Figure 17: An QEP for Example-9.

Example-9. Consider query

$Q = \pi_{id,name,salary}\sigma_{salary>50000}\sigma_{jobTitle=\text{“}DevelopmentEngineer\text{”}}(Employee)$

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 $S_{Employee}$ (Example-3) and $RELmat$, we get:

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

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

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

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:

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

step 6. $CNDtoUSD$ has an inverse $USDtoCND$, so does $jobMap$(Table 1). Let $C = USDtoCND(50000)$. Performing step 6, we get the final QEP shown in Figure 17:

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

5.5.2. 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 17. 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.

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

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

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 [39]. $p^{N \leftarrow X}$ 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.

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 18. Push σ across *pad* using rule $T_{pad}[3]$, many *pad* subtrees become ϕ, e.g.

$$\sigma_{jobTitle=\text{"}SoftwareEngineer\text{"}}(pad(retrieve(SysAdm),jobTitle,\text{"}SysAdm\text{"}))$$
$$= \sigma_{\text{"}SysAdm\text{"}=\text{"}SoftwareEngineer\text{"}}(pad(retrieve(SysAdm),jobTitle,\text{"}SysAdm\text{"})) = \phi$$

Trim the empty branches from Figure 18 to get Figure 19 (a). Use rule $T_{pad}[1]$ to push π across *pad* to obtain Figure 19 (b). Finally, push the relational operators across *retrieve*. Since the selection predicate involves a function +, known to the source database, both π

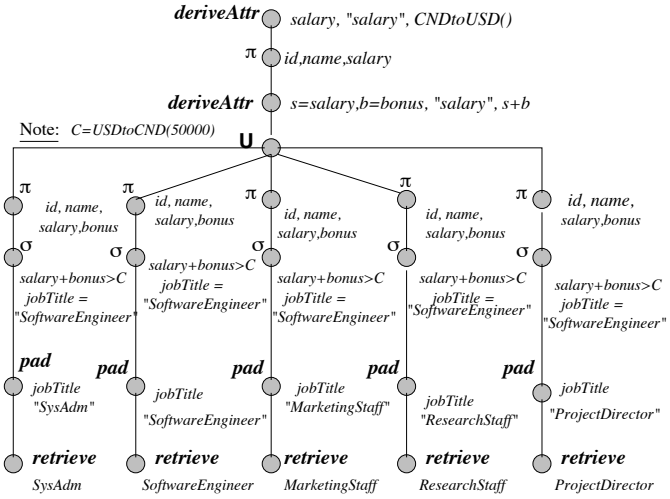

Figure 18: Transformed QEP

and σ exchange with *retrieve* to form Figure 19 (c).

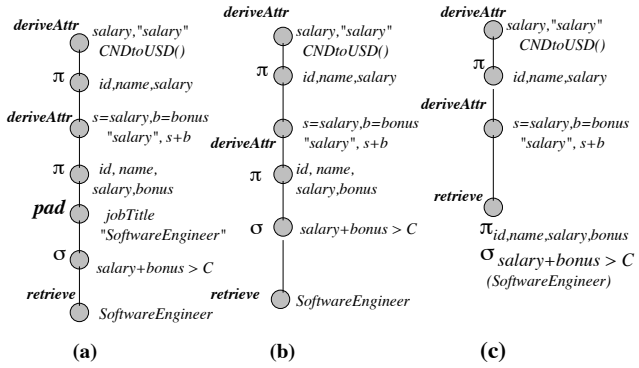

Figure 19: Transformed QEPs

6. Integration Mediators

AURORA integration mediators, AURORA-RI and AURORA-OI (Figure 5), are responsible for integrating a large number of *homogenized* sources. Since the sources are homogenized, the types of heterogeneities that the integration mediator must handle is limited. Lets assume that the application view consists of a single relation R_g. A data source is said to be *homogenized in regard to* R_g if:

1. It is structurally homogenized. It contains a single relation R_s that is a *fragment* of R_g, that is, $ATTR(R_s) \subseteq ATTR(R_g)$.
2. It is semantically homogenized. Each attribute in R_s is the *same* as that in R_g with the same attribute name.

Now the following plug-and-play integration mechanism can be imagined: To "plug" a data source into the the integration mediator, we first homogenize this data source in regard to the application view, that is, construct a *homogenizing view* on top of the data source. We then *register* each relation in this view with the integration mediator as a fragment of a global relation. To "unplug" a data source from the integration mediator, we remove all fragments from this source. When a particular source is down, all fragments from this source are considered to be empty.

At query time, the integration mediators derive the population of global relations and resolve instance level conflicts. We study the following techniques:

1. MEAs for integration. This involves designing new MEOs specially for integration. It also involves development of query modification algorithms and query transformation rules.
2. Efficient evaluation of expensive MEOs in such MEAs. In particular, we explore selectively materializing data to facilitate intelligent query decomposition and join order selection. An initial study along this direction has been done under the name of *Mediator Join Index (MJI)* [40].

7. Conclusion and Future Work

We have described AURORA, a project that develops techniques for building efficient and scalable mediation. Our contributions are the following. First, we have defined a 2-tier, plug-and-play mediation model (Figure 4). Technically, this model enables us to take a divide-and-conquer approach towards building integrated access to heterogeneous sources. Mediation is divided into 2 steps, homogenization followed by integration, and respective mediation methodologies are provided for each step. The general mediator query processing techniques are also divided into two categories: those in homogenization mediators and those in integration mediators. AURORA develops specialized mediation enabling algebras (MEAs) for each category. Second, we have described a complete suite of techniques used by a specific AURORA mediator, AURORA-RH. With this, we have explored the feasibility of our paradigm and are ready to develop technique suites for other AURORA mediators.

Research wise, our goal is to design a collection of mediators, AURORA-RI, AURORA-OH, and AURORA-OI (Figure 5). 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 also be developed.

We plan to implement proof-of-concept wrappers and mediators as distributed components communicating and cooperating via an ORB. Our target application and test-bed is an electronic commerce.

Acknowledgment. The first author would like to thank the Center for Advanced Studies, IBM Canada, for funding my PhD program and for providing a stimulating environment during my field study at IBM Toronto.

References

1. R. Ahmed et al. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, 24(12):19–27, Dec. 1991.
2. Y. Arens, C. Chee, C.-N. Hsu, and C. Knoblock. Retrieving and Integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Informations Systems*, pages 127–158, June 1993.
3. 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.
4. C. Chung. Dataplex: An access to Heterogenous Distributed Databases. *CACM*, 33(1):70–80, Jan. 1990.
5. C. Collet, M. Huhns, and W. Shen. Resource Integration Using a Large Knowledge Base in Carnot. *IEEE Computer*, 24(12):55–62, Dec. 1991.
6. 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.
7. W. Du, R. Krishnamurthy, and M. Shan. Query Optimization in a Heterogeneous DBMS. In *VLDB 92*, pages 277–291, 1992.
8. D. Florescu, L. Raschid, and P. Valduriez. Using Heterogeneous Equivalences for Query Rewriting in Multidatabase Systems. In *CoopIS 95*, 1995.
9. D. Florescu, L. Raschid, and P. Valduriez. Defining the search space for query optimization in a heterogeneous database management system. In *Under Review*, 1996.
10. 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.
11. W. Kent. Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language. In *VLDB 91*, pages 147–160, 1991.
12. W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12):12–18, Dec. 1991.
13. W. Kim et al. On Resolving Schematic Heterogeneity in Multidatabase Systems. *Distributed and Parallel Databases*, 1(3):251–279, 1993.
14. R. Krishnamurthy, W. Litwin, and W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. In *SIGMOD 91*, pages 40–49, 1991.
15. T. Landers and R. Rosenberg. An overview of Multibase. In H. J. Schneider, editor, *Distributed Databases*, pages 153–184. North-Holland, Netherland, 1982.
16. A. Levy. Obtaining Complete Answers from Incomplete Databases. In *VLDB 96*, Bombay, India, Sept. 1996.
17. A. Levy, A. Rajaraman, and J. Ordille. Query Answering Algorithms for Information Agents. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI-96,*, Portland, Oregon, Aug. 1996.
18. A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB 96*, Bombay, India, Sept. 1996.
19. W. Litwin et al. MSQL: A multidatabase Language. *Information Sciences*, 49(1-3):59–101, Oct. 1990.
20. 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.
21. H. J. Lu, B. C. Ooi, and C. H. Goh. On Global Multidatabase Query Optimization. *ACM SIGMOD Record*, 21(4):6–11, Dec. 1992.
22. W. Meng et al. Construction of Relational Front-end for Object-Oriented Database

Systems. In *ICDE 93*, pages 476–483, 1993.
23. R. Miller, Y. Ioannidis, and R. Ramakrishnan. The Use of information capacity in schema integration and translation. In *VLDB 93*, pages 120–133, 1993.
24. A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Trans. on Software Engineering*, SE-13(7):785–798, July 1987.
25. Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In *VLDB 96*, Bombay, India, Sept. 1996.
26. Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A Mediation System Based on Declarative Specifications. In *ICDE 96*, pages 132–141, New Orleans, Feb. 1996.
27. Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. In *ICDE 96*, 1996.
28. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *ICDE 95*, pages 251–260, Taipei, Taiwan, Mar. 1995.
29. Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A Query Translation Scheme for Rapid Implementation of Wrappers. In *International Conference on Deductive and Object-Oriented Databases*, 1995.
30. 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.
31. L. Raschid, Y. Chang, and B. Dorr. Query transformation techniques for interoperable query processing in cooperative information systems. In *CoopIS 94*, 1994.
32. E. Sciore, M. Siegel, and A. Rosenthal. Using Semantic values to facilitate interoperability among heterogeneous information systems. *ACM TODS*, 19(2):254–290, June 1994.
33. A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), Sept. 1990.
34. V. S. Subrahmanian et al. HERMES: Heterogeneous Reasoning and Mediator System. Unpublished document, University of Maryland.
35. M. Templeton, H. Henley, E. Maros, and D. V. Buer. InterViso: Dealing With the Complexity of Federated Database Access. *VLDB Journal*, 4(2):287–317, 1995.
36. 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.
37. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, pages 38–49, Mar. 1992.
38. L. L. Yan. Building Scalable and Efficient Mediation: the AURORA Approach. *PhD thesis in preparation*, 1997.
39. 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.
40. L. L. Yan, T. Ozsu, and L. Liu. Mediator Join Indices. In *Seventh International Workshop on Research Issues in Data Engineering: High-Performance Database Management for Large Scale Applications (RIDE'97)*, Birmingham, England, Apr. 1997.
41. C. Yu et al. Translation of Object-Oriented Queries to Relational Queries. In *ICDE 95*, pages 90–97, Taipei, Taiwan, Mar. 1995.